Documentação Técnica Unificada — APIs de Sentenças (Indexação, Busca e Download)

Versão: 26/09/2025

Sumário

---

1) Visão geral

O sistema é composto por duas APIs que atuam sobre documentos do tipo sentença:

● API de Indexação: orquestra a coleta de processos atualizados no Data Lake de Processos, filtra documentos de sentença, obtém o conteúdo quando público e indexa/atualiza o índice no OpenSearch. Pode ser acionada por endpoint HTTP (com Basic Auth) e/ou por agendamento diário.

● API de Busca e Download: expõe endpoints para consultar sentenças já indexadas, aplicando filtros e paginação por search_after e para baixar o binário do documento via proxy autenticado no Keycloak.

● Frontend: interface para uma pessoa leiga usar o serviço de maneira intuitiva e visual.

Objetivo

O Banco de Sentenças das Justiças Militares permite a extração e disponibilização automática de sentenças, sem gerar custos adicionais para os tribunais. Utilizando-se de soluções internas, na prática, elimina barreiras criadas pela diversidade de sistemas judiciais eletrônicos (EPROC, PJE etc.), unificando o acesso à produção judicial de 1ª instância de toda a Justiça Militar brasileira. Com isso, promove economia de tempo, ampliação da transparência e melhoria na prestação jurisdicional. O cidadão pode consultar decisões de forma simples e centralizada, enquanto juízes e equipes técnicas têm acesso facilitado à consulta no banco de dados das sentenças, o que também permite um intercâmbio de ideias, fomentando e melhorando a deliberação e as fundamentações das sentenças. A automação do processo, sem alimentação manual de sentenças no banco, também contribui para a celeridade, sustentabilidade e eficiência administrativa dos tribunais militares. De fato, a tecnologia desenvolvida no Banco de Sentenças das Justiças Militares contribuiu para a eficiência da Justiça Militar e, quiçá, futuramente, com a ampliação do projeto, de todo o Judiciário brasileiro.

Público-alvo

O público-alvo do Banco de Sentenças das Justiças Militares é composto por operadores do direito, tais como juízes, servidores, promotores, defensores, advogados, pesquisadores, bem como toda a população em geral, incluindo civis e militares, para fins de escrutínio, ampla transparência e publicidade do que é julgado nas Justiças Militares.

Escopo

O Banco de Sentenças das Justiças Militares congrega sentenças proferidas pelos juízes de 1ª instância das Justiças Militares de todo o país em um único portal, facilitando o acesso a todos os cidadãos, aumentando o escrutínio e conhecimento deste segmento de justiça. Desenvolvido em conjunto pelos Laboratórios de Inovação dos 4 tribunais militares do país (STM, TJMMG, TJMSP e TJMRS) em 2024 e lançado em 2025, a plataforma já se encontra disponível nos sites dos respectivos tribunais e agora na Plataforma Digital do Poder Judiciário do CNJ. Trata-se de ideia inovadora pois, pela 1ª vez, um mesmo segmento de justiça se une para disponibilizar suas sentenças, independentemente do tribunal ao qual o juiz está vinculado e do sistema de processo judicial eletrônico utilizado (EPROC, PJE, etc.), permitindo consulta avançada por termos, datas, classes processuais, dentre outros, democratizando a consulta do cidadão. Compartilhamos o vídeo de lançamento do projeto: https://youtu.be/cMxsOCGkL-I?feature=shared

2) Arquitetura geral

● Microservices Spring Boot com Eureka Client (opcional) para descoberta de serviços.

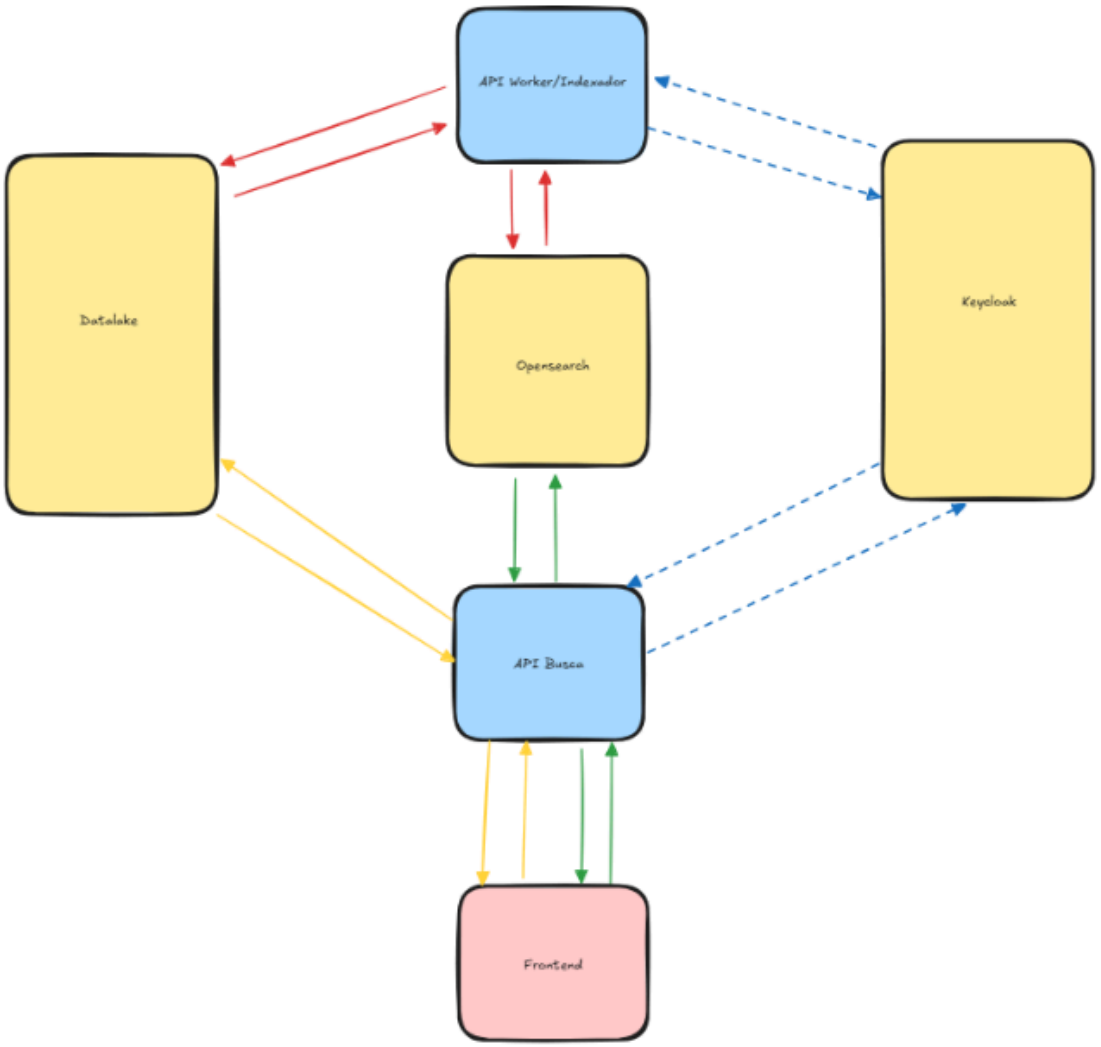
● OpenSearch como mecanismo de busca. O índice único opensearch.index agrega metadados e o texto público das sentenças.

● Data Lake de Processos como fonte de verdade para documentos e metadados processuais.

● Keycloak (Client Credentials) para obter access_token e autenticar chamadas ao Data Lake.

● Scheduler (fuso São Paulo) para execução periódica da indexação.

Diagrama (alto nível)



3) Stack, frameworks e bibliotecas

● Linguagem/Runtime: Java 21

● Framework: Spring Boot 3.5.x (parent)

● Web: Spring Web (REST)

● Segurança: Spring Security; OAuth2 Client para Keycloak

● Validações: Jakarta Validation + Hibernate Validator

● Cliente OpenSearch: High Level REST Client 2.13.0

● Descoberta: Spring Cloud Netflix Eureka Client (quando aplicável)

4) Código Fonte

Repositórios Git:

- **API de busca:** https://git.cnj.jus.br/pdpj/negocio/banco-sentencas
- **API indexadora:** https://git.cnj.jus.br/pdpj/negocio/banco-sentencas-worker
- **Frontend:** https://git.cnj.jus.br/pdpj/frontends/banco-sentencas

5) Segurança e autenticação

API de Indexação

● Proteção do endpoint é feita no controller via Basic Auth (par clientId:clientSecret).

● Para integrar com o Data Lake, obtém Bearer Token via Keycloak (client_credentials) com cache e renovação antecipada.

API de Busca e Download

● Expõe endpoints públicos do domínio (autenticação e rate-limit podem ser aplicados na borda). O endpoint de download usa Keycloak para chamar o Data Lake e repassa Content-Type e Content-Length ao cliente.

Frontend

● Não necessita de autenticação. O serviço será exposto na www.

6) Propriedades de configuração - Variáveis de ambiente

6.1) API de Indexação (conforme listado em application.yml)

● OPENSEARCH_HOST: endereço do opensearch

● OPENSEARCH_PORT: porta do opensearch

● OPENSEARCH_SCHEME: http ou https

● OPENSEARCH_USERNAME: usuário do opensearch

● OPENSEARCH_PASSWORD: senha do usuário

● OPENSEARCH_INDEX: índice no qual as sentenças estão armazenadas

● KEYCLOAK_AUTH_SERVER_URL: endereço do realm do keycloak autenticacao do serviço, via client credentials

● KEYCLOAK_CLIENT_ID: client do serviço do keycloak

● KEYCLOAK_CLIENT_SECRET: senha do client do serviço do keycloak

● DATALAKE_PROCESSOS_API_BASE_URL: endereço do datalake

● INDEXADOR_AUTH_CLIENT_ID: client para chamar a indexação por http get

● INDEXADOR_AUTH_CLIENT_SECRET: senha para o client que chama a indexação por http get

6.2) API de Busca (conforme listado em application.yml)

● EUREKA_SERVER_URL: endereço do eureka

● OPENSEARCH_HOST: endereço do opensearch

● OPENSEARCH_PORT: porta do opensearch

● OPENSEARCH_SCHEME: http ou https

● OPENSEARCH_USERNAME: usuário do opensearch

● OPENSEARCH_PASSWORD: senha do usuário

● OPENSEARCH_INDEX: índice no qual as sentenças estão armazenadas

● KEYCLOAK_AUTH_SERVER_URL: endereço do realm do keycloak autenticacao do serviço, via client credentials

● KEYCLOAK_CLIENT_ID: client do serviço do keycloak

● KEYCLOAK_CLIENT_SECRET: senha do client do serviço do keycloak

● DATALAKE_PROCESSOS_API_BASE_URL: endereço do datalake

6.3) Frontend

Conforme configuração de src\environments\environment.ts as requisições do front são disparadas para:

api: { url: 'http://localhost:8080/api' }

Como os serviços pegam a url base:

constructor(private http: HttpClient) { this.baseUrl = environment.api.url; }

Assim, faz-se necessário um mapeamento correto do endereço local para o endereço do gateway.

7) Modelo de dados — Índice das Sentenças

JSON

```
{
  "settings": {
    "analysis": {
      "analyzer": {
        "pt_br_analyzer": { "type": "brazilian" }
      }
    }
  },
  "mappings": {
    "dynamic": false,
    "properties": {
      "des_tribunal": { "type": "keyword" },
      "id_sentenca": { "type": "keyword" },
      "num_processo": { "type": "keyword" },
      "id_classe": { "type": "integer" },
      "des_classe": { "type": "keyword" },
      "id_assunto_principal": { "type": "integer" },
      "des_assunto_principal": { "type": "keyword" },
      "ids_assuntos": { "type": "integer" },
      "des_assuntos": { "type": "keyword" },
      "id_orgao": { "type": "integer" },
      "des_orgao": { "type": "keyword" },
      "data_juntada": { "type": "date", "format": "yyyy-MM-dd" },
      "@timestamp": { "type": "date", "format": "epoch_second" },
      "texto": { "type": "text", "analyzer": "pt_br_analyzer" }
    }
  }
}
```

8) Endpoints das aplicações (APIs)

8.1) API de Indexação

GET /api/sentencas/indexar — Dispara a indexação de sentenças de processos atualizados no Data Lake e sincroniza o índice no OpenSearch.

- Parâmetros (query):
  - tribunal (obrigatório)
  - dataHoraAtualizacaoInicio (ISO-8601, opc.)
  - dataHoraAtualizacaoFim (opc.)
  - searchAfter (cursor opc.)
- **Autenticação:** `Authorization: Bearer <token>`
- **Respostas:** 200 OK (boolean), 401 Unauthorized, 502 Bad Gateway (falha em dependências).
- **Agendamento:** Execução diária com fuso America/Sao_Paulo (via `@EnableScheduling` + `@Scheduled`).

GET /api/sentencas/indexarOrgao — Dispara a indexação de sentenças de processos atualizados no Data Lake e sincroniza o índice no OpenSearch.

- Parâmetros (query):
  - órgão(obrigatório)
  - dataHoraAtualizacaoInicio (ISO-8601, opc.)
  - dataHoraAtualizacaoFim (opc.)
  - searchAfter (cursor opc.)
- **Autenticação:** `Authorization: Bearer <token>`
- **Respostas:** 200 OK (boolean), 401 Unauthorized, 502 Bad Gateway (falha em dependências).
- **Agendamento:** Execução diária com fuso America/Sao_Paulo (via `@EnableScheduling` + `@Scheduled`).

8.2) API de Busca e Download

POST /api/sentencas/buscar — Busca sentenças no OpenSearch com filtros e ordenação por @timestamp.

- Request (JSON) — SentencaRequest
  - pageOffset (int) — 1 a 3 (validação personalizada)

- pageSize (int) — 10, 20 ou 30
- searchAfterValue (long) — cursor opcional
- order (string) — direção de ordenação em @timestamp (ASC/DESC)
- searchFilters (array; somente o primeiro item é consumido):
- texto (string; queryStringQuery)
- ids_assuntos (array<int>; terms em ids_assuntos)
- ids_classes (array<int>; terms em id_classe)
- ids_orgaos (array<int>; terms em id_orgao)
- data_inicio, data_fim (strings; range em data_juntada)

- Response — SentencaResponse
  - sentencas (SentencaDTO), total (long), status (int)
- SentencaDTO (campos relevantes)
  num_processo, id_sentenca, des_assuntos, des_assunto_principal, des_orgao, data_juntada, des_classe, texto, des_tribunal, ids_assuntos, id_assunto_principal, @timestamp, id_orgao, id_classe.

GET /api/sentencas/download — Proxy do binário do documento no Data Lake.

- **Parâmetros (query):** num_processo, id_sentenca
- **Respostas:** 200 OK (binário com Content-Type/Content-Length repassados), 502 Bad Gateway (falha de backend)

9) Fluxos e casos de uso

UC-Indexação (alto nível)

● Listar processos atualizados no Data Lake (paginação com searchAfter; resposta traz content, total, maxElementsSize e o cursor searchAfter).

● Para cada processo, buscar documentos e filtrar tipos aceitos (códigos 550/795/796). Se público: obter texto e indexar; Se sigiloso: remover do índice.

● Enriquecimento: consultar movimentos para complementar classe e órgão julgador (a partir do dia da juntada).

● Montar SentencaIndex e indexar no OpenSearch.

UC-Busca

● Cliente chama POST /api/sentencas/buscar com filtros.

● Serviço monta BoolQuery (texto, assuntos, classes, órgãos, intervalo de datas).

● Ordena por @timestamp (ASC/DESC) e pagina via search_after.

● Mapeia hits para SentencaDTO e retorna SentencaResponse.

UC-Download

● Cliente chama GET /api/sentencas/download?num_processo=...&id_sentenca=....

● Serviço obtém token no Keycloak (client_credentials) e chama o endpoint binário do Data Lake.

● API repassa Content-Type/Content-Length e retorna o binário ao cliente.

10) Paginação e limites do backend

Busca: ordenação sempre em @timestamp; pageSize em 10/20/30 e pageOffset ≤ 3 (para evitar deep pagination). Caso searchAfterValue seja informado, é usado diretamente; caso contrário, a API deriva o cursor internamente ao varrer páginas até pageOffset e retorna somente a página selecionada.

Indexação: integração com o Data Lake e com o OpenSearch deve usar searchAfter para evitar deep pagination; manter tamanhos de página moderados.

11) Requisitos não funcionais

● Timeouts em clientes HTTP (Keycloak/Data Lake) e conexões OpenSearch ajustados para resiliência.

● Retry/Rate Limiter (ex.: Resilience4j) para tratar erros transitórios nas integrações.

● Escalonamento: considerar réplicas ≥ 2 nas APIs para tolerância a falhas e throughput.

Perfis de carga (sugestão inicial):

● POST /api/sentencas/buscar: ~10 RPM

● GET /api/sentencas/download: 5 RPM

12) Execução e empacotamento

● Habilitações: @EnableConfigurationProperties e @EnableScheduling (na API de Indexação).

● Maven: mvn spring-boot:run / mvn package.

13) Exemplos

13.1) Disparar indexação (GET) - *ENDPOINT CRIADO PARA FUTURA ESTRATÉGIA DE FALLBACK*

HTTP

None

```
GET
/api/sentencas/indexar?tribunal=TJMMG&dataHoraAtualizacaoInicio=2
020-01-01T00%3A00%3A00.000&dataHoraAtualizacaoFim=2025-12-01T00%3
A00%3A00.000 HTTP/1.1
Host: localhost:8080
Authorization:                                          Basic
bWV1LWNsaWVudC1pZC1kZXY6bWV1LWNsaWVudC1zZWNyZXQtZGV2
```

13.2) Buscar sentenças (POST)

HTTP

None

```
POST /api/sentencas/buscar HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 326

{
  "pageOffset": 1,
  "pageSize": 10,
  "searchAfterValue": null,
  "order": "asc",
  "searchFilters": [
    {
      "ids_assuntos": [],
      "ids_classes": [],
      "ids_orgaos": [],
      "texto": null,
      "data_inicio": "2025-01-31",
      "data_fim": "2025-12-31"
    }
  ]
}
```

13.3) Download de documento (GET)

HTTP

```
None


GET
/api/sentencas/download?num_processo=01015894820201000000&id_sent
enca=1a9cdec4-c45c-5e9f-8155-678fb5e1def5 HTTP/1.1
```

Anexo — Referência de estilo

Este documento consolida a estrutura e conteúdo de ambas as APIs, harmonizando stack, casos de uso, paths, contratos, paginação, arquitetura e propriedades. Ajustes finos (nomes de propriedades variando por serviço) foram mantidos como observações para facilitar a adoção em ambientes existentes.