Sistema de Testes de API - Documentação Técnica

Esta documentação técnica apresenta um sistema abrangente de testes de API, desenvolvido para facilitar a validação e simulação de integrações de sistemas. O sistema oferece uma plataforma robusta e flexível que permite aos desenvolvedores e equipes de QA executarem testes controlados, simularem diversos cenários de resposta e monitorarem o comportamento das APIs em tempo real.

Com uma interface intuitiva e recursos avançados de monitoramento, o sistema é ideal para equipes que necessitam realizar testes de integração, validar comportamentos de API e garantir a qualidade das integrações em seus projetos.

Desenvolvido por:

- Fabio Henrique de Souza Venâncio Pinheiro
- Contato: (11) 95818-4521
- E-mail: fabiohvp2012@gmail.com

Sumário

Visão Geral

O sistema consiste em um dashboard interativo para testes de API, projetado para simular diferentes cenários de integração. O projeto implementa um conjunto de APIs RESTful com comportamentos controlados, permitindo testar diversos cenários de sucesso e falha.

Características Principais

- Dashboard interativo com feedback visual
- Dois modos de visualização (Cliente e Monitor)
- Simulação de cenários de erro controlados
- Fluxos de teste encadeados
- Interface responsiva com Bootstrap

Tecnologias Utilizadas

Camada	Tecnologias
Frontend	TypeScript, Bootstrap 5, HTML5
Backend	Node.js, Express.js
Ferramentas	TypeScript Compiler, Nodemon, CORS

Arquitetura

Estrutura do Projeto

```
projeto/
├── src/
│   ├── client/
│   │   └── app.ts          # Lógica principal do cliente
│   └── server/
│       └── server.ts       # Implementação das APIs
├── public/
│   ├── index.html         # Interface do usuário
│   ├── styles.css         # Estilos da aplicação
│   └── app.js            # JavaScript compilado
└── docs/
    └── documentacao_tecnica.md
```

Componentes Principais

ApiTestManager - Classe central responsável pelo gerenciamento dos testes e interface com usuário.

```
class ApiTestManager {
  private baseUrl: string;
  private isMonitorMode: boolean;
  private resultadosElement: HTMLElement;
  private statusIndicator: HTMLElement;
  private usuarioId: number | null;

  constructor() {
    this.inicializarComponentes();
    this.configurarModoVisualizacao();
    this.registrarEventos();
  }
}
```

APIs Implementadas

1. Cadastro de Usuário [POST /api/cadastro]

Entrada:

```
{
  nome: string,
  email: string,
  senha: string,
  origem: string
}
```

- **Processamento:** Gera ID aleatório (10000-99999)
- **Saída:** `{ usuarioId: number }`
- **Validações:** Todos os campos são obrigatórios

2. Login [GET /api/login]

- **Entrada:** usuariold (query parameter)
- **Regra de Negócio:** Falha para IDs múltiplos de 5
- **Sucesso:** `{ mensagem: string }`
- **Erro:** Erro aleatório do sistema

3. Alteração de Dados [PUT /api/alteracao]

- **Entrada:** usuariold (query parameter)
- **Regra de Negócio:** Falha para IDs múltiplos de 3
- **Taxa de Sucesso:** 100% para IDs não múltiplos de 3

4. Listagem de Pedidos [GET /api/pedidos]

- **Entrada:** usuariold (query parameter)
- **Taxa de Sucesso:** 90%
- **Saída:** Lista de pedidos fictícios com data e valor

Formato de Resposta:

```
{
  pedidos: Array<{
    id: number,
    valor: number,
    data: string
  }>
}
```

Modos de Visualização

Modo Cliente

- Visualização simplificada
- Dados sensíveis ocultados
- Mensagens amigáveis ao usuário
- Interface limpa e objetiva

```
if (!this.isMonitorMode) {
  this.adicionarResultado('Login Realizado com Sucesso', {
    status: response.status,
    mensagem: 'Autenticação realizada com sucesso'
  });
}
```

Modo Monitor

- Visualização técnica detalhada
- Exibição de payloads completos
- IDs e dados sensíveis visíveis
- Informações de debug disponíveis

```
if (this.isMonitorMode) {
  this.adicionarResultado('Login Realizado com Sucesso', {
    status: response.status,
    data: data,
    usuarioId: usuarioId
  });
}
```

Fluxos de Teste

1. Fluxo de Cadastro

```
Cadastro → Validação → Resposta
```

2. Fluxo de Login

```
Cadastro → Login → Validação
```

3. Fluxo de Edição

```
Cadastro → Login → Alteração → Validação
```

4. Fluxo de Listagem

```
Cadastro → Login → Alteração → Listagem → Validação
```

Segurança e Tratamento de Erros

Sistema de Erros

O sistema implementa um mecanismo de erros aleatórios para simular falhas reais:

```
const erros = [
  { status: 504, mensagem: 'Gateway Timeout' },
  { status: 401, mensagem: 'Unauthorized' },
  { status: 400, mensagem: 'Bad Request' },
  { status: 500, mensagem: 'Internal Server Error' }
];
```

Medidas de Segurança

Filtragem de Dados Sensíveis

- IDs de usuário ocultados
- Payloads filtrados
- Mensagens de erro sanitizadas

Validações de Entrada

- Verificação de campos obrigatórios
- Validação de tipos de dados
- Sanitização de parâmetros

Guia de Instalação e Execução

Pré-requisitos

- Node.js (versão 14 ou superior)
- npm (gerenciador de pacotes)

Instalação

```
# Instalar dependências
npm install
```

Execução

```
# Iniciar em modo de desenvolvimento
npm run monitor
```

Acessando a Aplicação

Modo	URL	Descrição
Cliente	http://localhost:3000	Visualização simplificada
Monitor	http://localhost:3000?mode=monitor	Visualização técnica detalhada

Scripts Disponíveis

- `npm run build` : Compila o TypeScript
- `npm run dev` : Inicia o servidor de desenvolvimento
- `npm run monitor` : Inicia com modo monitor ativado

Considerações de Desenvolvimento

Boas Práticas Implementadas

- Tipagem forte com TypeScript
- Tratamento consistente de erros
- Separação clara de responsabilidades
- Código modular e reutilizável
- Documentação inline