Documentação: Testes de API

Introdução

Os testes de API têm como objetivo validar os endpoints disponíveis, garantindo que a comunicação entre o cliente e o servidor ocorra corretamente. Esses testes verificam:

- Autenticação e autorização.

- Validação de entradas e saídas.

- Funcionamento de rotas protegidas e públicas.

- Comportamento em cenários de erro.

Ferramentas Utilizadas

- Postman: Para executar e validar requisições HTTP manuais.

- Cypress: Para automatizar testes de API.

- Servidor: ServeRest API (`http://localhost:3000`).

Configuração Inicial

1. Executar o servidor ServeRest :

- Navegue até o diretório do projeto:

```bash

cd "C:\Users\allan\Downloads\Projetos Tivia TI\ServeRest-trunk\ServeRest-trunk\src"

```

- Inicie o servidor:

```bash

node server.js

```

2. Configurar Postman :

- Crie uma coleção de testes com os endpoints usados.

- Configure os cabeçalhos e o corpo da requisição conforme necessário.

3. Instalar dependências para testes automatizados :

- Execute:

```bash

npm install cypress --save-dev

```

Endpoints Testados

# 1. Autenticação - POST /login

- Descrição: Autentica um usuário no sistema, retornando um token de autorização.

- Requisição:

- Body:

```json

{

"email": "usuario@qa.com",

"password": "teste"

}

```

- Headers:

```

Content-Type: application/json

```

- Resposta esperada:

- Status: `200 OK`

- Body:

```json

{

"message": "Login realizado com sucesso",

"authorization": "eyJhbGc...<token>"

}

```

# 2. Listar Usuários - GET /usuarios

- Descrição: Retorna uma lista de usuários cadastrados.

- Requisição:

- Headers:

```

Authorization: Bearer <seu\_token>

```

- Resposta esperada:

- Status: `200 OK`

- Body: JSON com detalhes dos usuários cadastrados.

# 3. Cadastrar Usuário - POST /usuarios

- Descrição: Adiciona um novo usuário ao sistema.

- Requisição:

- Body:

```json

{

"nome": "Fulano de Tal",

"email": "fulano@qa.com",

"password": "senha123",

"administrador": "true"

}

```

- Headers:

```

Content-Type: application/json

Authorization: Bearer <seu\_token>

```

- Resposta esperada:

- Status: `201 Created`

- Body:

```json

{

"message": "Cadastro realizado com sucesso",

"id": "<novo\_id\_usuario>"

}

```

---

# 4. Produtos - GET /produtos

- Descrição: Retorna os produtos cadastrados.

- Requisição:

- Headers: (Token necessário)

```

Authorization: Bearer <seu\_token>

```

- Resposta esperada:

- Status: `200 OK`

- Body: Lista de produtos em formato JSON.

---

Cenários de Testes

1. Login com Credenciais Válidas

- Expectativa: Retorna token de autorização.

2. Login com Credenciais Inválidas

- Expectativa: Retorna erro `401 Unauthorized`.

3. Listagem de Usuários Sem Token

- Expectativa: Retorna erro `403 Forbidden`.

4. Cadastro de Usuário Já Existente

- Expectativa: Retorna erro `400 Bad Request`.

5. Adição de Produto Sem Autorização

- Expectativa: Retorna erro `401 Unauthorized`.

---

Automatização de Testes com Cypress

Crie arquivos de teste no Cypress para validar os endpoints. Exemplo de teste para `/login`:

```javascript

describe('Teste de API - Autenticação', () => {

it('Deve autenticar com sucesso', () => {

cy.request('POST', 'http://localhost:3000/login', {

email: 'usuario@qa.com',

password: 'teste'

}).then((response) => {

expect(response.status).to.eq(200);

expect(response.body).to.have.property('authorization');

});

});

});

```

---

Boas Práticas

1. Organizar Testes: Separe os testes em categorias (autenticação, usuários, produtos, etc.).

2. Gerenciar Tokens: Armazene tokens em variáveis para reutilização em diferentes testes.

3. Simular Cenários de Erro: Valide como a API responde a dados incorretos.

4. Utilizar Variáveis de Ambiente: Configure URLs e dados dinâmicos no Postman ou Cypress para facilitar mudanças.

Conclusão

Os testes de API garantem a integridade das rotas e ajudam na identificação de problemas antes da entrega final. Com essa documentação, você tem um guia completo para configurar, testar e automatizar.