# Manual de execução Desafio API Bancária

Autor: Oswaldo Beltrani Neto

Este projeto tem como objetivo desenvolver uma API para gerenciamento de dados bancários.

Sua estrutura está composta da seguinte forma:

**Back-end:** A API é construída em **Java** utilizando o **framework Spring Boot** e a arquitetura **REST**, com **Maven** para gerenciamento de dependências. A documentação e os testes da API são realizados com **Swagger**, e os dados são armazenados em um banco **SQL Server**.

**Front-end:** A interface de usuário é desenvolvida com **React**, enquanto a biblioteca **Axios** é utilizada para o envio de requisições HTTP (GET, POST, DELETE e PUT) à API, garantindo a integração entre front-end e back-end.

## ▼ ATENÇÃO!

Para garantir o funcionamento desta API é necessário já conter instalado e configurado previamente em sua máquina:

- Java JDK 17
- SQL Server Management Studio 21
- Maven 3.9.11
- Node 18.17.1

## ▼ Primeiro passo

Primeiro, clone o repositório disponibilizado em https://github.com/Netu0/desafioAPI.git através do comando:

```
git clone https://github.com/Netu0/desafioAPI.git
```

## ▼ Segundo passo

Após clonar o projeto, é necessário instalar algumas bibliotecas e funcionalidades que serão utilizadas dentro do trabalho:

```
cd frontend
npm install axios
cd ..
```

## ▼ Terceiro passo

Crie um novo usuário no ambiente do banco, com login e senha de sua escolha.

Em seguida, Executar o Script de criação do banco de dados dentro do ambiente SQL Server Management Studio 21:

```
CREATE TABLE TB_PESSOA (
    idPessoa BIGINT IDENTITY(1,1) PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    cpf VARCHAR(14) UNIQUE NOT NULL,
    dataNascimento DATE NOT NULL
);

CREATE TABLE TB_CONTA (
    idConta BIGINT IDENTITY(1,1) PRIMARY KEY,
    idPessoa BIGINT NOT NULL,
    saldo DECIMAL(15,2) NOT NULL,
    limiteSaqueDiario DECIMAL(15,2),
    flagAtivo CHAR(1) NOT NULL,
    tipoConta INT NOT NULL,
    dataCriacao DATE NOT NULL,
    CONSTRAINT FK_CONTA_PESSOA FOREIGN KEY (idPessoa) REFERE
```

```
NCES TB_PESSOA(idPessoa)
);

CREATE TABLE TB_TRANSACAO (
    idTransacao BIGINT IDENTITY(1,1) PRIMARY KEY,
    idConta BIGINT NOT NULL,
    valor DECIMAL(15,2) NOT NULL,
    dataTransacao DATE NOT NULL,
    TIPO VARCHAR(10) NOT NULL COLLATE Latin1_General_CI_AS,
    CONSTRAINT FK_TRANSACAO_CONTA FOREIGN KEY (idConta) REFE
RENCES TB_CONTA(idConta)
);
```

## ▼ Quarto passo

Rodar o Back-end através do código da api.

```
cd desafio
mvn spring-boot:run
cd ..
```

Nesta etapa, é possível acessas o Swagger através do link: http://localhost:8080/swagger-ui/index.html#/, possibilitando a visualização da documentação de controladores da API, DTOs e testes.

Também é possível acessar essas funcionalidades através de links https, como:

- GET todas pessoas: http://localhost:8080/pessoas
- GET pessoas por id: http://localhost:8080/pessoas/{id}
- GET contas por id: http://localhost:8080/contas/{id}
- GET saldo de contas por id: http://localhost:8080/contas{id}/saldo
- GET transações de uma conta por id: http://localhost:8080/transacoes/{idConta}/extrato

As outras funcionalidades também contém paths (POST, PUT…) entretanto, essas precisam de um corpo em JSON, para isso, o

Swagger está implementado.

## ▼ Quinto passo

Por fim, basta executar o front end e utilizar sua interface de usuário.

```
cd frontend
npm start
```

Com a interface já aberta, é possível navegar através da Header entre as funcionalidades CONTAS e TRANSAÇÕES, consumidas diretamente da API.

Em contas, ao ser informado um ID válido, terá como resposta os dados referentes a esta conta.

Em Transações, é possível visualizar o extrato geral de uma conta ou entre duas determinadas datas.