Raphael - Challenge ServeRest

Plano de Teste

Apresentação

- Este documento é destinado no detalhamento da estratégia de testes para a API ServeRest, u sistema que simula uma aplicação de E-coormence para fins de treinamento e testes. Este Documento está dividido em seções. O objetivo define o proposito do documento. O resumo define um contexto geral do sistema que será testado.

Objetivo

- Garantir a qualidade do e-commerce ServeRest por meio de testes que garantem que os endpoints estejam funcionais garantindo a seguridade da aplicação.

Resumo

- O ServeRest é um E-cormmece genérico com os usuário de administrador, administrador do sistema, e usuário comum, usuário-*alvo da plataforma. Esse planejamento de teste será realizado por conta de problemas de ferramentas do sistema.
- A hipótese deste teste é que os endpoints estejam todos corretos e funcionais.

Pessoas Envolvidas:

- Raphael Sousa Rabelo Rates

Escopo

- Endpoints de teste:

ServeRest login

ServeRest usuários

ServeRest produtos

ServeRest carrinhos

Todas as rotas tiveram os seguintes testsesudados

Testes Unitários

Testes de acordo com a validação do retorno dos dados esperados de cada regra de negócio para cada rota da aplicação

Desempenho

Verificação do tempo de resposta para cada Rota

Regressão

Teste regressivo ara todas as rotas em apenas um Run, verificando os fluxos do sistema para verificar se não tem nenhuma rota com bugs.

Análise

- **Endpoints críticos**: login, cadastro de usuários, cadastro de vendedores (admin), criação do produto e finalização do produto.

Técnicas Aplicadas

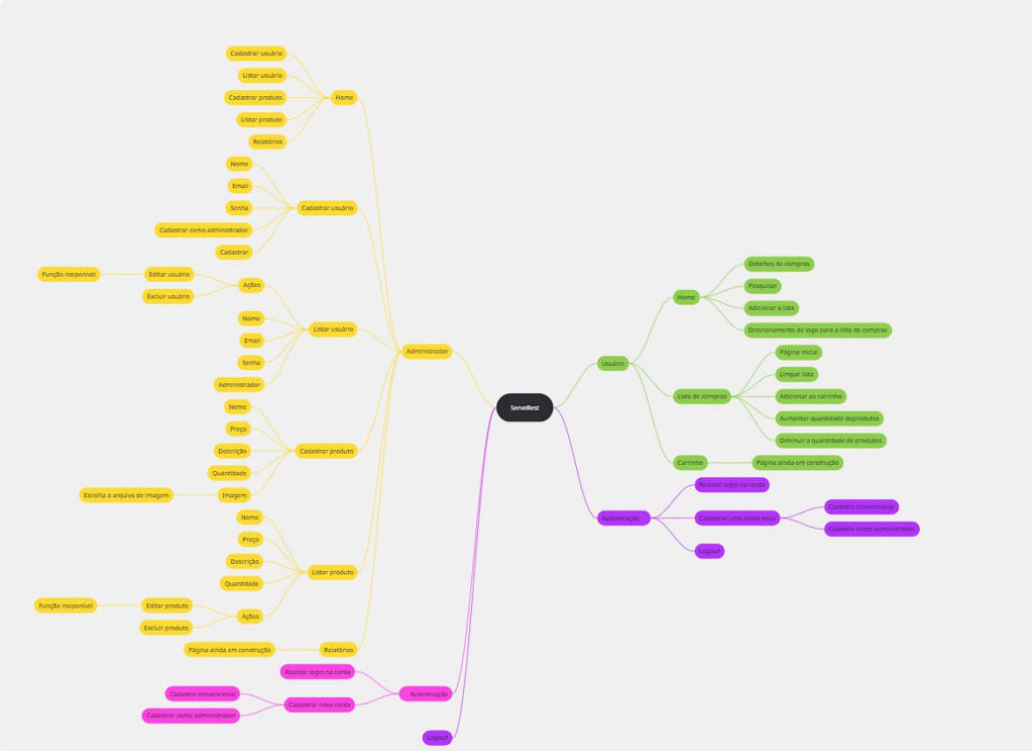
- Testes de API: usando scripts do postman para verificar as regras de negocio de cada rota
- Testes de automação: Usando flows automatizados com Monitores para verificação automática.

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();
});
```

2. /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);
});
```

3. /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);
});
```

4. /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 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
});
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);
});
```

5. /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") || "BeeJh5lz3k6kSIzA";
        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");
        });
    }
}
```

6. /carrinhos

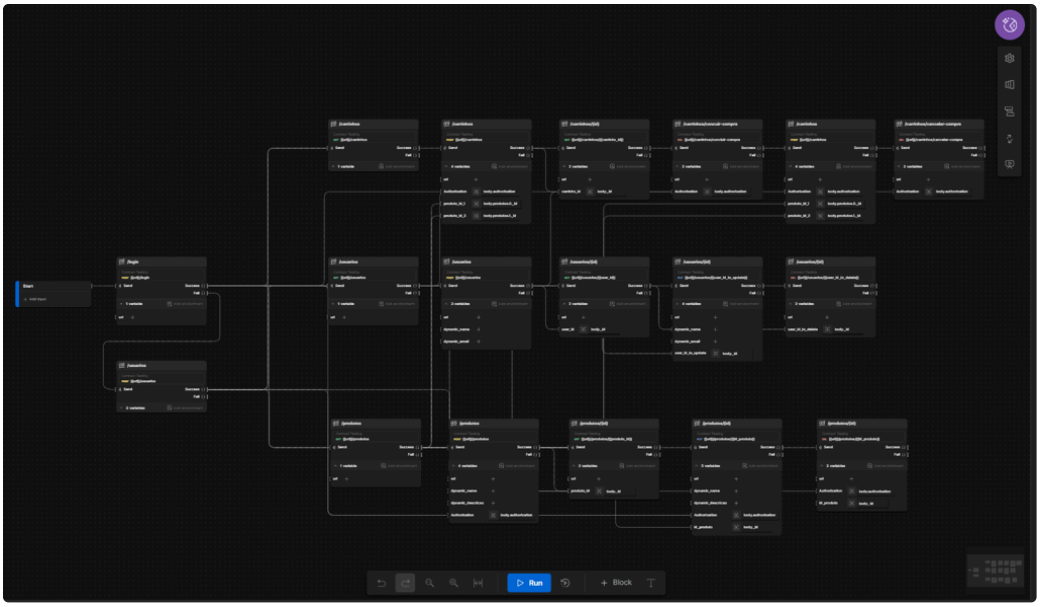
```
pm.test("Status 200 - Lista de carrinhos retornada", function () {
    pm.expect(pm.response.code).to.eql(200);
});
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});
pm.test("Estrutura JSON válida", 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);
                });
            }
        });
```

7. /carrinhos/{id}

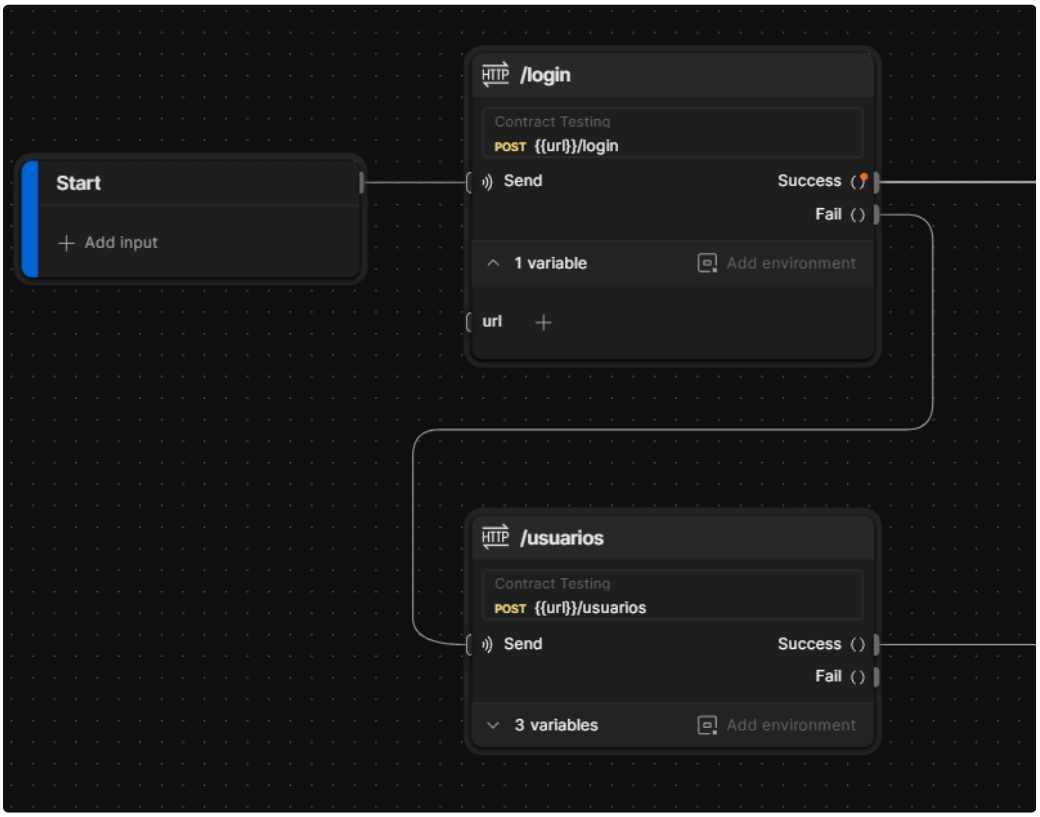
```
if (pm.response.code === 200) {
    const response = pm.response.json();
    if (response.message === "Carrinho não encontrado") {
        pm.test("Carrinho não encontrado", function () {
            pm.expect(response.message).to.eql("Carrinho não encontrado");
        });
        pm.test("Resposta 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);
        });
    }
}
```

Regressão

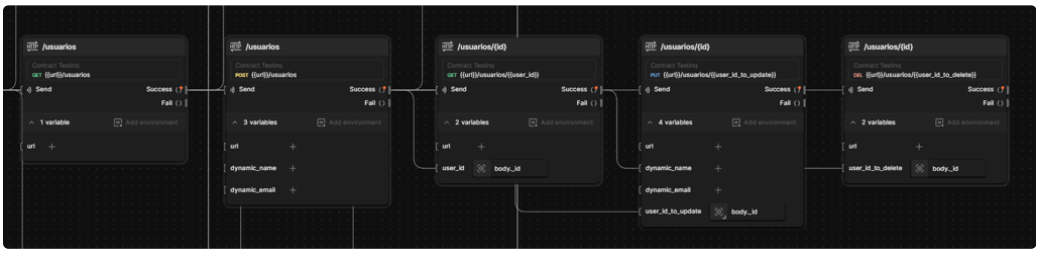
Todos os testes por rotas foram automatizados através das ferramentas propostas para a análise do fluxo geral do sistema no teste de regressão. O Teste automatizado possui o seguinte fluxo.



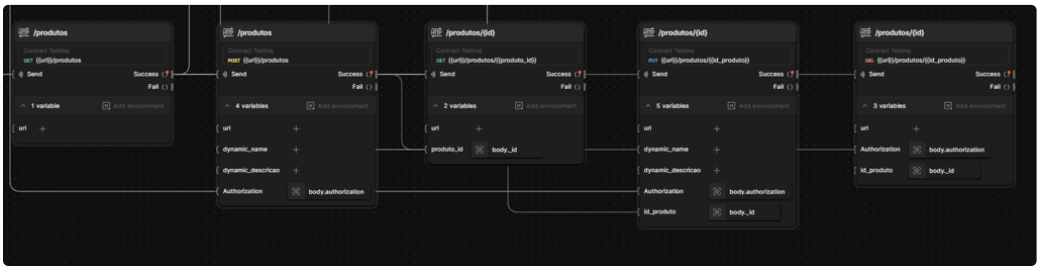
1. Criação de Um login com usuário para permissões de administrador



2. Fluxo dos usuários

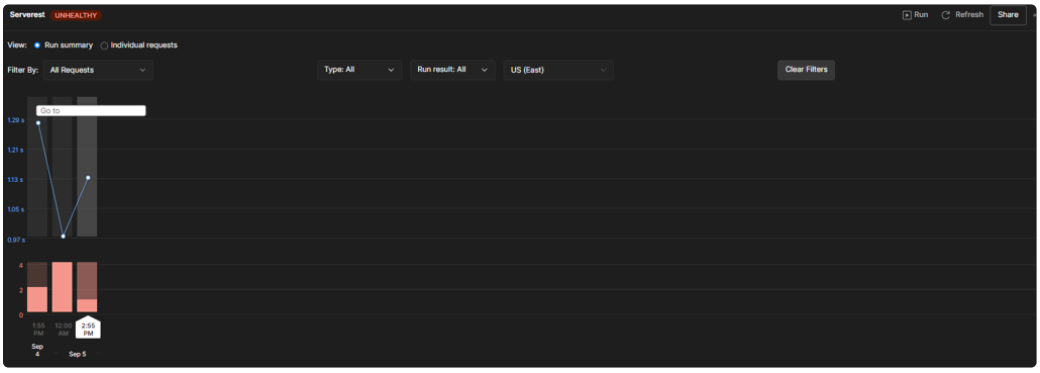


3. Fluxo dos Produtos

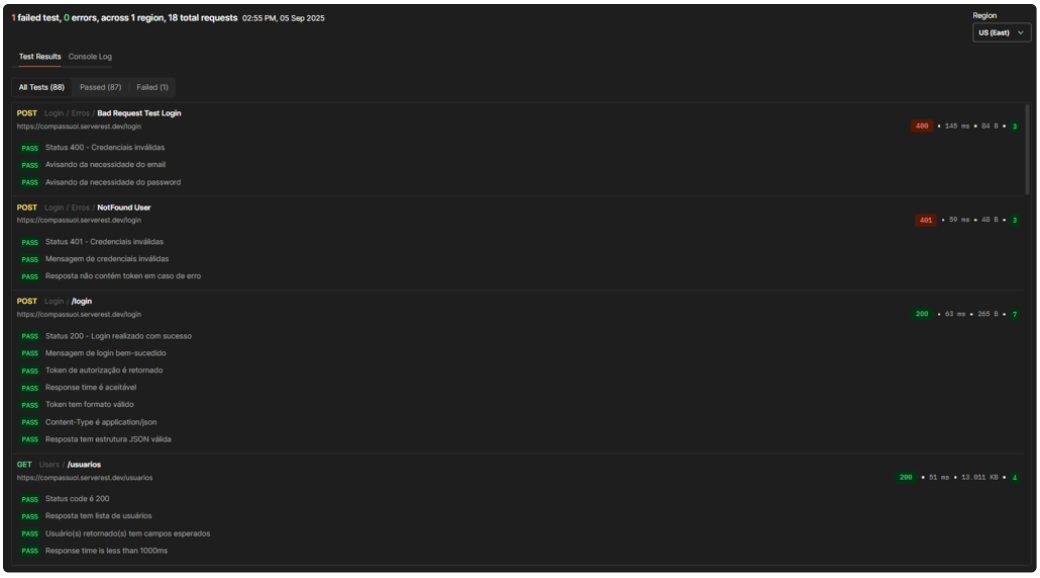


4.Fluxo do Carrinho

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.


1 failed test, 0 errors, across 1 region, 18 total requests 02:55 PM, 05 Sep 2025
Region
US (East)
Test Results
Console Log
All Tests (88)
Passed (87)
Failed (1)
POST Login / Erros / Bad Request Test Login
https://compassuol.serverest.dev/login
400 • 145 ms • 84 B • 3
PASS Status 400 - Credenciais inválidas
PASS Avisando da necessidade do email
PASS Avisando da necessidade do password
POST Login / Erros / NotFound User
https://compassuol.serverest.dev/login
401 • 59 ms • 48 B • 3
PASS Status 401 - Credenciais inválidas
PASS Mensagem de credenciais inválidas
PASS Resposta não contém token em caso de erro
POST Login / /login
https://compassuol.serverest.dev/login
200 • 63 ms • 265 B • 7
PASS Status 200 - Login realizado com sucesso
PASS Mensagem de login bem-sucedido
PASS Token de autorização é retornado
PASS Response time é aceitável
PASS Token tem formato válido
PASS Content-Type é application/json
PASS Resposta tem estrutura JSON válida
GET Users / /usuarios
https://compassuol.serverest.dev/usuarios
200 • 51 ms • 13.011 KB • 4
PASS Status code é 200
PASS Resposta tem lista de usuários
PASS Usuário(s) retornado(s) tem campos esperados
PASS Response time is less than 1000ms