Documentação Técnica – Integração com HubSpot

1. Visão Geral do Sistema

O sistema baseia-se em uma API REST desenvolvida em Java e Spring Boot, com o objetivo de integrar com a API do HubSpot, utilizando autenticação OAuth 2.0 com o fluxo Authorization Code**.**

2. Endpoints Implementados

- *GET /oauth/authorize*

Gera e retorna a URL de autorização do HubSpot, utilizada para iniciar o fluxo de autenticação.

- *GET /oauth/call-back*

Processo o callback do OAuth com o código de autorização e troca por um token de acesso, utilizado posteriormente nas chamadas das APIs.

- *POST /api/contacts/create*

Cria um contato no HubSpot utilizando a API, respeitando os limites de requisição.

- *POST /api/webhook/receive*

Recebe eventos do tipo contact.creation via webhook e salva os dados na base de dados.

3. Tecnologias Utilizadas

- Java 21
- Spring Boot 3.4.4
- Spring Web – Exposição dos endpoints REST
- Spring Data JPA – Persistência de dados com Hibernate
- Lombok – Redução de BoilerTemplate
- PostgreSQL – Banco de dados para armazenamento de informações
- SLF4J – Logging da aplicação
- Bucket4j – Biblioteca utilizada para controle de rate limiting
- OpenAPI – Documentação Swagger
- JUnit 5 – Framework de testes unitários
- MockMvc – Ferramenta de testes do Spring utilizada para simular chamadas HTTP e testar endpoints REST.

- H2 Database – Banco de dados em memória utilizado nos testes de integração.
- WireMock – Utilizado para mockar chamadas externas durante testes de integração, garantindo isolamento da API real do HubSpot.

4. Decisões Técnicas:

Para desenvolver a aplicação de integração com o HubSpot, escolhi utilizar o Java 21 por ser a versão LTS mais recente, com recursos modernos que ajudam a escrever um código mais limpo e eficiente. O Spring Boot 3.4.4 foi o framework escolhido por sua robustez, ampla documentação e facilidade na criação de aplicações REST

Na camada de persistência, foi utilizado o Spring Data JPA juntamente com o PostgreSQL como banco principal, já que é uma combinação estável, madura e que oferece todo o necessário em termos de consistência e flexibilidade.

Para os testes, optei por usar o H2, um banco em memória, que propõe isolamento e velocidade durante a execução dos testes de integração.

Para evitar a repetição de código desnecessária (como getters, setters e construtores), incluí o Lombok, que ajuda a deixar as classes mais limpas. Também utilizei o SLF4J para manter os logs padronizados e organizados.

Como a API do HubSpot impõe limites de requisições, adicionei a biblioteca Bucket4j para controlar o rate limiting de forma eficiente, protegendo minha aplicação contra bloqueios por excesso de chamadas. E para tornar a documentação da API mais acessível e clara, incluí o OpenAPI/Swagger, que gera uma interface interativa para explorar os endpoints.

Na parte de testes, usei JUnit 5 para garantir a cobertura de testes unitários e MockMvc para simular chamadas HTTP e testar os endpoints de forma prática. Também usei o WireMock para simular as respostas da API do HubSpot durante os testes de integração, o que me permitiu testar o comportamento da aplicação sem depender da API real.

5. Possíveis melhorias futuras:

Embora a aplicação esteja funcionando de forma estável, existem algumas melhorias que podem ser implementadas para deixar ainda mais robusta e preparada para ambientes de produção com maior escala.

Uma das primeiras melhorias seria a implementação de uma persistência segura para os tokens OAuth. Atualmente, esses tokens estão sendo armazenados de forma simplificada, o que é suficiente para um ambiente de desenvolvimento ou testes. No entanto, em produção, o ideal é garantir que esses dados estejam protegidos. Por exemplo, criptografando e armazenando em um local seguro para evitar qualquer risco de acesso indevido ou uso incorreto.

Outra melhoria seria refinar o controle de rate limiting. O uso do Bucket4j já resolve boa parte do problema, mas é possível torná-lo mais eficiente com estratégias como backoff exponencial ou até mesmo distribuir a carga entre múltiplas threads ou instâncias, caso o volume de requisições cresça com o tempo.

Também vale a pena considerar a implementação de cache para chamadas que retornam dados que não mudam com frequência. Isso ajudaria a reduzir o número de chamadas à API do HubSpot, economizando recursos e melhorando o desempenho da aplicação.

Por fim, seria interessante adicionar uma camada de monitoramento e observabilidade, usando ferramentas como Prometheus e Grafana para acompanhar métricas, consumo de recursos e comportamento da aplicação. Além disso, uma camada de auditoria para os eventos recebidos por webhook pode facilitar bastante o rastreamento de problemas e contribuir para a confiabilidade geral do sistema em produção.