**NTTBANK API**

v0.1.0

Projeto: Curso Java Intermediário – Alura

Responsável Técnico: João Guilherme Hagemann Seleme

Github: <https://github.com/jghagemann/nttbank>

**CONTROLE DE VERSÕES**

|  |  |  |
| --- | --- | --- |
| **Data** | **Versão API** | **Responsável** |
| 04/12/2024 | 0.1.0 | João Guilherme Hagemann Seleme |

**CHANGELOG**

**0.1.0:**

* Versão inicial.

Sumário

[**1.** **Introdução** 5](#_Toc184207223)

[**2.** **Ferramentas utilizadas** 5](#_Toc184207224)

[**3.** **Descrição técnica** 6](#_Toc184207225)

[**3.1** **Arquitetura** 6](#_Toc184207226)

[**3.2** **Descrição dos pacotes** 6](#_Toc184207227)

[**4.** **Configurando e rodando o projeto** 7](#_Toc184207228)

1. **Introdução**

O projeto tem o objetivo incentivar o treinamento de profissionais voltados ao *backend* no desenvolvimento de APIs seguindo as boas práticas de programação utilizando ferramentas em demanda no mercado.

A API criada destina-se a simular a criação de usuários e contas, manter um histórico de transferências, efetuar carga de usuários via Excel, download de um relatório de transferências e exibição de um gráfico para análise de despesas.

Além disso, conta com integrações feitas em uma API *Mock* simulando o saldo em conta do usuário em uma ou mais contas em moedas diferentes, e conversão de despesas para o Euro através de integração com API externa.

1. **Ferramentas utilizadas**

* Java 21
* SpringBoot 3.4.0
* PostgreSQL 17
* Docker
* JPA / Hibernate ORM/Validator
* SpringDoc OpenAPI para documentação
* Spring Security
* Auth0 java-jwt para geração de token
* WebClient para chamadas à APIs
* Apache POI para Excel
* Apache PDFBox para PDF
* JFreeChart para gráficos

1. **Descrição técnica**

## **Arquitetura**

Foram desenvolvidas camadas de *Domain,* *Controllers, Services, Repositories* e *Clients*, além disso foram criados pacotes *Config, Infra/Security, Helper, Handler e Exception.*

## **Descrição dos pacotes**

**Domain:** Contém classes que através do Hibernate compõe as tabelas e relacionamentos do Banco de dados e algumas validações de *constraints*

**Controllers:** Responsável por receber o conteúdo das chamadas feitas pelo front-end e direcionar para a camada de *Service.*

**Services:** Responsável por aplicar as regras de negócio.

**Repositories:** Responsável por realizar a comunicação com o banco de dados, inserção, recuperação, atualização ou exclusão de tabelas.

**Clients:** Responsável por montar e realizar chamadas para APIs externas que foram integradas ao projeto, retornando o resultado para a camada de *Service.*

**Config:** Possui classes de configuração para a interface web do SpringDoc OpenAPI.

**Infra/Security:** Possui classes de configuração responsável pela Segurança do acesso aos *endpoints* da aplicação e pela geração do Token JWT.

**Helper:** Possui classes de geração de arquivos PDF, Excel e gráfico de despesas utilizados na aplicação.

**Handler:** Possui classe de GlobalExceptionHandler responsável por direcionar o tratamento de exceções.

**Exception:** Possui classes de Exceção utilizadas na aplicação.

1. **Configurando e rodando o projeto**

Será necessário criar um arquivo *.env* na raíz do projeto com as seguintes variáveis:

* SPRING\_DATASOURCE\_URL
* SPRING\_DATASOURCE\_USERNAME
* SPRING\_DATASOURCE\_PASSWORD
* POSTGRES\_DB
* POSTGRES\_USER
* POSTGRES\_PASSWORD
* EXCHANGE\_RATES\_API\_KEY

1. Baixar o projeto do repositório no github
2. Realizar o build do projeto com o maven através da IDE ou linha de comando executando:

***mvn clean install***

1. Realizar o build da imagem do Docker executando:  
   **docker compose build**
2. Subir os containers docker executando:  
   **docker compose up**
3. Conferir se o projeto iniciou visitando a página da documentação OpenAPI:  
   [*http://localhost:8080/swagger-ui/index.html*](http://localhost:8080/swagger-ui/index.html)