Sistema GAD

Documentação Técnica do Backend para Gerenciamento de Atividades Diversificadas no IFMS Naviraí

1. Introdução

Este documento detalha a arquitetura e implementação do componente backend para o Sistema de Gerenciamento de Atividades Diversificadas (GAD), desenvolvido como parte de um Trabalho de Conclusão de Curso (TCC) para o Instituto Federal de Mato Grosso do Sul (IFMS), campus Naviraí. O objetivo principal do Sistema GAD é modernizar e automatizar o processo de registro, validação e acompanhamento das horas de atividades complementares (UC - Unidade Curricular, PE - Pesquisa e Extensão, PP - Participação em Projetos, PD - Participação Docente, PA - Atividades de Apoio, PV - Participação Voluntária, TCC - Trabalho de Conclusão de Curso) realizadas pelos estudantes, substituindo procedimentos manuais propensos a erros e demorados.

O sistema segue uma arquitetura cliente-servidor, onde o frontend, desenvolvido em Angular, é responsável pela interface do usuário e interação, enquanto o backend, foco deste relatório, é construído utilizando o Spring Framework (especificamente Spring Boot) e a linguagem Java. As responsabilidades centrais do backend incluem o recebimento de certificados (em formato PDF ou imagem), a extração de informações relevantes utilizando técnicas de Reconhecimento Óptico de Caracteres (OCR) e Inteligência Artificial (IA), a classificação automática das atividades segundo as normativas do IFMS, a validação das horas (considerando limites por categoria e regras de distribuição de excedentes) e, potencialmente, a validação de autenticidade através de QR codes presentes nos certificados. Adicionalmente, o backend gerencia a persistência dos dados e expõe uma API RESTful para comunicação com o frontend.

Este relatório técnico visa fornecer uma descrição detalhada da arquitetura do backend,

das tecnologias empregadas, dos módulos principais, da integração com a base de dados vetorial e das considerações de implantação, servindo como documentação fundamental para o desenvolvimento, manutenção e futuras evoluções do sistema.

2. Arquitetura do Sistema

2.1. Arquitetura Geral

O Sistema GAD adota uma arquitetura cliente-servidor padrão. O cliente é uma aplicação web single-page (SPA) desenvolvida em Angular, responsável pela apresentação da interface do usuário e pela comunicação com o servidor via requisições HTTP. O servidor consiste em uma aplicação backend monolítica desenvolvida com Spring Boot.

A escolha por uma arquitetura monolítica para o backend justifica-se pela natureza do projeto (TCC) e pela equipe reduzida, simplificando o desenvolvimento, o teste e a implantação iniciais. No entanto, a arquitetura é projetada com modularidade em mente, permitindo uma eventual refatoração para microsserviços caso a complexidade e a escala do sistema aumentem significativamente no futuro. A comunicação entre frontend e backend ocorre exclusivamente através de uma API RESTful bem definida.

2.2. Pilha Tecnológica do Backend (Technology Stack)

A seleção das tecnologias para o backend foi guiada pela robustez, maturidade, ecossistema de ferramentas e adequação aos requisitos do projeto, especialmente no que tange ao processamento de documentos e integração com IA:

- **Java (LTS):** Linguagem de programação principal, escolhida por sua maturidade, performance, vasto ecossistema e forte tipagem, adequada para aplicações empresariais robustas. Utilizar uma versão Long-Term Support (LTS) garante estabilidade e suporte estendido.
- **Spring Boot:** Framework principal para o desenvolvimento rápido de aplicações Java. Facilita a configuração, gerenciamento de dependências e criação de

aplicações autônomas e prontas para produção, seguindo o princípio de convenção sobre configuração.

- **Spring Data JPA:** Facilita a interação com o banco de dados relacional, abstraindo grande parte do código boilerplate de persistência através do mapeamento objeto-relacional (ORM) e da implementação de repositórios.[1]
- **PostgreSQL com pgVector:** Banco de dados relacional open-source robusto, estendido com a extensão pgvector para suportar armazenamento e consulta eficiente de vetores de alta dimensionalidade, essencial para funcionalidades de busca por similaridade semântica.[2]
- **Spring AI:** Projeto Spring que fornece uma abstração unificada para interagir com diversos modelos de IA (LLMs, modelos de embedding, etc.), simplificando a integração com provedores como OpenAI, Hugging Face e ferramentas locais como Ollama.[3]
- **Tess4J:** Biblioteca Java que atua como um wrapper para a engine Tesseract OCR. Permite a extração de texto a partir de imagens e documentos PDF digitalizados.[7]
- **LangChain4j (Opcional/Complementar):** Framework Java inspirado no LangChain Python, oferecendo ferramentas para construir cadeias de processamento de linguagem mais complexas, incluindo pipelines de Retrieval-Augmented Generation (RAG).[10] Pode ser considerado se as abstrações do Spring AI se mostrarem limitantes para fluxos específicos.
- **ZXing ("Zebra Crossing"):** Biblioteca Java para geração e leitura de códigos de barras multi-formato, incluindo QR codes, utilizada para a funcionalidade de validação de certificados via QR code.[15]
- **Ollama:** Ferramenta que permite executar modelos de linguagem grandes (LLMs) localmente, facilitando o desenvolvimento e teste sem dependência de APIs pagas e oferecendo maior controle sobre os modelos utilizados.[3]
- **Modelos Hugging Face:** Através do Ollama ou diretamente pelo Spring AI, é possível acessar uma vasta gama de modelos pré-treinados disponíveis no Hugging Face Hub, incluindo modelos específicos para o português como o

BERTimbau [18] e suas variantes fine-tuned [19], ou modelos de classificação [21], que podem ser importados se estiverem no formato GGUF compatível com Ollama.[5]

2.3. Esquema do Banco de Dados

O esquema do banco de dados PostgreSQL é projetado para armazenar as informações essenciais do sistema. As entidades principais incluem:

- **User (ou Student):** Representa os estudantes do IFMS que utilizarão o sistema. Contém informações básicas como ID, nome, matrícula, email, senha (hashed), e total de horas validadas por categoria.
- **Certificate:** Armazena informações sobre os certificados submetidos pelos estudantes.
  - id: Chave primária (UUID).
  - student_id: Chave estrangeira referenciando User.
  - file_name: Nome original do arquivo.
  - file_path (ou file_content): Caminho no sistema de arquivos onde o certificado original está armazenado (ou o conteúdo binário, se armazenado como BLOB).
  - extracted_text: Texto extraído pelo OCR (pode ser extenso).
  - processing_status: Estado do processamento (e.g., PENDING, OCR_DONE, AI_DONE, VALIDATED, REJECTED, ERROR).
  - upload_timestamp: Data e hora do upload.
  - error_message: Mensagem de erro, caso ocorra falha no processamento.
- **ActivityCategory:** Tabela de domínio para armazenar as categorias de atividades permitidas (UC, PE, PP, PD, PA, PV, TCC) e seus respectivos limites de horas, conforme regulamento do IFMS.
  - id: Chave primária.
  - code: Código da categoria (e.g., "PE").
  - name: Nome descritivo (e.g., "Pesquisa e Extensão").
  - max_hours: Limite máximo de horas para esta categoria.

- **Activity:** Representa uma atividade validada a partir de um certificado.
  - id: Chave primária (UUID).
  - certificate_id: Chave estrangeira referenciando Certificate.
  - category_id: Chave estrangeira referenciando ActivityCategory.
  - description: Descrição da atividade (extraída ou resumida pela IA).
  - validated_hours: Quantidade de horas validadas para esta atividade específica.
  - activity_date: Data de realização da atividade (extraída pela IA).
  - validation_timestamp: Data e hora da validação.
  - embedding: Vetor de embedding gerado a partir da descrição ou texto relevante, armazenado usando o tipo vector da extensão pgvector.[2] Este campo é crucial para a funcionalidade de busca por similaridade.

A utilização de UUIDs como chaves primárias é recomendada para evitar colisões em ambientes distribuídos e facilitar a integração futura. A coluna embedding na tabela Activity (ou em uma tabela associada, como vector_store gerenciada pelo Spring AI [2]) é fundamental para habilitar buscas semânticas.

3. Módulos Principais do Backend

O backend é organizado em módulos funcionais coesos, cada um responsável por uma parte específica do fluxo de processamento.

3.1. Upload e Processamento de Certificados

Este módulo gerencia o recebimento e o início do processamento dos certificados enviados pelos usuários.

- **Fluxo:**
  1. O usuário, através do frontend Angular, envia um arquivo (PDF ou imagem) via requisição POST para um endpoint específico da API (e.g., /api/certificates).
  2. O Controller Spring recebe o arquivo como um MultipartFile.
  3. O CertificateService é invocado para lidar com o arquivo.
  4. O serviço salva o arquivo original em um local de armazenamento persistente.

Para um TCC, armazenar no sistema de arquivos do servidor é uma abordagem simples e eficaz, embora soluções como armazenamento em nuvem (AWS S3, Azure Blob Storage) ou no próprio banco de dados (como BLOB) sejam alternativas mais escaláveis para produção.

5. Um registro inicial é criado na tabela Certificate com o status PENDING.
6. O processamento subsequente (OCR, IA, Validação) é disparado de forma assíncrona. Isso é crucial porque essas etapas podem ser demoradas e não devem bloquear a resposta ao usuário. A assincronicidade pode ser implementada usando a anotação @Async do Spring em métodos de serviço ou, para maior robustez e escalabilidade, utilizando um message broker como RabbitMQ ou Kafka (embora possa ser um exagero para o escopo inicial do TCC).

- **Armazenamento de Arquivos:** A escolha do local de armazenamento (filesystem, DB, cloud) deve considerar simplicidade, custo e escalabilidade. Para o TCC, o filesystem é recomendado. É importante garantir que o caminho do arquivo seja armazenado corretamente na entidade Certificate.
- **Processamento Assíncrono:** Essencial para não degradar a experiência do usuário. O status na entidade Certificate deve ser atualizado conforme o processamento avança pelas diferentes etapas (e.g., OCR_DONE, AI_PROCESSING, VALIDATED, ERROR).

3.2. Módulo OCR (Tesseract/Tess4J)

Responsável por extrair o texto bruto contido nos arquivos de certificado.

- **Função:** Converter imagens de texto (sejam elas arquivos de imagem ou páginas de PDF renderizadas) em texto editável e pesquisável.[7]
- **Integração:**
  - Adicionar a dependência Maven net.sourceforge.tess4j ao pom.xml.[7] Versões específicas devem ser verificadas quanto à compatibilidade.[24]
  - Instalar o Tesseract OCR na máquina de desenvolvimento e no ambiente de

implantação.[7]

- Configurar o caminho para o diretório tessdata, que contém os arquivos de dados de treinamento de idioma (.traineddata). Isso pode ser feito via variável de ambiente ou programaticamente usando tesseract.setDatapath().[8] É crucial incluir o arquivo para o português (por.traineddata).
- Pode ser necessário configurar o java.library.path ou usar utilitários do Tess4J para extrair bibliotecas nativas incluídas no JAR.[9]

- **Implementação:**
  - Criar um OcrService injetável.
  - Para arquivos de imagem (MultipartFile), o serviço pode salvar temporariamente o arquivo [7] e então chamar tesseract.doOCR(file).[7]
  - Para arquivos PDF, utilizar uma biblioteca como Apache PDFBox.[7] O PDF é carregado, cada página é renderizada como uma BufferedImage (com DPI adequado, e.g., 300, para melhor qualidade [7]), e o OCR é aplicado a cada imagem de página. Os textos resultantes de todas as páginas são concatenados.[7]
  - O texto extraído é então armazenado na entidade Certificate correspondente.
- **Desafios e Pré-processamento:** A precisão do OCR é altamente dependente da qualidade da imagem de entrada. Documentos escaneados, com ruído, baixa resolução, fontes incomuns ou layout complexo podem resultar em extrações imprecisas.[25] Tesseract funciona melhor com texto bem segmentado do fundo.[25] Para mitigar isso, podem ser aplicadas técnicas de pré-processamento de imagem antes do OCR, como:
  - Conversão para escala de cinza.[25]
  - Ajuste de contraste e brilho.[25]
  - Binarização (conversão para preto e branco puro).
  - Remoção de ruído (denoising).
  - Aumento da resolução/DPI (upscaling), como configurar tesseract.setTessVariable("user_defined_dpi", "300").[7]

- Correção de inclinação (deskewing). A implementação dessas técnicas pode envolver bibliotecas adicionais de processamento de imagem Java (como java.awt.image ou OpenCV via wrappers).

3.3. Módulo de IA (Extração de Informação e Classificação)

Este módulo utiliza modelos de linguagem para analisar o texto extraído pelo OCR, identificar informações chave e classificar a atividade.

- **Propósito:** Automatizar a interpretação do conteúdo do certificado, extraindo dados como nome do participante, título/descrição da atividade, data de realização, carga horária, e classificando a atividade em uma das categorias definidas pelo IFMS (UC, PE, PP, etc.).
- **Tecnologia:** A abordagem principal é usar o Spring AI como camada de abstração, conectado ao Ollama para execução local de LLMs.[3]
- **Seleção de Modelo:**
  - Via Ollama, modelos como Llama 3, Mistral ou Phi-3 podem ser testados.[3] Ollama permite puxar modelos do seu próprio repositório ou diretamente do Hugging Face Hub, desde que estejam no formato GGUF.[5]
  - A busca por modelos específicos para o português no Hugging Face [18] é recomendada. Se um modelo GGUF adequado existir, pode ser usado com Ollama.
  - Para a tarefa específica de classificação, se os LLMs gerais não forem precisos o suficiente, poderia se considerar o fine-tuning de um modelo como o BERTimbau [18] em um dataset específico de certificados/atividades do IFMS.[20] No entanto, integrar um modelo fine-tuned que não seja via Ollama/Spring AI pode exigir mais esforço (e.g., usando bibliotecas Python via um serviço separado ou wrappers Java para PyTorch/TensorFlow).
- **Configuração Spring AI:**
  - Adicionar a dependência spring-ai-starter-model-ollama (ou o nome atualizado conforme a versão do Spring AI).[3]

    - Configurar as propriedades no application.properties ou application.yml:
        - spring.ai.ollama.base-url: Endereço onde o servidor Ollama está rodando (e.g., http://localhost:11434).[5]
        - spring.ai.ollama.chat.options.model: Nome do modelo a ser usado (e.g., llama3, mistral, ou um modelo customizado puxado via Ollama como hf.co/neuralmind/bert-base-portuguese-cased se convertido para GGUF).[4]
        - Outras opções como temperature, top-k, top-p podem ser ajustadas para controlar a criatividade e a previsibilidade da resposta.[5]
- **Engenharia de Prompts:** A qualidade da extração e classificação depende fortemente dos prompts enviados ao LLM.
    - Utilizar a classe PromptTemplate do Spring AI para criar prompts dinâmicos que incluam o texto extraído do OCR e instruções claras.[3]
    - Exemplo de prompt para extração: "Analise o texto do certificado a seguir e extraia as seguintes informações no formato JSON: nome_participante (string), titulo_atividade (string), data_inicio (formato YYYY-MM-DD), data_fim (formato YYYY-MM-DD), carga_horaria (float). Texto: {texto_ocr}".
    - Exemplo de prompt para classification: "Com base no título e descrição da atividade extraídos ('{titulo_atividade}'), classifique-a em uma das seguintes categorias do IFMS: UC, PE, PP, PD, PA, PV, TCC. Forneça apenas o código da categoria mais apropriada.".
    - A capacidade do modelo de retornar JSON estruturado deve ser verificada. Caso contrário, será necessário parsear a resposta em texto livre.
- **Alternativa LangChain4j:** Se o processo exigir etapas mais complexas como divisão inteligente do documento (chunking), embedding de partes específicas, ou estratégias de RAG mais sofisticadas (além da busca por similaridade básica), LangChain4j oferece mais blocos de construção modulares.[10] Poderia ser usado em conjunto ou como alternativa a partes do Spring AI.

3.4. Módulo de Validação

Implementa as regras de negócio específicas do IFMS para validar as horas das atividades.

- **Lógica de Negócio:** Este módulo contém a lógica para aplicar as regras definidas no regulamento de atividades complementares do IFMS. Isso inclui:
  - Verificar o limite máximo de horas permitido para a categoria classificada pela IA.
  - Calcular o total de horas do estudante em cada categoria.
  - Aplicar regras de aproveitamento de horas excedentes (se houver, e como podem ser distribuídas entre outras categorias ou para o total geral).
  - Verificar a validade da data da atividade em relação ao período do curso do estudante.
- **Entrada:** Os dados extraídos e classificados pelo Módulo de IA (categoria, carga horária declarada no certificado, data).
- **Saída:** Uma nova entrada na tabela Activity contendo as horas efetivamente validadas (validated_hours), que podem ser iguais ou menores que a carga horária declarada, dependendo dos limites. Atualização do total de horas do estudante. Atualização do status do Certificate para VALIDATED ou REJECTED (com motivo).
- **Implementação:**
  - Criar um ValidationService dedicado.
  - Implementar métodos claros e testáveis para cada regra de validação.
  - Considerar armazenar os parâmetros das regras (limites de horas por categoria) em um local configurável (e.g., tabela ActivityCategory no banco de dados ou arquivos de propriedades) para facilitar atualizações sem modificar o código.

3.5. Módulo de Validação de QR Code (ZXing)

Adiciona uma camada opcional de verificação de autenticidade se os certificados contiverem QR codes válidos.

- **Propósito:** Ler e interpretar dados codificados em QR codes presentes nos

certificados para fins de validação (e.g., verificar um link de autenticidade, confirmar dados com uma fonte externa).

- **Tecnologia:** Biblioteca ZXing.[15]
- **Integração:**
  - Adicionar as dependências Maven com.google.zxing:core e com.google.zxing:javase.[16]
- **Implementação:**
  - Criar um QrCodeService.
  - O serviço recebe a imagem do certificado (ou uma região específica onde o QR code se encontra, se detectável). Pode ser um InputStream, File ou BufferedImage.[15]
  - Utilizar ImageIO.read() para carregar a imagem.[15]
  - Criar um LuminanceSource (e.g., BufferedImageLuminanceSource) a partir da imagem.[15]
  - Criar um BinaryBitmap usando um Binarizer (e.g., HybridBinarizer).[15]
  - Utilizar MultiFormatReader().decode(binaryBitmap) para tentar decodificar o QR code.[15] Hints podem ser passados para otimizar a decodificação.[30]
  - Se a decodificação for bem-sucedida (Result obtido), extrair o texto (result.getText()).[15]
  - Processar o texto decodificado: verificar se é uma URL válida, consultar um serviço externo, comparar com dados extraídos pelo OCR/IA.
  - Tratar a exceção NotFoundException que é lançada se nenhum QR code for encontrado ou decodificado na imagem.[15]

4. Integração com Banco de Dados Vetorial (pgVector)

A integração com pgvector permite funcionalidades avançadas de busca baseadas no significado semântico do conteúdo das atividades.

4.1. Propósito

Habilitar a busca por similaridade semântica. Em vez de apenas buscar por palavras-chave exatas, o sistema pode encontrar atividades ou certificados cujo conteúdo textual (descrição, título, texto extraído) seja semanticamente similar a uma consulta fornecida pelo usuário. Isso pode ser útil para:

- Estudantes encontrarem exemplos de atividades já validadas em áreas de interesse.
- Administradores identificarem possíveis duplicatas ou inconsistências.
- Sugerir categorias para novas submissões com base em atividades similares já existentes.

4.2. Configuração

- **Extensões PostgreSQL:** É fundamental que as extensões vector, hstore (frequentemente usada para metadados com pgvector) e uuid-ossp (se UUIDs forem usados como IDs) estejam habilitadas na base de dados PostgreSQL.[2] O PgVectorStore do Spring AI pode tentar habilitá-las automaticamente na inicialização se configurado (spring.ai.vectorstore.pgvector.initialize-schema=true), mas é mais seguro garantir que elas já existam.[2]
- **Configuração Spring AI:**
    - Adicionar a dependência do starter pgvector do Spring AI (e.g., spring-ai-starter-vector-store-pgvector).
    - Configurar as propriedades no application.properties/yml [2]:
        - spring.datasource.url, username, password: Credenciais do banco de dados PostgreSQL.
        - spring.ai.vectorstore.pgvector.index-type: Tipo de índice a ser usado para busca aproximada de vizinhos mais próximos (ANN). Opções comuns são HNSW (geralmente mais performático para busca, mas mais lento para construir) e IVFFlat (mais rápido para construir, usa menos memória, mas potencialmente mais lento na busca).[2] NONE realiza busca exata (força bruta), que é lenta para grandes volumes. HNSW é um bom padrão inicial.

    - spring.ai.vectorstore.pgvector.distance-type: Métrica de distância para comparar vetores. COSINE_DISTANCE (Cosseno) é comum e funciona bem para embeddings normalizados. EUCLIDEAN_DISTANCE (L2) ou NEGATIVE_INNER_PRODUCT (Produto Interno Negativo) podem ser mais eficientes se os vetores estiverem normalizados (comprimento 1).[2] A escolha deve ser compatível com o modelo de embedding usado.
    - spring.ai.vectorstore.pgvector.dimensions: A dimensionalidade dos vetores gerados pelo EmbeddingModel escolhido. Este valor *deve* corresponder à saída do modelo (e.g., 1536 para text-embedding-ada-002 da OpenAI, 768 para muitos modelos BERT base). O Spring AI pode tentar inferir isso do EmbeddingModel, mas é mais seguro definir explicitamente.[2] Mudar as dimensões exige recriar a tabela ou coluna do vetor.
    - spring.ai.vectorstore.pgvector.table-name: Nome da tabela usada pelo PgVectorStore (padrão: vector_store).[2]
    - spring.ai.vectorstore.pgvector.schema-name: Nome do schema (padrão: public).[2]
- **Modelo de Embedding:** É necessário um EmbeddingModel configurado no Spring AI para converter o texto das atividades em vetores numéricos.[2] Pode ser um modelo da OpenAI, Hugging Face (via Ollama ou API), ou outro suportado pelo Spring AI.[6] A escolha do modelo impacta a qualidade da busca por similaridade e a dimensionalidade dos vetores.

4.3. Indexação

O processo de converter texto em vetores e armazená-los no banco de dados.

- **Processo:**
    1. Após uma atividade ser validada e seus dados relevantes (descrição, título) estarem disponíveis, esse texto é selecionado para indexação.
    2. O texto é passado para o EmbeddingModel configurado, que retorna um vetor de ponto flutuante (embedding).[11]

3. Um objeto Document do Spring AI é criado, contendo o texto original, metadados relevantes (e.g., ID da atividade, ID do estudante, categoria) e o embedding gerado.[11]
4. A lista de Documents é passada para o método vectorStore.add(documents).[2] O PgVectorStore se encarrega de inserir os dados na tabela configurada (e.g., vector_store), incluindo o vetor na coluna apropriada do tipo vector.

- **Quando Indexar:** A indexação deve ocorrer tipicamente após a validação bem-sucedida de uma atividade, garantindo que apenas conteúdo relevante e verificado seja usado para buscas futuras.

4.4. Busca por Similaridade

Recuperar atividades/documentos do banco de dados com base na similaridade semântica com uma consulta.

- **Implementação:**
  - Utilizar o método vectorStore.similaritySearch(SearchRequest).[2]
  - Construir um SearchRequest usando SearchRequest.builder():
    - .query("texto da busca"): Fornecer o texto de consulta do usuário. O PgVectorStore usará internamente o EmbeddingModel para converter essa query em um vetor.
    - .topK(5): Especificar o número máximo de resultados similares a serem retornados.[2]
    - .similarityThreshold(0.75): Definir um limiar mínimo de similaridade (score entre 0 e 1, dependendo da distance-type - para cosseno, mais perto de 1 é mais similar). Apenas resultados com score acima do limiar são retornados.[2]
    - .filterExpression("metadata_field == 'value'"): Opcionalmente, aplicar filtros nos metadados armazenados junto com os vetores para refinar a busca (e.g., buscar apenas atividades de uma categoria específica ou de um determinado período).[2] Spring AI suporta uma linguagem de expressão de

filtro ou um DSL programático.[2]

- **Resultado:** O método retorna uma List<Document>, onde cada Document contém o texto original, seus metadados e, potencialmente, o score de similaridade. Esses dados podem ser usados para apresentar os resultados ao usuário no frontend.

4.5. Uso Avançado (Consultas Nativas)

Embora o VectorStore do Spring AI ofereça uma abstração conveniente, pode haver cenários onde o acesso direto aos recursos do pgvector seja necessário.

- **Quando Considerar:**
  - Se for preciso usar operadores específicos do pgvector que não são expostos pela abstração VectorStore (e.g., <-> para distância L2, <#> para produto interno negativo, <=> para distância cosseno) diretamente na cláusula ORDER BY ou WHERE.[2]
  - Para controle fino sobre a criação e utilização de índices pgvector (além das opções index-type básicas).
  - Para combinar buscas vetoriais com lógicas SQL complexas que não se encaixam bem nos filtros de metadados do Spring AI.
- **Implementação com Spring Data JPA:**
  - Utilizar a anotação @Query em métodos de repositório, definindo nativeQuery = true.[1]
  - Escrever a consulta SQL específica do PostgreSQL que utiliza a coluna do tipo vector e os operadores do pgvector.
  - Exemplo:

```
Java
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
//... outras importações

public interface ActivityRepository extends JpaRepository<Activity, UUID> {
```

```
    @Query(nativeQuery = true, value = "SELECT * FROM activity ORDER BY embedding <=>
CAST(:vector AS vector) LIMIT :k")
    List<Activity> findSimilarActivities(@Param("vector") float vector, @Param("k") int k);

    // Nota: A forma de passar o vetor (:vector) e o CAST podem variar
    // dependendo da versão do driver JDBC e do pgvector.
    // Pode ser necessário usar String ou outros tipos e fazer o CAST no SQL.
}
```

- **Desafios:**
  - **Mapeamento de Tipo:** Lidar com o tipo vector do PostgreSQL no Java pode exigir configuração. O driver JDBC do PostgreSQL pode mapeá-lo para float, double, ou String. Em alguns casos, pode ser necessário registrar um tipo customizado no Hibernate ou até mesmo criar um Dialeto Hibernate customizado para funções específicas do pgvector, embora para operações básicas de ordenação por distância, o CAST no SQL possa ser suficiente.[35]
  - **SQL Específico do Banco:** A consulta nativa fica atrelada ao PostgreSQL e à sintaxe do pgvector, perdendo a portabilidade oferecida pelo JPA e Spring AI.[1]
  - **Manutenção:** Consultas nativas complexas podem ser mais difíceis de manter e testar.
  - **Limitações:** Spring Data JPA tem limitações com consultas nativas, como a falta de suporte para ordenação dinâmica e a necessidade de fornecer uma countQuery separada para paginação.[34]

O uso de consultas nativas deve ser uma exceção, aplicada apenas quando a abstração do VectorStore for comprovadamente insuficiente para os requisitos da funcionalidade.

5. Design da API (Backend)

A API RESTful é a interface de comunicação entre o backend Spring Boot e o frontend

Angular. Um design claro e consistente é crucial para a integração.

5.1. Princípios REST

A API adere aos princípios RESTful:

- **Statelessness:** Cada requisição do cliente para o servidor deve conter toda a informação necessária para o servidor entender e processar a requisição. O servidor não armazena estado da sessão do cliente entre requisições. A autenticação é geralmente feita via tokens (e.g., JWT) enviados em cada requisição protegida.
- **Interface Uniforme:** Uso de URIs baseadas em recursos (e.g., /api/certificates, /api/activities), métodos HTTP padrão (GET, POST, PUT, DELETE) com semântica bem definida, e representações de recursos (JSON).
- **Operações Padrão:** Utilização dos métodos HTTP conforme sua semântica: GET para leitura, POST para criação, PUT para atualização completa, PATCH para atualização parcial, DELETE para remoção.

5.2. Endpoints Principais da API

A tabela a seguir descreve os endpoints essenciais planejados para o Sistema GAD:

Método HTTP	**Caminho (Path)**	**Descrição**	**Campos Chave (Request Body)**	**Campos Chave (Response Body)**	**Autenticação Requerida**
POST	/api/auth/login	Autenticação do usuário	username, password	accessToken (e.g., JWT)	Não
POST	/api/certificates	Upload de um novo certificado	MultipartFile (arquivo)	CertificateDetailsDTO (com status)	Sim

GET	/api/certificates	Lista certificados submetidos pelo usuário	N/A	List<CertificateSummaryDTO>	Sim
GET	/api/certificates/{id}	Obtém detalhes de um certificado específico	N/A	CertificateDetailsDTO	Sim
GET	/api/activities	Lista atividades validadas do usuário	N/A	List<ActivityDTO>	Sim
GET	/api/activities/summary	Obtém resumo de horas validadas por categoria	N/A	Map<String, Double>	Sim
GET	/api/activities/similar?query={text}	Busca atividades similares por descrição	N/A (Query Param text)	List<ActivityDTO>	Sim

Esta tabela serve como um contrato inicial entre o backend e o frontend, definindo como interagir com os serviços, os dados esperados e retornados, e os requisitos de segurança.

5.3. Data Transfer Objects (DTOs)

Para desacoplar a camada da API das entidades de persistência do banco de dados e

controlar precisamente quais dados são expostos, utiliza-se o padrão Data Transfer Object (DTO). Isso evita expor detalhes internos da implementação do banco de dados e permite moldar os dados especificamente para as necessidades do cliente. Exemplos de DTOs:

- **CertificateUploadDTO:** Pode conter metadados enviados junto com o upload (embora no caso simples, apenas o MultipartFile seja necessário no request).
- **CertificateSummaryDTO:** Usado na listagem, contendo apenas informações essenciais (ID, nome do arquivo, status, data de upload).
- **CertificateDetailsDTO:** Usado para detalhes, incluindo todos os campos do CertificateSummaryDTO mais o texto extraído, mensagens de erro (se houver), e potencialmente um link para a Activity validada.
- **ActivityDTO:** Representa uma atividade validada, contendo ID, descrição, categoria (nome ou código), horas validadas, data da atividade, data da validação.
- **LoginRequestDTO:** Contém username e password.
- **LoginResponseDTO:** Contém o accessToken.

O uso de DTOs é uma prática recomendada em aplicações Spring Boot, similar ao AddressResponse visto em exemplos de consultas nativas.[1] Bibliotecas como ModelMapper ou MapStruct podem auxiliar na conversão entre Entidades e DTOs.

5.4. Documentação da API (Swagger/OpenAPI)

Para facilitar a integração com o frontend e o teste da API, é altamente recomendável gerar documentação interativa usando a especificação OpenAPI (anteriormente Swagger).

- **Integração:** Adicionar a dependência springdoc-openapi-starter-webmvc-ui ao projeto Spring Boot.
- **Uso:** Com a dependência adicionada, o Spring Boot automaticamente gera a descrição da API em formato JSON (geralmente em /v3/api-docs) e expõe uma interface de usuário interativa (Swagger UI, geralmente em /swagger-ui.html).

- **Benefícios:** A Swagger UI permite visualizar todos os endpoints, seus parâmetros, corpos de requisição/resposta esperados (baseados nos DTOs) e até mesmo executar requisições diretamente pelo navegador, o que é extremamente útil para o desenvolvedor frontend e para testes manuais. Anotações adicionais (@Operation, @Parameter, @ApiResponse, etc.) podem ser usadas nos Controllers para enriquecer a documentação gerada.

6. Considerações de Implantação

A implantação do backend do Sistema GAD requer atenção a alguns pontos chave para garantir seu funcionamento correto no ambiente de produção (ou homologação).

6.1. Empacotamento

A aplicação Spring Boot será empacotada como um arquivo JAR executável padrão. Isso inclui o servidor de aplicação embarcado (Tomcat por padrão), simplificando a execução. O comando mvn clean package (ou equivalente Gradle) gera o JAR no diretório target.

6.2. Configuração de Ambiente

As configurações que variam entre ambientes (desenvolvimento, teste, produção) devem ser externalizadas. Isso inclui:

- Credenciais do banco de dados (URL, usuário, senha).[2]
- Chaves de API para serviços externos (se utilizados, e.g., APIs de IA pagas, embora Ollama seja local).[4]
- Endereço do servidor Ollama (se rodando em máquina/container separado).[5]
- Caminho para o diretório tessdata do Tesseract.[7]
- Níveis de log.

O Spring Boot oferece várias maneiras de gerenciar isso, sendo as mais comuns:

- Arquivos application-{profile}.properties (ou .yml) para diferentes perfis (e.g., dev,

prod).
- Variáveis de ambiente do sistema operacional.
- Argumentos de linha de comando ao iniciar a aplicação.

Variáveis de ambiente são geralmente preferidas para informações sensíveis como senhas e chaves de API.

6.3. Opções de Implantação

- **Docker:** A containerização com Docker é fortemente recomendada.
  - Criar um Dockerfile para a aplicação Spring Boot que copie o JAR e defina o comando de execução (java -jar app.jar). O Dockerfile deve usar uma imagem base Java apropriada (e.g., openjdk:17-jdk-slim).
  - Utilizar docker-compose.yml para orquestrar a execução conjunta dos contêineres:
    - Contêiner da aplicação Spring Boot.
    - Contêiner do PostgreSQL com a extensão pgvector habilitada (existem imagens Docker prontas para pgvector).
    - Contêiner do Ollama, garantindo que a rede do Docker Compose permita a comunicação entre a aplicação e o Ollama.
    - O Tesseract OCR precisa ser instalado *dentro* do contêiner da aplicação Spring Boot (adicionar os comandos de instalação no Dockerfile) ou em um contêiner separado acessível pela aplicação.
  - O Docker Compose simplifica a configuração da rede e dos volumes para persistência de dados do PostgreSQL.
- **Nuvem:** Para maior escalabilidade e disponibilidade, a implantação em provedores de nuvem (AWS, Azure, Google Cloud) é uma opção futura. Isso pode envolver:
  - Usar serviços gerenciados de banco de dados (e.g., AWS RDS for PostgreSQL com pgvector, Azure Database for PostgreSQL).
  - Implantar os contêineres Docker em serviços como AWS ECS, AWS EKS, Azure Kubernetes Service (AKS), Google Kubernetes Engine (GKE), ou

plataformas PaaS como Heroku ou Google App Engine.
    - Configurar balanceadores de carga, escalonamento automático e monitoramento.

6.4. Dependências de Runtime

É crucial garantir que as dependências externas estejam disponíveis e acessíveis no ambiente onde o backend será executado:

- **Tesseract OCR:** A engine Tesseract e seus arquivos de idioma (tessdata, incluindo por.traineddata) devem estar instalados no mesmo ambiente (servidor ou contêiner Docker) que a aplicação Spring Boot.[7] O caminho para tessdata deve ser corretamente configurado para a aplicação.
- **Ollama:** O serviço Ollama deve estar em execução e acessível pela rede a partir da aplicação Spring Boot. A URL base do Ollama deve ser configurada corretamente na aplicação.[3] Os modelos de LLM necessários devem ter sido previamente baixados (ollama pull <model_name>) no servidor Ollama, ou a configuração do Spring AI deve permitir o download automático na inicialização (com os devidos cuidados sobre tempo de inicialização).[5]
- **PostgreSQL com pgVector:** O servidor PostgreSQL com a extensão pgvector habilitada deve estar acessível pela rede, e as credenciais de acesso configuradas na aplicação.

7. Conclusão e Trabalhos Futuros

7.1. Resumo da Contribuição do Backend

O backend do Sistema GAD, desenvolvido em Spring Boot, constitui o núcleo lógico e de processamento da aplicação. Ele é responsável por receber os certificados dos estudantes, orquestrar um pipeline complexo que envolve OCR (Tess4J) para extração de texto, Inteligência Artificial (Spring AI com Ollama/LLMs) para análise semântica, extração de informações e classificação de atividades, e validação de regras de negócio específicas do IFMS. Além disso, implementa a persistência de dados em

PostgreSQL, aproveitando a extensão pgvector para funcionalidades avançadas de busca por similaridade, e expõe uma API RESTful documentada para interação com o frontend Angular. A arquitetura modular e a escolha de tecnologias robustas visam fornecer uma base sólida para o gerenciamento eficiente das atividades diversificadas.

7.2. Desafios e Soluções

O desenvolvimento do backend enfrenta desafios técnicos inerentes à natureza do problema:

- **Precisão do OCR:** A variabilidade na qualidade dos certificados (digitalizados, fotos, layouts diversos) representa um desafio para a extração precisa de texto.[25] A solução envolve o uso do Tesseract via Tess4J, com potencial aplicação de técnicas de pré-processamento de imagem e configuração cuidadosa (DPI, idioma).[7]
- **Precisão da IA:** A extração de informações estruturadas (datas, nomes, horas) e a classificação correta das atividades a partir do texto livre extraído dependem da capacidade do LLM e da qualidade dos prompts. A solução adotada utiliza Spring AI com Ollama, permitindo experimentação com diferentes modelos locais [3] e engenharia de prompts iterativa. Modelos específicos para português [18] e fine-tuning [26]-[39] são alternativas caso a precisão inicial não seja satisfatória.
- **Complexidade das Regras de Validação:** As regras de negócio do IFMS (limites de horas, distribuição de excedentes) podem ser complexas. A solução é encapsular essa lógica em um ValidationService dedicado, com métodos claros e testes unitários robustos, e externalizar os parâmetros das regras.
- **Integração de Tecnologias:** Combinar OCR, IA, banco de dados vetorial e regras de negócio em um fluxo coeso requer uma arquitetura bem definida, com processamento assíncrono para tarefas demoradas.

7.3. Melhorias Futuras

O Sistema GAD possui potencial para diversas evoluções futuras:

- **OCR Aprimorado:** Explorar engines de OCR comerciais (baseadas em nuvem ou licenciadas) ou bibliotecas mais avançadas que possam oferecer maior precisão, especialmente para documentos de baixa qualidade.
- **Fine-tuning de Modelos de IA:** Se a precisão da extração ou classificação com modelos pré-treinados gerais for insuficiente, realizar o fine-tuning de um modelo (como BERTimbau [18]) em um dataset específico de certificados do IFMS pode melhorar significativamente o desempenho.[26]
- **Analytics e Relatórios:** Desenvolver um painel administrativo com visualizações e relatórios sobre as atividades submetidas, horas validadas por curso/período, categorias mais populares, etc., utilizando os dados agregados.
- **Integração com Sistemas Acadêmicos:** Conectar o GAD ao portal acadêmico principal do IFMS para sincronização de dados de estudantes e, potencialmente, registro automático das horas validadas no histórico escolar.
- **Aplicação Móvel:** Desenvolver um frontend mobile (nativo ou PWA) para facilitar o envio de certificados pelos estudantes através de seus smartphones.
- **RAG Avançado:** Implementar técnicas de Retrieval-Augmented Generation mais sofisticadas, utilizando os componentes modulares do LangChain4j [11] ou Spring AI, para melhorar a busca por similaridade ou habilitar funcionalidades de chatbot que respondam perguntas sobre o regulamento de atividades com base nos documentos normativos carregados no sistema.
- **Validação de Autenticidade Aprimorada:** Expandir a validação de QR code [15] ou integrar com sistemas de verificação de certificados digitais, se disponíveis.

Estas sugestões representam caminhos para tornar o Sistema GAD uma ferramenta ainda mais completa e integrada ao ecossistema acadêmico do IFMS Naviraí.

8. Referências

- [7] https://learnerbits.com/ocr-tesseract-in-spring-boot-2024/
- [25] https://www.geeksforgeeks.org/tesseract-ocr-with-java-with-examples/
- [8] https://www.tutorialspoint.com/tesseract-ocr-with-java-with-examples

- [24] https://stackoverflow.com/questions/64708101/how-to-set-up-tesseract-ocr-in-java
- [9] https://www.mscharhag.com/page/6
- [3] https://piotrminkowski.com/2025/03/10/using-ollama-with-spring-ai/
- [4] https://piotrminkowski.com/2025/01/28/getting-started-with-spring-ai-and-chat-model/
- [5] https://docs.spring.io/spring-ai/reference/api/chat/ollama-chat.html
- [17] https://www.youtube.com/watch?v=5HX1ZPXN-os
- [6] https://github.com/tzolov/spring-ai-ollama-huggingface-demo
- [1] https://www.geeksforgeeks.org/spring-boot-jpa-native-query-with-example/
- [38] https://www.youtube.com/watch?v=AFG8IDX7iUg
- [2] https://docs.spring.io/spring-ai/reference/api/vectordbs/pgvector.html
- [40] https://www.youtube.com/watch?v=yXwHu6g-wlY
- [34] https://www.baeldung.com/spring-data-jpa-query
- [35] https://cloud.google.com/spanner/docs/use-spring-data-jpa-postgresql
- [36] https://www.baeldung.com/spring-data-jpa-custom-database-functions
- [37] https://dev.to/tolgee_i18n/creating-custom-hibernate-dialect-1bb9
- [33] https://docs.spring.io/spring-ai/reference/api/vectordbs.html
- [32] https://dev.to/mcadariu/springai-llama3-and-pgvector-bragging-rights-2n8o
- [15] https://www.blackslate.io/articles/generate-and-read-qr-code-in-java-using-zxing-library
- [16] https://www.geeksforgeeks.org/how-to-generate-and-read-qr-code-with-java-using-zxing-library/
- [29] https://www.codeproject.com/Articles/5387296/Generating-and-Reading-QR-Codes-in-a-Java-Applicat

- [30] https://stackoverflow.com/questions/18863466/reading-qrcode-with-zxing-in-java
- [31] https://stackoverflow.com/questions/2489048/qr-code-encoding-and-decoding-using-zxing
- [10] https://quarkus.io/blog/using-langchain4j-to-analyze-pdf-documents/
- [11] https://github.com/langchain4j/langchain4j/blob/main/docs/docs/tutorials/rag.md
- [12] https://dev.to/mongodb/how-to-make-a-rag-application-with-langchain4j-1mad
- [13] https://kindgeek.com/blog/post/experiments-with-langchain4j-or-java-way-to-llm-powered-applications
- [14] https://rito.hashnode.dev/hands-on-guide-to-building-llm-apps-in-java-with-langchain4j-and-qdrant-db
- [18] https://huggingface.co/neuralmind/bert-base-portuguese-cased
- [21] https://huggingface.co/bastao/PeroVaz_PT-BR_Classifier
- [22] https://huggingface.co/lisaterumi/postagger-portuguese
- [19] https://huggingface.co/models?other=base_model:finetune:neuralmind/bert-base-portuguese-cased
- [23] https://huggingface.co/jcfneto/bert-br-portuguese
- [26] https://www.kaggle.com/code/neerajmohan/fine-tuning-bert-for-text-classification
- [27] https://towardsdatascience.com/fine-tuning-bert-for-text-classification-a01f89b179fc/
- [28] https://www.researchgate.net/publication/382689134_Fine-Tuning_of_Distil-BERT_for_Continual_Learning_in_Text_Classification_An_Experimental_Analysis
- [20] https://huggingface.co/lucasbalponti/fine-tuned-bertimbau-for-legal-area-classificati

on-v1

- [39] https://huggingface.co/lucasbalponti/fine-tuned-bertimbau-for-legal-area-classification-v1/tree/dfca0d89993a53c0006ea981cfa3de35b6cf1458

Referências citadas

1. Spring Boot JPA Native Query with Example | GeeksforGeeks, acessado em maio 4, 2025, https://www.geeksforgeeks.org/spring-boot-jpa-native-query-with-example/
2. PGvector :: Spring AI Reference, acessado em maio 4, 2025, https://docs.spring.io/spring-ai/reference/api/vectordbs/pgvector.html
3. Using Ollama with Spring AI - Piotr's TechBlog, acessado em maio 4, 2025, https://piotrminkowski.com/2025/03/10/using-ollama-with-spring-ai/
4. Getting Started with Spring AI and Chat Model - Piotr's TechBlog, acessado em maio 4, 2025, https://piotrminkowski.com/2025/01/28/getting-started-with-spring-ai-and-chat-model/
5. Ollama Chat :: Spring AI Reference, acessado em maio 4, 2025, https://docs.spring.io/spring-ai/reference/api/chat/ollama-chat.html
6. This project demonstrates the integration of Spring AI with Ollama, leveraging Hugging Face GGUF models for both chat completion and embedding tasks. - GitHub, acessado em maio 4, 2025, https://github.com/tzolov/spring-ai-ollama-huggingface-demo
7. OCR-Tesseract in Spring Boot 2024 - LearnerBits, acessado em maio 4, 2025, https://learnerbits.com/ocr-tesseract-in-spring-boot-2024/
8. Tesseract OCR with Java Examples - Tutorialspoint, acessado em maio 4, 2025, https://www.tutorialspoint.com/tesseract-ocr-with-java-with-examples
9. Optical character recognition (OCR) is the conversion of images containing text to machine-encoded text. A popular tool for this is the open source project Tesseract.

Tesseract can be used as standalone application from the command line. Alternatively it can be integrated into applications using its C++ API. For other programming languages various wrapper APIs are available. In this post we will use the Java Wrapper Tess4J. - Michael Scharhag's software development blog about Java technologies including Java EE, Spring and REST, acessado em maio 4, 2025, https://www.mscharhag.com/page/6

10. Using LangChain4j to analyze PDF documents - Quarkus, acessado em maio 4, 2025, https://quarkus.io/blog/using-langchain4j-to-analyze-pdf-documents/
11. langchain4j/docs/docs/tutorials/rag.md at main - GitHub, acessado em maio 4, 2025, https://github.com/langchain4j/langchain4j/blob/main/docs/docs/tutorials/rag.md
12. How to Make a RAG Application With LangChain4j - DEV Community, acessado em maio 4, 2025, https://dev.to/mongodb/how-to-make-a-rag-application-with-langchain4j-1mad
13. How to incorporate LM/LLM features into Java using Langchain4j - Kindgeek, acessado em maio 4, 2025, https://kindgeek.com/blog/post/experiments-with-langchain4j-or-java-way-to-llm-powered-applications
14. Hands-on Guide to Building LLM Apps in Java with Langchain4j and Qdrant DB, acessado em maio 4, 2025, https://rito.hashnode.dev/hands-on-guide-to-building-llm-apps-in-java-with-langchain4j-and-qdrant-db
15. Generate and Read QR code in Java using Zxing Library - BlackSlate, acessado em maio 4, 2025, https://www.blackslate.io/articles/generate-and-read-qr-code-in-java-using-zxing-library
16. How to generate and read QR code with Java using ZXing Library - GeeksforGeeks, acessado em maio 4, 2025, https://www.geeksforgeeks.org/how-to-generate-and-read-qr-code-with-java-using

-zxing-library/

17. #3 Spring AI | Ollama Setup - YouTube, acessado em maio 4, 2025, https://www.youtube.com/watch?v=5HX1ZPXN-os
18. neuralmind/bert-base-portuguese-cased - Hugging Face, acessado em maio 4, 2025, https://huggingface.co/neuralmind/bert-base-portuguese-cased
19. Models - Hugging Face, acessado em maio 4, 2025, https://huggingface.co/models?other=base_model:finetune:neuralmind/bert-base-portuguese-cased
20. lucasbalponti/fine-tuned-bertimbau-for-legal-area-classification-v1 - Hugging Face, acessado em maio 4, 2025, https://huggingface.co/lucasbalponti/fine-tuned-bertimbau-for-legal-area-classification-v1
21. bastao/PeroVaz_PT-BR_Classifier - Hugging Face, acessado em maio 4, 2025, https://huggingface.co/bastao/PeroVaz_PT-BR_Classifier
22. lisaterumi/postagger-portuguese - Hugging Face, acessado em maio 4, 2025, https://huggingface.co/lisaterumi/postagger-portuguese
23. jcfneto/bert-br-portuguese - Hugging Face, acessado em maio 4, 2025, https://huggingface.co/jcfneto/bert-br-portuguese
24. How to set up Tesseract OCR in Java? - Stack Overflow, acessado em maio 4, 2025, https://stackoverflow.com/questions/64708101/how-to-set-up-tesseract-ocr-in-java
25. Tesseract OCR with Java with Examples - GeeksforGeeks, acessado em maio 4, 2025, https://www.geeksforgeeks.org/tesseract-ocr-with-java-with-examples/
26. Fine-tuning BERT for Text classification - Kaggle, acessado em maio 4, 2025, https://www.kaggle.com/code/neerajmohan/fine-tuning-bert-for-text-classification
27. Fine-Tuning BERT for Text Classification | Towards Data Science, acessado em maio 4, 2025, https://towardsdatascience.com/fine-tuning-bert-for-text-classification-a01f89b179fc/

28. (PDF) Fine-Tuning of Distil-BERT for Continual Learning in Text Classification: An Experimental Analysis - ResearchGate, acessado em maio 4, 2025, https://www.researchgate.net/publication/382689134_Fine-Tuning_of_Distil-BERT_for_Continual_Learning_in_Text_Classification_An_Experimental_Analysis
29. Generating and Reading QR Codes in a Java Application using ZXing Library., acessado em maio 4, 2025, https://www.codeproject.com/Articles/5387296/Generating-and-Reading-QR-Codes-in-a-Java-Applicat
30. Reading QRCode with Zxing in Java - Stack Overflow, acessado em maio 4, 2025, https://stackoverflow.com/questions/18863466/reading-qrcode-with-zxing-in-java
31. QR Code encoding and decoding using zxing - java - Stack Overflow, acessado em maio 4, 2025, https://stackoverflow.com/questions/2489048/qr-code-encoding-and-decoding-using-zxing
32. Spring AI, Llama 3 and pgvector: bRAGging rights! - DEV Community, acessado em maio 4, 2025, https://dev.to/mcadariu/springai-llama3-and-pgvector-bragging-rights-2n8o
33. Vector Databases :: Spring AI Reference, acessado em maio 4, 2025, https://docs.spring.io/spring-ai/reference/api/vectordbs.html
34. Spring Data JPA @Query - Baeldung, acessado em maio 4, 2025, https://www.baeldung.com/spring-data-jpa-query
35. Integrate Spanner with Spring Data JPA (PostgreSQL dialect) - Google Cloud, acessado em maio 4, 2025, https://cloud.google.com/spanner/docs/use-spring-data-jpa-postgresql
36. Calling Custom Database Functions With JPA and Spring Boot - Baeldung, acessado em maio 4, 2025, https://www.baeldung.com/spring-data-jpa-custom-database-functions
37. Creating Custom Hibernate Dialect - DEV Community, acessado em maio 4, 2025,

https://dev.to/tolgee_i18n/creating-custom-hibernate-dialect-1bb9

38. Native Queries with Spring Data JPA - YouTube, acessado em maio 4, 2025, https://www.youtube.com/watch?v=AFG8lDX7iUg
39. lucasbalponti/fine-tuned-bertimbau-for-legal-area-classification-v1 at dfca0d89993a53c0006ea981cfa3de35b6cf1458 - Hugging Face, acessado em maio 4, 2025, https://huggingface.co/lucasbalponti/fine-tuned-bertimbau-for-legal-area-classification-v1/tree/dfca0d89993a53c0006ea981cfa3de35b6cf1458
40. Spring Data JPA Native Query Examples - YouTube, acessado em maio 4, 2025, https://www.youtube.com/watch?v=yXwHu6g-wlY