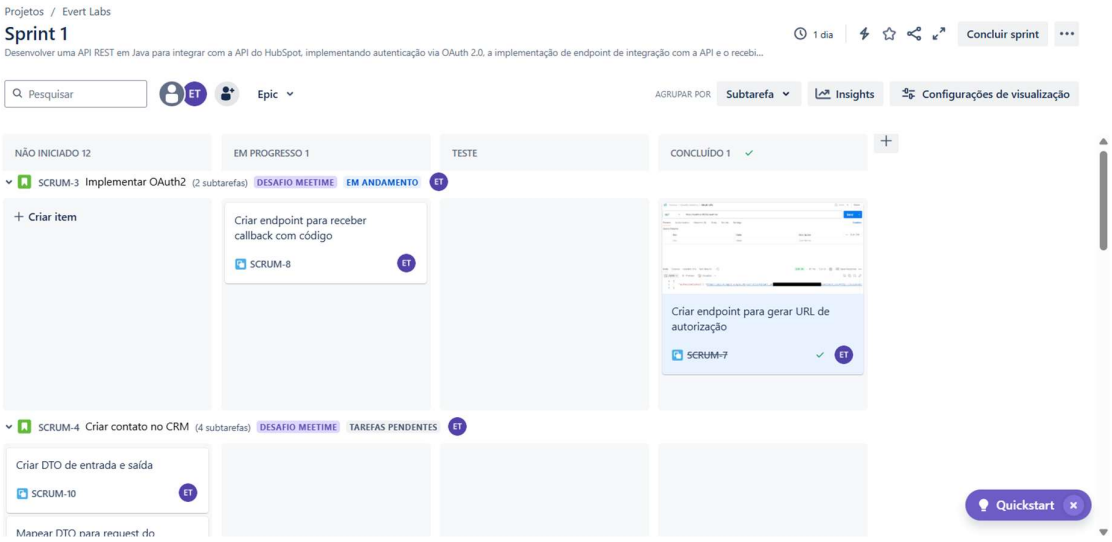
Documentação Técnica - Integração HubSpot

Visão Geral

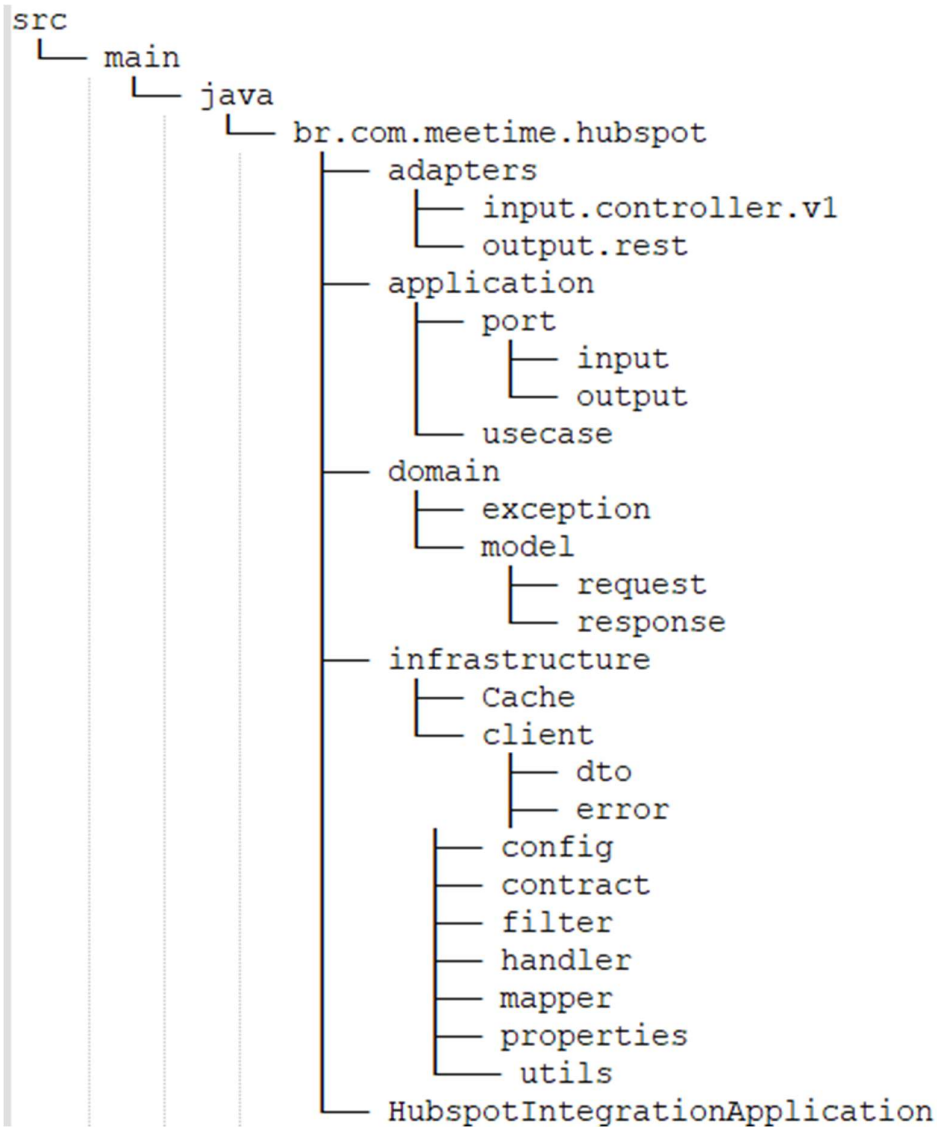
Este documento técnico descreve, de maneira completa e estruturada, as decisões de arquitetura, bibliotecas, padrões de projeto e boas práticas adotadas no desenvolvimento da aplicação srv-integracao-hubspot. A estrutura adotada e as práticas empregadas foram cuidadosamente escolhidas a partir de toda minha experiencia de trabalho.

Utilização do Jira para Planejamento e Execução

Durante o desenvolvimento deste desafio técnico, ultilizei o **Jira** como ferramenta de gestão e acompanhamento das atividades. A escolha pelo Jira foi feita organizar minha linha de pensamento, ter algo visual para me ajudar a não esquecer de nenhuma parte, e desmostrar que tenho conhecimento também em metodologias ágeis, como Scrum e Kanban.



Estrutura de Pastas



Arquitetura Hexagonal (Ports and Adapters)

A escolha pela arquitetura hexagonal (também conhecida como Ports and Adapters) veio da ideia de ser uma arquetetura que está sendo muito falada e uma arquitetura na qual estou estudando e começando a ultilizar nos projetos da empresa, porém na parte técnica eu pensei na necessidade de isolar a lógica de negócio das dependências externas (frameworks, SDKs, banco de dados, APIs externas). Essa abordagem promove alta testabilidade, flexibilidade e facilidade de manutenção.

Camadas:

- **Domain:** Modelos de domínio e regras de negócio.
- **Application:** Casos de uso e interfaces (ports).

- **Adapters:** Interfaces de entrada (REST controllers) e saída (chamadas a APIs externas).
- **Infrastructure:** Implementações concretas, como clientes Feign e caches.

Swagger/OpenAPI

O projeto utiliza a biblioteca springdoc-openapi-starter-webmvc-ui para documentação automática dos endpoints REST.

Motivo da Escolha:

- Compatibilidade nativa com Spring Boot.
- Geração de documentação interativa (Swagger UI).

Interface contract

As interfaces localizadas em contract centralizam as anotações do Swagger, promovendo **reusabilidade** e **limpeza dos controllers**, seguindo o princípio da separação de responsabilidades.

Bibliotecas Utilizadas

Spring Boot

Framework principal para desenvolvimento rápido e produtivo de aplicações Java modernas.

Spring Cloud

Utilizado para abstrações como Feign, circuit breaker, config client etc.

Feign

Para comunicação HTTP com APIs externas de maneira declarativa e com suporte a fallback.

Caffeine Cache

Foi implementado um cache baseado em Caffeine para salvar o body das requisições do webhook e permitir reprocessamento/validação.

Lombok

Reduz boilerplate em classes com geração automática de getters, setters, construtores etc.

Logback + CorrelationID

A inclusão do Correlation-ID nos logs e respostas permite o rastreamento fim-a-fim de requisições, crucial em ambientes distribuídos e microserviços.

Logs e Correlation ID

- Foi criado um **filtro global** que injeta o Correlation-ID automaticamente nos headers das requisições e respostas.
- Os logs foram configurados para exibir esse ID, permitindo rastreabilidade em ambientes complexos.

Cache do Body da Requisição

Para capturar o body das requisições de webhook e validar sua assinatura posteriormente, foi criada a classe CachedBodyRequest.

Por quê?

- O body do HttpServletRequest só pode ser lido uma vez.
- Solução permite reuso do conteúdo sem perder o stream original.

Segurança e Filtro de Assinatura

Foi implementado um **filtro de segurança** responsável por:

- Validar a assinatura HMAC vinda no header do webhook.
- Garantir que a requisição é legítima e não foi adulterada.

Motivo:

Segurança é essencial em integrações com sistemas de terceiros. A validação de assinatura protege contra ataques de replay e spoofing.

Manipulação de Exceções (Exception Handler)

A aplicação conta com @ControllerAdvice para:

- Capturar exceções genéricas e específicas.
- Padronizar respostas de erro.
- Garantir mensagens claras e estruturadas ao consumidor.

Exceções mapeadas incluem:

- IllegalArgumentException
- ConstraintViolationException
- Exceções customizadas do domínio

Handler de Controle de Taxa e Retentativa

Foi implementado um mecanismo com:

- **Controle de taxa (Rate Limiting)**
- **Tentativas automáticas com backoff exponencial (retry)**

Esse handler evita:

- Sobrecarga por chamadas em massa.
- Falhas temporárias em chamadas externas.

Construido a partir da documentação da Hubspot onde é sinalizado a quatidade de rate limit.

Properties Centralizados

Uma classe em properties carrega configurações externas via @ConfigurationProperties, garantindo:

- Centralização da configuração
- Validação automática com anotações do Spring Boot

Padrões de Projeto

Padrão	Onde foi aplicado
Builder	Construção de DTOs de resposta
Strategy	Validação de headers e assinatura
Adapter	Entrada/saída de dados (Hexagonal)
Proxy	Clientes Feign
Singleton	Configurações e cache

Boas Práticas Adotadas

- Separação clara de responsabilidades (SRP).
- Código limpo.
- Convenção de nomenclatura consistente.
- Tratamento robusto de erros e logs rastreáveis.

Melhorias Futuras

Testes Automatizados

- **Unitários**: Cobertura total dos casos de uso e regras de negócio (camada usecase).
- **Integração**: Testes que simulam chamadas externas ao HubSpot e Webhooks.

Integração com CI/CD (DevOps)

- Automatizar pipeline de build com **GitHub Actions**, **Jenkins**, ou **GitLab CI**.
- Executar testes e build em múltiplos ambientes.

Análise Estática de Código

- **SonarQube**: Verificação de cobertura de testes, duplicação de código, bugs e code smells.
- **Fortify (SAST)**: Análise de vulnerabilidades de segurança no código-fonte.
- **Mend**: Verificação de vulnerabilidades em bibliotecas de terceiros.

Observabilidade

- Integração com ferramentas como **Prometheus + Grafana** ou **Elastic Stack (ELK)** para análise de métricas e logs.

Documentação Viva

- Incluir exemplos reais e mocks para facilitar testes de QA.

Containerização e Orquestração

- Uso de Kubernetes (K8s) para escalar e monitorar.

Conclusão

A aplicação srv-integracao-hubspot foi projetada e construída com base em princípios de arquitetura, segurança, rastreabilidade. Cada decisão tomada teve como objetivo a manutenção da qualidade, a facilidade de evolução do projeto e a segurança.

Essa base sólida garante que o sistema possa escalar e se adaptar às necessidades futuras do negócio.