Relatório Técnico: GED IFMS - Estratégia de Arquitetura Backend e Integração de IA

I. Introdução e Refinamento do Problema

- **A. Visão Geral do Projeto:**
  - O objetivo central deste projeto de Trabalho de Conclusão de Curso (TCC) é o desenvolvimento do sistema GED IFMS (Gerenciamento Eletrônico de Documentos) para o Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso do Sul (IFMS) Câmpus Naviraí. O sistema visa automatizar o gerenciamento, a validação e a contabilização de certificados de atividades complementares submetidos pelos alunos.
  - As responsabilidades do projeto estão divididas: Vitor D. se concentrará no desenvolvimento do backend e na integração da Inteligência Artificial (IA), enquanto Mateus A. cuidará do frontend e da pesquisa geral. Este relatório foca nas responsabilidades de backend e IA atribuídas a Vitor.
  - É fundamental reconhecer o contexto de um TCC, com um cronograma definido para conclusão até julho [Audio 01, Audio 11, Audio 12]. Portanto, a ênfase recai sobre a criação de uma solução prática, implementável e robusta dentro deste prazo, priorizando a funcionalidade essencial sobre a complexidade desnecessária ou pesquisa inédita [Audio 08, Audio 10, Audio 11].
- **B. Definição do Problema Central:**
  - O desafio principal reside na transição do atual processo manual de gerenciamento de certificados – descrito como suscetível a erros e falta de informação – para um sistema automatizado que utilize IA para otimizar o fluxo de trabalho.
  - As funcionalidades chave incluem: upload de documentos pelos alunos, extração automática de informações relevantes (nome do aluno, título da atividade, carga horária, datas, entidade emissora), classificação das

atividades em categorias predefinidas (UC, PE, PP, PD, PA, PV, TCC) utilizando IA, validação assistida por IA das informações submetidas pelos alunos (conforme sugestão do professor [Audio 05, Audio 06]), cálculo complexo de horas (considerando limites por categoria e redistribuição de excedentes) e, potencialmente, verificação de autenticidade dos certificados (via QR codes [Audio 06, Audio 10]).

- **C. Desafios Técnicos e Considerações:**
  - **Variabilidade dos Documentos:** Os certificados provavelmente serão submetidos em diversos formatos (PDFs baseados em texto, PDFs digitalizados como imagem, arquivos de imagem padrão como PNG e JPG) e apresentarão layouts variados. Isso exige métodos de extração de texto robustos e adaptáveis.
  - **Ambiguidade na Classificação:** Conforme observado na consulta inicial, algumas atividades podem se enquadrar em múltiplas categorias. A definição clara dessas categorias, suas sobreposições e as regras para classificação múltipla são cruciais para uma IA eficaz.
  - **Complexidade da Integração de IA:** A seleção, integração e gerenciamento eficientes e sustentáveis das ferramentas de IA (OCR, LLMs, modelos de ML) dentro do ecossistema Spring representam um desafio significativo. É necessário balancear a precisão dos modelos com o desempenho e o consumo de recursos computacionais.
  - **Lógica de Negócios Específica:** A implementação precisa das regras institucionais para limites de carga horária por categoria e a redistribuição das horas excedentes é um requisito complexo.
  - **Fluxo de Validação:** O desenho de um processo claro que envolva a submissão do aluno, a validação pela IA e a possibilidade de revisão manual por professores ou validadores é essencial para a confiabilidade do sistema [Audio 05, Audio 06, Audio 07].

II. Estratégia de Processamento de Certificados e Extração de Dados

- **A. Ingestão de Documentos e Extração de Texto:**
  - **Manuseio de Uploads:** Utilizar os mecanismos padrão do Spring Boot para lidar com uploads de arquivos via MultipartFile. É importante configurar limites de tamanho de arquivo adequados (por exemplo, usando spring.servlet.multipart.max-file-size [1]) para evitar problemas com documentos grandes.
  - **Tratamento Inicial do Formato:** O primeiro passo no processamento é determinar o tipo de arquivo recebido: um PDF com texto selecionável, um PDF que é essencialmente uma imagem digitalizada, ou um formato de imagem padrão (PNG, JPG). Uma verificação inicial no serviço de processamento pode direcionar o fluxo.[2]
  - **Extração de Texto de PDFs Baseados em Texto:**
    - Para PDFs onde o texto já existe digitalmente, a extração direta é preferível.
    - **Abordagem com Spring AI:** Recomenda-se o uso das implementações de DocumentReader fornecidas pelo Spring AI, como PagePdfDocumentReader ou ParagraphPdfDocumentReader.[3] Essas classes se integram naturalmente ao pipeline ETL (Extract, Transform, Load) do Spring AI [4], utilizando a biblioteca Apache PDFBox internamente para a análise do PDF.[3] A classe Document do Spring AI encapsula o conteúdo textual e os metadados do arquivo.[4]
    - **Abordagem com Langchain4j:** Como alternativa, Langchain4j oferece carregadores de documentos (Document Loaders) como FileSystemDocumentLoader em conjunto com ApachePdfBoxDocumentParser.[11] Langchain4j também abstrai o carregamento de documentos de várias fontes.[13]
    - *Análise Comparativa:* A integração nativa do DocumentReader do Spring AI [4] com o restante do ecossistema Spring parece mais alinhada para este

projeto, que tem o Spring como framework principal.[14] Langchain4j é uma ferramenta poderosa [16], mas introduz uma dependência de framework adicional que pode não ser estritamente necessária apenas para a leitura de PDFs. A menos que outras funcionalidades avançadas do Langchain4j sejam críticas, a abordagem Spring AI é preferível pela simplicidade e coesão.

- **Extração de Texto de PDFs Digitalizados/Imagens (OCR):**
  - Quando o PDF contém apenas imagens de páginas ou o upload é diretamente uma imagem, o Reconhecimento Óptico de Caracteres (OCR) é necessário.
  - **Tesseract OCR:** É o motor OCR de código aberto mais referenciado e recomendado para iniciar.[2] A menção do professor sobre testar documentos variados sugere a possibilidade de certificados digitalizados [Audio 02].
  - **Integração Java (Tess4J):** A biblioteca Tess4J (net.sourceforge.tess4j) atua como um wrapper Java (usando JNA) para a API nativa do Tesseract.[2] Isso requer a adição da dependência Maven correspondente [2] e a instalação prévia do próprio motor Tesseract no ambiente do servidor onde a aplicação Spring Boot será executada.[2]
  - **Passos de Implementação:**
    1. Instalar o motor Tesseract OCR no ambiente de execução (máquina local, servidor da universidade, etc.).
    2. Baixar os arquivos de dados de idioma necessários (arquivos .traineddata, como por.traineddata para português) e configurar o caminho para o diretório tessdata no código Java usando tesseract.setDatapath().[18] A localização padrão varia conforme o sistema operacional.[23]
    3. Se o input for um PDF digitalizado, é necessário primeiro converter cada página do PDF em uma imagem. Bibliotecas como Apache

PDFBox [2] podem renderizar páginas PDF para BufferedImage. O snippet [2] demonstra essa renderização.

4. Passar o objeto File ou BufferedImage resultante para o método tesseract.doOCR() da instância Tess4J.[2]
5. Considerar o pré-processamento da imagem (conversão para escala de cinza, ajuste de contraste, aumento de DPI usando tesseract.setTessVariable("user_defined_dpi", "300")) pode melhorar significativamente a precisão do OCR, especialmente para imagens de baixa qualidade.[2]

- **Integração com Spring Boot:** Encapsular a lógica do Tess4J dentro de um serviço Spring (@Service).[2] Este serviço deve gerenciar a criação e exclusão de arquivos temporários necessários para processar os MultipartFiles recebidos.[2] Existem projetos de exemplo disponíveis que demonstram essa integração.[24]
- **Alternativas de OCR:** Serviços de OCR em nuvem (como Google Cloud Vision API, Amazon Textract [21]) ou as capacidades de visão de LLMs mais recentes (por exemplo, GPT-4o acessado via Spring AI [29], Google Gemini [16], ou potencialmente o discutido Mistral OCR [31]) são alternativas. No entanto, podem introduzir custos ou complexidade de integração que podem ser excessivos para o escopo do TCC. Ferramentas como Zerox [32] também utilizam modelos de visão para OCR. OCRmyPDF [33] é uma ferramenta popular que envolve o Tesseract, frequentemente usada para criar PDFs pesquisáveis a partir de digitalizações.
- *Considerações sobre Tesseract/Tess4J:* A integração do Tesseract/Tess4J introduz dependências externas significativas: a instalação do motor Tesseract nativo e dos pacotes de idioma.[2] Além disso, o OCR é uma operação computacionalmente intensiva, especialmente para documentos de múltiplas páginas, o que pode impactar o desempenho e exigir mais recursos do servidor.[35] É crucial garantir que o ambiente de implantação

(seja local ou um servidor da universidade) possa suportar essa configuração e carga. As APIs de OCR na nuvem [21] evitam a configuração local, mas trazem chamadas de rede externas e potenciais custos. Os LLMs com capacidade de visão [29], acessíveis via Spring AI, podem ser mais simples de invocar, mas podem ser mais lentos ou caros que o Tesseract local para a extração pura de texto.

- **Recomendação:** Iniciar com o PagePdfDocumentReader do Spring AI para PDFs baseados em texto. Implementar a integração Tesseract/Tess4J dentro de um serviço dedicado para lidar com PDFs digitalizados e imagens, utilizando PDFBox para renderizar as páginas conforme necessário. Manter as opções de OCR na nuvem ou via LLM como alternativas ou melhorias futuras.

- **B. Extração de Informações Estruturadas (Potencializada por IA):**
  - **Objetivo:** Extrair campos específicos do texto obtido na etapa anterior, como Nome do Aluno, Título da Atividade, Carga Horária, Datas (início/fim), Órgão Emissor, etc.
  - **Desafios:** A extração precisa ser resiliente a variações na formulação e no layout dos diferentes certificados.
  - **Abordagens:**
    - **Expressões Regulares (Regex)/Baseada em Regras:** Abordagem frágil e difícil de manter, pois pequenas variações nos certificados podem quebrar as regras. Adequada apenas para campos extremamente padronizados, se houver.
    - **Reconhecimento de Entidades Nomeadas (NER) Tradicional:** Possível com bibliotecas como Stanford CoreNLP [36], mas geralmente requer treinamento de modelos específicos para dados de certificados ou o uso de modelos pré-treinados que podem não ser adequados, representando um esforço considerável para um TCC.
    - **LLMs para Saída Estruturada (Recomendado):** Utilizar a capacidade dos LLMs de compreender o contexto do texto extraído e extrair as informações

solicitadas em um formato predefinido.

- **Spring AI ChatClient:** A API fluente do ChatClient (ChatClient.prompt()...call().entity(SeuRecord.class)) é a forma mais idiomática no Spring AI para essa tarefa.[1] O Spring AI pode gerar automaticamente um esquema JSON a partir da classe/record Java de destino e instruir o modelo (especialmente modelos OpenAI que suportam saída estruturada de forma nativa [40]) a retornar uma resposta JSON que corresponda a esse esquema.[30]
- **Conversores:** Para controle explícito, o Spring AI oferece BeanOutputConverter, MapOutputConverter e ListOutputConverter.[37] O BeanOutputConverter é ideal neste cenário para mapear a saída diretamente para um record/POJO Java CertificateData.[37]
- **Engenharia de Prompt:** É crucial criar um prompt claro que instrua o LLM a extrair os campos específicos do texto do certificado fornecido e a retorná-los no formato desejado (implicitamente gerenciado pelo método .entity() ou explicitamente com conversores).[1] O prompt deve listar os campos a serem extraídos (Nome do Aluno, Título da Atividade, Carga Horária, Data de Início, Data de Fim, etc.).
- **Escolha do Modelo:** Modelos como GPT-4o/GPT-4o-mini [30], Gemini [16], ou modelos locais via Ollama (como Llama3, Mixtral) podem ser acessados através do Spring AI.[14] Os modelos da OpenAI oferecem suporte explícito e mais confiável para saída estruturada.[40] Outros modelos dependem da adesão ao prompt, o que pode ser menos consistente.[40]

- **Spring AI Function Calling:** Uma alternativa é definir funções Java (por exemplo, extractCertificateData(String text)) com descrições claras e esquemas de parâmetros (derivados implicitamente de POJOs).[46] O LLM pode então "chamar" essa função, retornando os argumentos (os dados extraídos) em um formato JSON estruturado.[46] Conceitualmente similar à

saída estruturada, mas enquadrado como uma chamada de ferramenta. A configuração pode ser ligeiramente mais complexa que ChatClient.entity(). O Azure também oferece funções de extração de entidades.[49]

- **Recomendação:** Utilizar o método ChatClient.call().entity(CertificateData.class) do Spring AI, definindo um Record Java CertificateData bem estruturado para conter os campos a serem extraídos. Esta é a abordagem mais simples e alinhada com o Spring AI para extração estruturada. Começar com um modelo Ollama local para desenvolvimento/teste e considerar a API da OpenAI para maior confiabilidade na saída estruturada, se necessário e permitido. Elaborar um prompt de sistema claro instruindo o modelo sobre a tarefa de extração.
- *Confiabilidade da Extração:* A eficácia da extração baseada em LLM depende fortemente da qualidade do prompt e das capacidades do modelo, particularmente sua habilidade em aderir a formatos de saída estruturados.[40] Para modelos que não sejam da OpenAI, pode ser necessário implementar validação adicional da estrutura JSON retornada no código Java. Modelos podem, por vezes, falhar em gerar JSON válido ou aderir ao esquema solicitado.[40] Portanto, ao usar modelos via Ollama, o código da aplicação deve incluir tratamento de exceções (blocos try-catch) em torno da chamada .entity() ou da etapa de conversão. Implementar lógica de fallback (por exemplo, registrar um erro e sinalizar para revisão manual) caso a análise falhe adiciona uma camada de robustez importante.

III. Classificação e Validação de Atividades Orientadas por IA

- **A. Definição da Tarefa de Classificação:**
  - O objetivo é atribuir uma ou mais das categorias predefinidas (UC, PE, PP, PD, PA, PV, TCC) a cada certificado processado. A classificação deve se basear no conteúdo textual extraído, especialmente no título e na descrição da atividade.
  - A entrada para este módulo é o texto extraído (por exemplo, do objeto

CertificateData) e a saída esperada é uma lista de códigos de categoria aplicáveis (ex: ["PA", "PV"]).

- **B. Comparação das Abordagens de Classificação:**

Abordagem	Prós	Contras	Complexidade (TCC)	Requisitos de Dados	Recomendação GED IFMS
Palavra-chave/Regras	Simples de implementar inicialmente. Rápido.	Frágil, difícil de manter, baixa precisão para nuances, não lida bem com ambiguidades.	Baixa	Nenhum (regras manuais)	Apenas como baseline ou auxiliar.
Machine Learning (ML)	Potencialmente alta precisão com dados suficientes. Modelos específicos para a tarefa.	Requer dataset rotulado significativo, treinamento, tuning, infraestrutura de ML. Curva de aprendizado maior.	Alta	Dataset rotulado extenso [Audio 02]	Não recomendado inicialmente.
LLM (Prompting)	Flexível, adapta-se à	Dependente da qualidade	Média	Nenhum (usa	**Recomendado**.

	linguagem natural, não requer dataset de treino específico, rápido de prototipar.	do prompt e do modelo, pode ser inconsistente, latência/custo de API.[52]		pré-treino do LLM)	
LLM + RAG	Melhora precisão para casos específicos do domínio, usa documentos existentes como conhecimento.[5]	Adiciona complexidade (Vector DB, embeddings, recuperação), mais lento que prompting direto.[5]	Alta	Documentos de exemplo/descrições	Melhoria futura, se necessário.

- **Análise Detalhada das Abordagens:**
    - **Baseada em Palavras-chave/Regras:** Embora fácil de começar (ex: verificar a presença de "pesquisa" para classificar como PE), essa abordagem rapidamente se torna insustentável devido à diversidade de descrições de atividades. Pode servir como um filtro inicial ou para categorias muito distintas, mas não como solução principal.
    - **Machine Learning Tradicional:**
        - *Algoritmos:* Classificadores como Naive Bayes (bom ponto de partida, rápido [53, 54]), SVM (frequentemente eficaz em texto [53, 54, 55]), Regressão Logística [54] e Random Forest [56] são opções clássicas. Redes neurais profundas (CNNs, LSTMs [53, 57]) podem oferecer maior precisão, mas exigem mais dados e complexidade.[54]

        - *Features:* O texto precisa ser convertido em vetores numéricos usando técnicas como TF-IDF [55, 56] ou, preferencialmente, embeddings de palavras/sentenças (como GloVe [56] ou gerados por modelos como nomic-embed-text via Ollama/Spring AI [58]) que capturam melhor o significado semântico.[56]
        - *Ferramentas:* Scikit-learn (Python) é o padrão. Weka e Orange (mencionados na consulta) são ferramentas com GUI, úteis para exploração inicial (o professor mencionou Weka [Audio 09, Audio 10]), mas menos adequadas para integração direta no backend Spring. Integrar um modelo treinado em Python exigiria uma API separada ou o uso de bibliotecas Java de ML (geralmente menos maduras).
        - *Necessidade de Dados:* O principal obstáculo é a necessidade de um dataset substancial de certificados previamente classificados manualmente para treinar e avaliar o modelo. A criação deste dataset pode consumir um tempo significativo do cronograma do TCC.
        - *Viabilidade no TCC:* Embora o professor tenha mencionado a existência de uma base de documentos [Audio 02], ele também enfatizou o foco na implementação prática e o uso de ferramentas existentes, sem a necessidade de resultados inéditos ou aprofundamento em pesquisa de ML [Audio 08, Audio 10, Audio 11]. O processo completo de treinar um classificador ML (preparação de dados, engenharia de features, seleção/treinamento/avaliação de modelo, implantação com ferramentas como MLFlow) é uma tarefa substancial que pode conflitar com o prazo de julho [Audio 01].
    - **Modelos de Linguagem Grandes (LLMs):**
        - *Abordagem:* Utilizar LLMs diretamente para a tarefa de classificação através de prompts bem elaborados (Zero-shot ou Few-shot).
        - *Implementação (Spring AI):* Usar o ChatClient de forma similar à extração de dados. Fornecer o texto extraído do certificado e a lista de categorias possíveis (com suas descrições) no prompt. Instruir o modelo a retornar a(s) categoria(s) mais apropriada(s).[52, 59]
        - *Técnicas de Prompting:*
            - *Zero-Shot:* Pedir diretamente a classificação com base nas

descrições das categorias.[60] Exemplo: "Classifique a atividade descrita abaixo em uma ou mais das seguintes categorias: [Lista de categorias com descrições]. Título da Atividade: '...' Descrição: '...'. Retorne apenas os códigos das categorias aplicáveis."

- *Few-Shot:* Incluir exemplos de atividades já classificadas dentro do próprio prompt para guiar o modelo.[60] Isso pode aumentar a precisão em casos ambíguos, mas também aumenta o tamanho e o custo do prompt.

- *Saída Estruturada:* Usar .entity(List.class) ou ListOutputConverter [37, 38, 43] para obter o resultado da classificação como uma lista de strings (os códigos das categorias).
- *Vantagens:* Não requer dados de treinamento específicos (aproveita o conhecimento pré-treinado do modelo), é flexível e adapta-se bem a descrições em linguagem natural.
- *Desvantagens:* O desempenho depende da capacidade do modelo e da qualidade do prompt, pode haver inconsistências, e há latência/custo associados às chamadas de API (se usar modelos na nuvem).

- **Geração Aumentada por Recuperação (RAG) para Classificação:**
  - *Conceito:* Se a classificação for muito específica do domínio ou depender de conhecimento institucional não presente no LLM base, o RAG pode ajudar. Armazenar exemplos de certificados ou descrições detalhadas de atividades por categoria em um banco de dados vetorial (como o pgVector).
  - *Processo:* 1) Gerar um embedding (vetor numérico) para o texto do certificado de entrada. 2) Realizar uma busca por similaridade no pgVector para encontrar os exemplos/descrições mais relevantes.[58, 61, 62, 63, 64, 65] 3) Incluir esses exemplos recuperados no contexto do prompt enviado ao LLM para a classificação final.[5, 8, 66]
  - *Implementação (Spring AI):* Usar EmbeddingClient para criar os embeddings, VectorStore configurado para pgVector [61, 65] para armazenar e buscar, e ChatClient para enviar o prompt final com os documentos recuperados adicionados ao contexto.[8, 67, 68] O pipeline ETL do Spring AI (DocumentReader, DocumentTransformer,

DocumentWriter) pode ser usado para carregar os dados no banco vetorial.[4, 5, 6, 9]

- *Vantagens:* Pode aumentar a precisão para classificações específicas do domínio, aproveitando a base de documentos existente [Audio 02].
- *Desvantagens:* Adiciona complexidade significativa (configuração do banco vetorial, gerenciamento de embeddings, etapa de recuperação). Pode ser excessivo se o prompting direto com LLM for suficiente.

- **C. Implementando a Validação por IA:**
  - **Sugestão do Professor:** O fluxo ideal envolve o aluno submetendo o certificado juntamente com sua sugestão de categoria e outros dados (horas, título). A IA, então, valida essa entrada do aluno comparando-a com o conteúdo extraído do documento [Audio 05, Audio 06].
  - **Fluxo de Trabalho Proposto:**
    1. Aluno faz upload do certificado e fornece dados iniciais (categoria sugerida, horas, etc.).
    2. O backend extrai o texto do certificado (usando os métodos da Seção II.A).
    3. O backend invoca um LLM (via Spring AI ChatClient) especificamente para a tarefa de validação.
  - **Design do Prompt de Validação:**
    - O prompt enviado ao LLM deve conter:
      - O texto completo extraído do certificado.
      - As informações submetidas pelo aluno (categoria, horas, título).
      - A lista de categorias válidas e suas descrições oficiais.
      - Instruções claras e objetivas: "Valide se a categoria '{categoria_aluno}' e as horas '{horas_aluno}' submetidas pelo aluno são consistentes com as informações presentes no texto do certificado fornecido. Verifique também a precisão do título da atividade. O texto do certificado é:

'{texto_certificado}'. Com base neste texto: a categoria informada pelo aluno está correta? As horas mencionadas são consistentes? O título é preciso? Explique quaisquer discrepâncias encontradas. Retorne um resultado de validação (ex: 'Válido', 'Categoria Inválida', 'Horas Inválidas', 'Discrepância Encontrada') e uma breve explicação."

    - Utilizar saída estruturada (por exemplo, um record Java ValidationResult com campos para status, explicação, e talvez sugestões de categoria/horas se houver divergência) através de ChatClient.entity() [37] para facilitar o processamento da resposta no backend.
    - *Reforçando a Robustez do Prompt:* É crucial que o prompt de validação seja formulado para evitar que a IA simplesmente concorde com a entrada do aluno. O LLM deve ser explicitamente instruído a *comparar* a entrada do aluno *com* o conteúdo do documento. Instruções como "Explique quaisquer discrepâncias" forçam uma análise mais crítica. Solicitar uma saída estruturada com campos separados para 'status' e 'explicação' também incentiva uma resposta detalhada e verificável, mitigando o risco de o modelo ser excessivamente complacente ou ser vítima de "prompt injection".[52]

- **D. Recomendação:**
    - Iniciar com a classificação baseada em LLM usando prompting Zero-Shot via Spring AI ChatClient e saída estruturada para a lista de categorias. Esta abordagem se alinha melhor com o escopo do TCC e as orientações do professor.
    - Implementar o fluxo de Validação por IA conforme descrito acima, utilizando um prompt de validação dedicado e saída estruturada para o resultado.
    - Manter o ML Tradicional e o RAG como potenciais melhorias futuras caso a abordagem inicial com LLM se mostre insuficiente, mas evitá-los inicialmente para gerenciar a complexidade do projeto.

IV. Arquitetura Backend e Pilha Tecnológica

- **A. Estrutura da Aplicação Spring Boot:**
  - **Modularidade:** Recomenda-se uma estrutura de projeto bem organizada, seja em um único módulo ou, se a complexidade justificar, em múltiplos módulos Maven (embora possa ser excessivo para o TCC inicial). Uma estrutura de pacotes sugerida para um único módulo seria:
    - com.gedifms.api: Controladores REST (ex: CertificateController, StudentController).
    - com.gedifms.service: Lógica de negócios (ex: CertificateService, ClassificationService, HourCalculationService).
    - com.gedifms.ai: Componentes de integração de IA (ex: OcrService, ExtractionService, ValidationService, wrappers de cliente Spring AI).
    - com.gedifms.repository: Repositórios Spring Data JPA.
    - com.gedifms.domain: Entidades JPA e DTOs/Records (ex: Certificate, Student, ActivityClassification, CertificateData, ValidationResult).
    - com.gedifms.config: Configurações do Spring (ex: setup de clientes AI, Segurança).
  - **Tecnologias:** Spring Boot 3.x (requer Java 17+), Spring Data JPA, Spring Web, Spring Security.[30] Utilizar Maven como gerenciador de dependências, conforme solicitado.
- **B. Projeto do Banco de Dados (PostgreSQL + pgVector):**
  - **Justificativa:** PostgreSQL é um banco de dados relacional robusto e amplamente utilizado. A extensão pgVector adiciona capacidades eficientes de busca por similaridade vetorial, o que é útil caso técnicas como RAG ou busca semântica sejam implementadas futuramente.[58] O professor mencionou Postgres/H2; recomenda-se usar PostgreSQL para persistência devido à capacidade do pgVector, e H2 para testes unitários/integração.
  - **Entidades Principais (JPA):**
    - Student: Informações básicas do aluno.

    - Certificate: Armazena metadados sobre o documento enviado (nome do arquivo, data de upload, associação ao aluno, status do processamento: PENDING_VALIDATION, VALIDATED, REJECTED), e uma referência ao local de armazenamento do documento real (ex: caminho no sistema de arquivos ou ID em um blob store).
    - ExtractedData: Armazena os dados extraídos pela IA (e potencialmente submetidos pelo aluno: título, horas, datas, etc.), vinculados a um Certificate.
    - ActivityClassification: Representa as categorias predefinidas (UC, PE, etc.) com suas descrições e, crucialmente, os limites máximos de horas permitidos para cada uma.
    - CertificateClassification: Tabela de ligação que associa um Certificate a uma ou mais ActivityClassifications. Deve armazenar as horas *efetivamente atribuídas* para aquela categoria a partir daquele certificado e o status de validação dessa atribuição (ex: AI_VALIDATED, PROFESSOR_APPROVED, REJECTED). Pode incluir uma coluna para o embedding vetorial (vector) se pgVector/RAG for utilizado.
    - HourLog: Tabela para rastrear o total de horas validadas por categoria para cada aluno, facilitando a consulta do progresso do aluno.
- **Integração com pgVector:**
    - Adicionar a dependência spring-ai-pgvector-store-spring-boot-starter ao pom.xml.[58]
    - Configurar o arquivo application.properties (ou application.yml) com os detalhes de conexão do PostgreSQL e as configurações específicas do pgVector (ex: spring.ai.vectorstore.pgvector.enabled=true, dimension do vetor, index-type como HNSW ou IVFFlat, distance-type como COSINE ou L2).[58] O Spring AI pode inicializar automaticamente o esquema da tabela vector_store se configurado.[58]
    - Um bean VectorStore será provavelmente auto-configurado se o starter for

usado.[63]

- Se RAG for implementado, os embeddings (ex: do texto do certificado) seriam armazenados em uma coluna do tipo vector (por exemplo, na tabela CertificateClassification ou em uma tabela dedicada).[62]
- *Uso Estratégico do pgVector:* Embora o pgVector seja poderoso para RAG e busca por similaridade [62], sua necessidade imediata depende da estratégia de classificação adotada. Se o prompting direto ao LLM for suficiente, a configuração completa do pgVector pode ser adiada, simplificando a configuração inicial. A configuração do pgVector envolve a instalação da extensão no banco (CREATE EXTENSION vector) [62], a definição de tabelas com colunas vetoriais de dimensões específicas [61], e a criação de índices apropriados (HNSW, IVFFlat).[61] Isso adiciona uma sobrecarga na configuração inicial. Recomenda-se incluir a dependência do pgVector no projeto, mas talvez desabilitar ou comentar a auto-configuração do VectorStore inicialmente. Focar primeiro na extração e validação central. Reabilitar e configurar completamente o pgVector se e quando o RAG se tornar necessário.

- **C. Estratégia de Integração de Ferramentas de IA:**
  - **Framework Principal: Spring AI:** Utilizar suas abstrações como ChatClient, EmbeddingClient (se necessário para pgVector/RAG), VectorStore, e DocumentReader.[14] Isso garante portabilidade entre diferentes modelos de IA (OpenAI, Ollama, etc.).[14]
  - **Acesso a LLMs:**
    - **Ollama (Local):** Usar o starter spring-ai-ollama-spring-boot-starter [58] para desenvolvimento e testes locais com modelos como Llama3 ou Mixtral [User Query]. Configurar a URL base no application.properties. É uma ótima opção para experimentação sem custo.
    - **OpenAI (Nuvem):** Usar o starter spring-ai-openai-spring-boot-starter.[30] Requer a configuração da chave de API.[30] Oferece potencialmente maior

confiabilidade, especialmente para saída estruturada.[40]

- **Integração OCR:** Utilizar a biblioteca Tess4J diretamente dentro de um serviço Spring, pois o Spring AI atualmente não possui leitores OCR integrados (embora discussões sobre integrações futuras existam [31]). Gerenciar a instalação do Tesseract e o caminho para tessdata.
- **Vector Store:** Usar o starter spring-ai-pgvector-store-spring-boot-starter para integração com pgVector.[58]
- **Alternativa: Langchain4j:** Poderia ser usado como alternativa ou complemento.[12] Oferece abstrações similares para LLMs, carregamento de documentos e vector stores. A integração com Spring existe, mas pode ser menos fluida que o próprio Spring AI.[72] Recomenda-se manter o foco no Spring AI para uma experiência mais nativa do Spring, a menos que Langchain4j ofereça uma funcionalidade específica e crítica não disponível facilmente no Spring AI.

- **D. Gerenciamento de Configuração:**
  - Utilizar perfis Spring (ex: dev, prod) para gerenciar diferentes configurações (URL base do Ollama vs chave OpenAI, strings de conexão H2 vs PostgreSQL).
  - Externalizar configurações sensíveis (chaves de API) e variáveis (nomes de modelos, caminhos) usando application.properties ou variáveis de ambiente.

V. Implementação da Lógica de Negócios Central

- **A. Serviço de Cálculo de Horas:**
  - Entrada: Dados validados de CertificateClassification (categoria atribuída, horas extraídas do certificado).
  - Lógica:
    1. Recuperar o limite máximo de horas permitido para a categoria específica a partir da entidade ActivityClassification (ActivityClassification.maxHours).
    2. Calcular as horas aplicáveis para esta categoria neste certificado:

min(horas_extraidas, limite_max_categoria).
3. Calcular as horas remanescentes (excedentes): horas_extraidas - horas_aplicaveis.
4. Registrar as horas_aplicaveis na entrada correspondente de CertificateClassification.
5. Atualizar o total de horas validadas do aluno para essa categoria na tabela HourLog.

- **B. Lógica de Redistribuição de Horas:**
  - Gatilho: Executado quando horas_remanescentes > 0 após o cálculo inicial.
  - Entrada: O Certificate em questão, o valor de horas_remanescentes, e a lista de *todas* as categorias às quais o certificado foi classificado (pode ser mais de uma, conforme exemplos do usuário).
  - Lógica:
    1. Identificar outras categorias válidas para o *mesmo certificado* nas quais o aluno ainda precisa de horas E cujo limite máximo de horas (para *esta contribuição* do certificado) ainda não foi atingido.
    2. Determinar as regras de redistribuição permitidas. O exemplo do usuário (PA -> PV) sugere que pode haver regras específicas ou mapeamentos definindo quais categorias podem receber o excedente de outras. **Esta lógica precisa ser formalmente definida pelas regras do IFMS.**
    3. Distribuir iterativamente as horas_remanescentes para as categorias de destino elegíveis, respeitando seus limites máximos individuais *para a contribuição deste certificado*.
    4. Atualizar os registros CertificateClassification das categorias de destino com as horas redistribuídas.
    5. Atualizar a tabela HourLog do aluno de acordo.
  - *Necessidade de Clarificação das Regras:* A lógica de redistribuição é complexa e depende criticamente de regras institucionais que não foram completamente especificadas na consulta inicial. Os exemplos fornecidos levantam questões

chave:

- Um único certificado pode contribuir para múltiplas categorias simultaneamente? (Os exemplos sugerem que sim).
- Como as horas excedentes são tratadas? Podem preencher *qualquer* outra categoria correspondente do certificado, ou apenas algumas específicas (ex: PE só pode redistribuir para PV)?
- Os limites de categoria se aplicam por certificado ou ao total do aluno? (Os exemplos sugerem que o limite inicial é por certificado, mas a redistribuição precisa de clareza sobre como interage com o progresso total do aluno). Esta lógica de negócios é fundamental e requer uma especificação formal e precisa por parte do IFMS, provavelmente envolvendo o professor orientador ou a administração acadêmica. A implementação só pode prosseguir após essa definição.

- **C. Fluxo de Dados e Gerenciamento de Status:**
  - É essencial rastrear o status de cada certificado ao longo do processo: UPLOADED, OCR_DONE, EXTRACTION_DONE, PENDING_VALIDATION, AI_VALIDATED, VALIDATION_FAILED, PENDING_MANUAL_REVIEW, PROFESSOR_APPROVED, REJECTED, COMPLETED.
  - Garantir a atomicidade e a consistência das operações durante as etapas de processamento, possivelmente usando transações de banco de dados onde apropriado.

VI. Melhorias na Validação de Certificados

- **A. Validação por QR Code:**
  - Esta funcionalidade foi sugerida pelo professor [Audio 06, Audio 10] como uma forma de aumentar a confiabilidade. Muitos certificados modernos incluem QR codes que levam a URLs de validação ou contêm dados assinados.
  - **Biblioteca:** ZXing (“Zebra Crossing”) é a biblioteca Java padrão para processamento de QR code.[73] As dependências Maven com.google.zxing:core

e com.google.zxing:javase são necessárias.[73]

- **Implementação:**
  1. Durante o processamento do documento (após OCR/extração de texto, ou diretamente do PDF original se possível), tentar detectar a presença de QR codes. Isso pode envolver bibliotecas de processamento de imagem (como OpenCV, mencionado tangencialmente [18]) ou funcionalidades de detecção dentro do próprio ZXing.[75]
  2. Usar QRCodeReader ou MultiFormatReader do ZXing para decodificar o conteúdo do QR code a partir da área detectada ou da imagem/página inteira.[73] Isso requer a conversão do segmento da imagem/página PDF para um BinaryBitmap usando uma LuminanceSource apropriada.[73]
  3. Analisar o conteúdo decodificado. Frequentemente, será uma URL.
  4. Se for uma URL, realizar uma requisição HTTP (usando o WebClient do Spring) para o endpoint de validação.
  5. Analisar a resposta recebida da URL de validação para confirmar a autenticidade do certificado.
  6. Integrar o resultado da validação do QR code ao status geral de validação do certificado no sistema.
- *Complexidade vs. Valor:* A validação por QR code adiciona uma camada significativa de robustez, mas também de complexidade. Detectar QR codes de forma confiável em digitalizações potencialmente tortas ou de baixa qualidade pode ser desafiador. Lidar com as diversas respostas dos endpoints de validação e possíveis erros de rede adiciona sobrecarga. Considerando as restrições do TCC e a orientação do professor para focar na implementação essencial [Audio 08, Audio 10], esta funcionalidade pode ser classificada como "desejável" em vez de "essencial" para a entrega inicial.

- **B. Tratamento de Falhas na Validação:**
  - Se a validação por IA falhar (detectar discrepâncias entre a entrada do aluno e o certificado) ou a validação por QR code (se implementada) falhar, o

certificado deve ser automaticamente sinalizado para revisão manual por um Professor/Validador.

- A interface do validador deve apresentar claramente a imagem do certificado, o texto extraído, os dados submetidos pelo aluno, a explicação da falha fornecida pela IA e o status da validação do QR code.

VII. Fluxo de Trabalho Proposto

- **A. Definição do Fluxo (conforme solicitado pelo professor [Audio 06, Audio 07]):**
  1. **Aluno:** Realiza login no sistema e acessa a seção de envio de certificados.
  2. **Aluno:** Faz upload do arquivo do certificado (PDF/Imagem). Fornece dados iniciais, como a classificação sugerida e a carga horária (conforme [Audio 05]).
  3. **Sistema (Backend):** Recebe o upload, armazena o arquivo (sistema de arquivos ou blob storage), cria um registro Certificate no banco de dados com status inicial UPLOADED. Dispara uma tarefa de processamento (potencialmente assíncrona para não bloquear a requisição do usuário).
  4. **Sistema (Backend - Tarefa de Processamento):**
     - Detecta o tipo de arquivo. Se for imagem ou PDF digitalizado, executa OCR (Tesseract/Tess4J) para obter o Texto Extraído. Se for PDF textual, extrai o texto diretamente (Spring AI Reader). Atualiza o status para OCR_DONE (se aplicável).
     - Executa a Extração de Dados Estruturados usando LLM (Spring AI ChatClient.entity()) para preencher um registro ExtractedData. Atualiza o status para EXTRACTION_DONE.
     - Executa a Validação por IA usando LLM (Spring AI ChatClient.entity()) comparando ExtractedData + Entrada do Aluno com o texto do certificado, gerando um ValidationResult.
     - (Opcional) Executa a Detecção/Decodificação/Validação de QR Code (ZXing, HTTP Client), registrando o Status do QR.

- Se a Validação por IA (e a Validação QR, se realizada) forem bem-sucedidas: Atualiza o status para AI_VALIDATED. Prossegue para o cálculo de horas.
- Se a Validação por IA ou a Validação QR falharem: Atualiza o status para PENDING_MANUAL_REVIEW. Notifica um validador (professor/administrador).

5. **Sistema (Backend - Cálculo de Horas):** (Executado se o status for AI_VALIDATED)
   - Aplica os limites de horas com base na categoria (ActivityClassification).
   - Calcula as horas aplicáveis e remanescentes.
   - Executa a lógica de redistribuição de horas, se necessário (e se as regras estiverem definidas).
   - Atualiza os registros CertificateClassification e HourLog.
   - Se a política institucional não exigir aprovação manual após validação por IA: Atualiza o status para COMPLETED. Notifica o aluno.
   - Se a aprovação manual *for* necessária mesmo após validação por IA: Atualiza o status para PENDING_FINAL_APPROVAL. Notifica um validador.
6. **Professor/Validador:** Realiza login, visualiza a fila de certificados que requerem revisão (status PENDING_MANUAL_REVIEW ou PENDING_FINAL_APPROVAL).
7. **Professor/Validador:** Analisa o certificado, os dados extraídos/submetidos, a análise da IA. Aprova ou Rejeita a solicitação. Tem a capacidade de corrigir/sobrescrever a classificação ou as horas atribuídas pela IA, se necessário.
8. **Sistema (Backend):** Atualiza o status final do certificado (COMPLETED ou REJECTED). Atualiza as horas no HourLog se houver correções manuais. Notifica o aluno sobre o resultado final.

- **B. Diagrama Potencial:**

- Recomenda-se incluir um diagrama visual (BPMN ou Diagrama de Atividade UML) na documentação final do TCC (Seção VII deste relatório) para ilustrar claramente este fluxo de trabalho. O diagrama deve usar raias (swimlanes) para distinguir as ações realizadas pelo Aluno, Sistema (Backend) e Professor/Validador, e mostrar claramente os pontos de decisão (Validação OK/Falha, Revisão Manual Necessária?).

VIII. Recomendações e Próximos Passos

- **A. Lista de Tarefas Priorizadas (Backend/IA):**
  1. Configurar a estrutura do projeto Spring Boot, adicionar dependências essenciais (Spring AI para Ollama/OpenAI, Spring Web, Spring Data JPA, driver PostgreSQL, Tess4J, ZXing).
  2. Implementar o endpoint básico para upload de arquivos.
  3. Implementar a extração de texto de PDFs textuais usando DocumentReader do Spring AI.
  4. Implementar o pipeline de OCR para imagens/PDFs digitalizados (integração Tesseract/Tess4J, conversão de página PDF para imagem com PDFBox).
  5. Implementar o serviço de Extração de Dados Estruturados usando ChatClient.entity() do Spring AI. Definir o record CertificateData. Criar o prompt de extração.
  6. Implementar o serviço de Validação por IA usando ChatClient.entity() do Spring AI. Definir o record ValidationResult. Criar o prompt de validação que compara a entrada do aluno com o texto extraído.
  7. Definir o Esquema do Banco de Dados (Entidades JPA) e criar os Repositórios Spring Data JPA.
  8. Implementar a lógica de negócios central para o cálculo de horas (aplicação de limites).
  9. Implementar a lógica de redistribuição de horas (requer clarificação prévia das regras institucionais).

10. Implementar o gerenciamento dos estados do fluxo de trabalho e as transições entre eles.
11. (Prioridade Menor) Implementar o serviço de validação de QR Code usando ZXing.
12. (Prioridade Menor / Opcional) Configurar pgVector e implementar RAG se a classificação/validação inicial via LLM direto se mostrar insuficiente.

- **B. Resumo das Recomendações de Ferramentas:**
  - **Framework Principal:** Spring Boot, Spring AI.
  - **Banco de Dados:** PostgreSQL (com extensão pgVector habilitada, mesmo que não usada inicialmente). H2 para testes.
  - **OCR:** Tesseract + biblioteca Tess4J.
  - **Interação com LLM:** ChatClient do Spring AI. Iniciar com Ollama (Llama3/Mixtral) localmente, considerar API OpenAI se necessário/permitido.
  - **QR Codes:** Biblioteca ZXing.
  - **Evitar Inicialmente:** Weka/Orange (usar LLMs/Tesseract diretamente), pipelines complexos de treinamento de ML (Scikit-learn, Tensorflow, Pytorch, a menos que RAG/Embeddings os exijam), MLFlow (excessivo para este TCC). Langchain(4j) pode ser mantido como alternativa se o Spring AI apresentar limitações para uma tarefa específica.
- **C. Testes e Avaliação:**
  - Testes unitários para os serviços (especialmente cálculo de horas e redistribuição).
  - Testes de integração para os endpoints da API e interações com o banco de dados.
  - Testes manuais com uma variedade de exemplos de certificados reais (PDF textual, PDF digitalizado, diferentes layouts, diferentes tipos de atividades) para avaliar a robustez do pipeline.
  - Avaliar a precisão da extração e validação da IA qualitativamente – o sistema funciona razoavelmente bem para a maioria dos casos? A perfeição não é o

objetivo do TCC [Audio 08, Audio 10]. O foco deve ser demonstrar o funcionamento do pipeline implementado.

- **D. Colaboração com o Frontend (Mateus):**
  - Definir contratos de API claros (DTOs de requisição/resposta) para os endpoints que serão consumidos pelo frontend Angular (ex: upload, consulta de status, visualização de horas validadas).
  - Utilizar uma ferramenta como Swagger/OpenAPI para documentar a API REST.
- **E. Conselho Final:**
  - Concentrar esforços na entrega de um fluxo funcional de ponta a ponta para o cenário principal de validação, conforme orientado pelo professor [Audio 05, Audio 06, Audio 07].
  - Priorizar a implementação prática e funcional sobre a perfeição teórica ou a exploração de técnicas excessivamente complexas [Audio 08, Audio 10, Audio 11].
  - Documentar claramente as escolhas de design e os passos de implementação no relatório do TCC (mencionando as ferramentas chave utilizadas, como Spring AI, Tesseract, etc.). A pasta pesquisa (latex) planejada inicialmente é apropriada para esta documentação.

Este relatório fornece uma base sólida e um roteiro prático para o desenvolvimento do backend e da integração de IA para o sistema GED IFMS, alinhado com os requisitos do TCC e as orientações fornecidas. A execução focada nas tarefas priorizadas e a busca por clareza nas regras de negócio institucionais serão cruciais para o sucesso do projeto dentro do prazo estabelecido.

Referências citadas

1. Java AI: Extracting Structured Data from Images with Spring AI - Vaadin, acessado em maio 2, 2025, https://vaadin.com/blog/java-ai-image-data-extraction-spring-ai
2. OCR-Tesseract in Spring Boot 2024 - LearnerBits, acessado em maio 2, 2025,

https://learnerbits.com/ocr-tesseract-in-spring-boot-2024/
3. ParagraphPdfDocumentReader (Spring AI 0.8.1 API), acessado em maio 2, 2025, https://docs.spring.io/spring-ai/docs/current/api/org/springframework/ai/reader/pdf/ParagraphPdfDocumentReader.html
4. ETL Pipeline :: Spring AI Reference, acessado em maio 2, 2025, https://docs.spring.io/spring-ai/reference/1.0/api/etl-pipeline.html
5. How to Implement RAG Pipeline Using Spring AI - Talentica Software, acessado em maio 2, 2025, https://www.talentica.com/blogs/rag-pipeline-using-spring-ai/
6. ETL Pipeline :: Spring AI Reference, acessado em maio 2, 2025, https://docs.spring.io/spring-ai/reference/api/etl-pipeline.html
7. DocumentReader (Spring AI 0.8.1 API), acessado em maio 2, 2025, https://docs.spring.io/spring-ai/docs/current/api/org/springframework/ai/document/DocumentReader.html
8. RAG made easy with Spring AI + Elasticsearch, acessado em maio 2, 2025, https://www.elastic.co/search-labs/blog/java-rag-spring-ai-es
9. [Question] How to retrieve metadata of document using spring AI? #4173 #678 - GitHub, acessado em maio 2, 2025, https://github.com/spring-projects/spring-ai/discussions/678
10. Vector Databases - Spring AI, acessado em maio 2, 2025, https://www.spring-doc.cn/spring-ai/1.0.0-SNAPSHOT/api_vectordbs.en.html
11. langchain4j/langchain4j/src/main/java/dev/langchain4j/data/document/loader/FileSystemDocumentLoader.java at main · langchain4j/langchain4j - GitHub, acessado em maio 2, 2025, https://github.com/langchain4j/langchain4j/blob/main/langchain4j/src/main/java/dev/langchain4j/data/document/loader/FileSystemDocumentLoader.java
12. RAG using Langchain4j and Astra Vector Db for reading PDF file and generating AI response - shdhumale, acessado em maio 2, 2025, https://shdhumale.wordpress.com/2024/02/20/rag-using-langchain4j-and-aster-vector-db-for-reading-pdf-file-and-generating-ai-response/

13. Document Loaders - LangChain4j, acessado em maio 2, 2025, https://docs.langchain4j.dev/category/document-loaders/
14. Spring AI, acessado em maio 2, 2025, https://spring.io/projects/spring-ai
15. spring-projects/spring-ai: An Application Framework for AI Engineering - GitHub, acessado em maio 2, 2025, https://github.com/spring-projects/spring-ai
16. Using LangChain4j to analyze PDF documents - Quarkus, acessado em maio 2, 2025, https://quarkus.io/blog/using-langchain4j-to-analyze-pdf-documents/
17. langchain4j/docs/docs/tutorials/rag.md at main - GitHub, acessado em maio 2, 2025, https://github.com/langchain4j/langchain4j/blob/main/docs/docs/tutorials/rag.md
18. Tesseract OCR with Java Examples - Tutorialspoint, acessado em maio 2, 2025, https://www.tutorialspoint.com/tesseract-ocr-with-java-with-examples
19. Tesseract OCR with Java with Examples - GeeksforGeeks, acessado em maio 2, 2025, https://www.geeksforgeeks.org/tesseract-ocr-with-java-with-examples/
20. Optical Character Recognition with Tesseract | Baeldung, acessado em maio 2, 2025, https://www.baeldung.com/java-ocr-tesseract
21. Best tool for OCR and AI in combination : r/ChatGPTPro - Reddit, acessado em maio 2, 2025, https://www.reddit.com/r/ChatGPTPro/comments/1e6e98g/best_tool_for_ocr_and_ai_in_combination/
22. How to Perform OCR Using Tesseract OCR API in Java - Omi, acessado em maio 2, 2025, https://www.omi.me/blogs/api-guides/how-to-perform-ocr-using-tesseract-ocr-api-in-java
23. Tesseract OCR for Non-English Languages - PyImageSearch, acessado em maio 2, 2025, https://pyimagesearch.com/2020/08/03/tesseract-ocr-for-non-english-languages/
24. fatihyildizli/springboot-tesseract-ocr: Tesseract OCR Engine POC project in spring boot, acessado em maio 2, 2025,

https://github.com/fatihyildizli/springboot-tesseract-ocr

25. Muhammederendemir/spring-boot-tesseract-ocr - GitHub, acessado em maio 2, 2025, https://github.com/Muhammederendemir/spring-boot-tesseract-ocr
26. darkn3to/pdfocr: A simple Spring Boot application to convert image-based PDFs to text-embedded PDFs. - GitHub, acessado em maio 2, 2025, https://github.com/darkn3to/pdfocr
27. Intelligent Document Reader - Salesforce Help, acessado em maio 2, 2025, https://help.salesforce.com/s/articleView?id=ind.intelligent_document_reader.htm&language=en_US&type=5
28. Intelligent Document Reader | Industries Common Resources Developer Guide, acessado em maio 2, 2025, https://developer.salesforce.com/docs/atlas.en-us.industries_reference.meta/industries_reference/intelligent_document_reader.htm
29. Multimodality API :: Spring AI Reference, acessado em maio 2, 2025, https://docs.spring.io/spring-ai/reference/api/multimodality.html
30. Extracting Structured Data From Images Using Spring AI | Baeldung, acessado em maio 2, 2025, https://www.baeldung.com/spring-ai-extract-data-from-images
31. Add support for Mistral OCR · Issue #2582 · spring-projects/spring-ai - GitHub, acessado em maio 2, 2025, https://github.com/spring-projects/spring-ai/issues/2582
32. getomni-ai/zerox: OCR & Document Extraction using vision models - GitHub, acessado em maio 2, 2025, https://github.com/getomni-ai/zerox
33. OCRmyPDF adds an OCR text layer to scanned PDF files, allowing them to be searched - GitHub, acessado em maio 2, 2025, https://github.com/ocrmypdf/OCRmyPDF
34. aborroy/alf-tengine-ocr: Alfresco Transformer For ACS 70+ from PDF to OCRd PDF - GitHub, acessado em maio 2, 2025, https://github.com/aborroy/alf-tengine-ocr
35. how to increase processing speed of tesseract OCR? · Issue #160 ·

nguyenq/tess4j - GitHub, acessado em maio 2, 2025, https://github.com/nguyenq/tess4j/issues/160

36. Entity Recognition In Spring Applications | Restackio, acessado em maio 2, 2025, https://www.restack.io/p/entity-recognition-knowledge-answer-spring-applications-cat-ai
37. Structured Output - Spring AI, acessado em maio 2, 2025, https://spring.io/blog/2024/05/09/spring-ai-structured-output/
38. A Guide to Structured Output in Spring AI - Baeldung, acessado em maio 2, 2025, https://www.baeldung.com/spring-artificial-intelligence-structure-output
39. Chat Client API :: Spring AI Reference, acessado em maio 2, 2025, https://docs.spring.io/spring-ai/reference/api/chatclient.html
40. How to Use Structured Outputs with Spring AI - SFG, acessado em maio 2, 2025, https://springframework.guru/using-structured-outputs-with-spring-ai/
41. Spring AI Embraces OpenAI's Structured Outputs: Enhancing JSON Response Reliability, acessado em maio 2, 2025, https://spring.io/blog/2024/08/09/spring-ai-embraces-openais-structured-outputs-enhancing-json-response/
42. Spring AI Structured Output Example - Java Code Geeks, acessado em maio 2, 2025, https://www.javacodegeeks.com/spring-ai-structured-output-example.html
43. Using Spring AI Structured Output: List, Map, and Bean Converters - Java Code Geeks, acessado em maio 2, 2025, https://www.javacodegeeks.com/using-spring-ai-structured-output-list-map-and-bean-converters.html
44. How to use Spring to extract complex structural data #1122 - GitHub, acessado em maio 2, 2025, https://github.com/spring-projects/spring-ai/discussions/1122
45. Spring AI: A Door to GenAI Heaven for Java Developers - Sabre, acessado em maio 2, 2025, https://www.sabre.com/locations/india/wp/wp-content/uploads/83-85_Spring-AI_OSFY-Feb.-2025.pdf

46. Function Calling API :: Spring AI Reference, acessado em maio 2, 2025, https://docs.spring.io/spring-ai/reference/api/functions.html
47. Getting Started with Spring AI Function Calling - Piotr's TechBlog, acessado em maio 2, 2025, https://piotrminkowski.com/2025/01/30/getting-started-with-spring-ai-function-calling/
48. AI Concepts :: Spring AI Reference, acessado em maio 2, 2025, https://docs.spring.io/spring-ai/reference/concepts.html
49. Extract entities with the `ai.extract` function - Microsoft Fabric, acessado em maio 2, 2025, https://learn.microsoft.com/en-us/fabric/data-science/ai-functions/extract
50. Extract Entities Using Azure OpenAI Structured Outputs Mode | Microsoft Learn, acessado em maio 2, 2025, https://learn.microsoft.com/en-us/azure/developer/ai/how-to/extract-entities-using-structured-outputs
51. Spring AI and Spring Cloud Compatibility Issue: JSON Parsing Error in OpenAI Chat Client, acessado em maio 2, 2025, https://stackoverflow.com/questions/79547015/spring-ai-and-spring-cloud-compatibility-issue-json-parsing-error-in-openai-cha
52. Text Classification with Java, Spring AI, LLMs and Ollama - Thomas Vitale, acessado em maio 2, 2025, https://www.thomasvitale.com/text-classification-with-spring-ai/
53. Text Classification: How Machine Learning Is Revolutionizing Text Categorization - MDPI, acessado em maio 2, 2025, https://www.mdpi.com/2078-2489/16/2/130
54. Comparison of Different Machine Learning Approaches to Text Classification, acessado em maio 2, 2025, https://www.researchgate.net/publication/363054421_Comparison_of_Different_Machine_Learning_Approaches_to_Text_Classification
55. Machine Learning-Based Text Classification Comparison | Encyclopedia MDPI, acessado em maio 2, 2025, https://encyclopedia.pub/entry/48715

56. Text Classification: Neural Networks VS Machine Learning Models VS Pre-trained Models - arXiv, acessado em maio 2, 2025, https://arxiv.org/html/2412.21022v1
57. Comparative Analysis of Machine and Deep Learning Techniques for Text Classification with Emphasis on Data Preprocessing - Qeios, acessado em maio 2, 2025, https://www.qeios.com/read/XHC9J1
58. Building a Semantic Search System with Spring AI and PGVector - Java Code Geeks, acessado em maio 2, 2025, https://www.javacodegeeks.com/building-a-semantic-search-system-with-spring-ai-and-pgvector.html
59. Prompt Engineering Techniques with Spring AI, acessado em maio 2, 2025, https://spring.io/blog/2025/04/14/spring-ai-prompt-engineering-patterns/
60. AI LLM Test Prompts: Best Practices for AI Evaluation and Optimization, acessado em maio 2, 2025, https://www.patronus.ai/llm-testing/ai-llm-test-prompts
61. Implementing Semantic Search Using Spring AI and PGVector - Baeldung, acessado em maio 2, 2025, https://www.baeldung.com/spring-ai-pgvector-semantic-search
62. Getting Started With Spring AI and PostgreSQL PGVector - DZone, acessado em maio 2, 2025, https://dzone.com/articles/spring-ai-with-postgresql-pgvector
63. PGvector :: Spring AI Reference - GitHub Pages, acessado em maio 2, 2025, https://markpollack.github.io/spring-ai-0.7.1/api/vectordbs/pgvector.html
64. PGvector :: Spring AI Reference, acessado em maio 2, 2025, https://docs.spring.io/spring-ai/reference/api/vectordbs/pgvector.html
65. Integrating Spring AI with Vector Databases - A Guide Using PGVector, acessado em maio 2, 2025, https://docs.rapidapp.io/blog/integrating-spring-ai-with-vector-databases
66. Spring AI, Kafka, RAG and microservices, acessado em maio 2, 2025, https://rarcos.com/2025/02/12/GMCCA_microservices/
67. Spring AI - openAi Error extracting response of type OpenAiApi$EmbeddingList<.....OpenAiApi$Embedding> - Stack Overflow,

acessado em maio 2, 2025, https://stackoverflow.com/questions/78121525/spring-ai-openai-error-extracting-response-of-type-openaiapiembeddinglist

68. Advanced RAG techniques with Spring AI - Vaadin, acessado em maio 2, 2025, https://vaadin.com/blog/advanced-rag-techniques-with-spring-ai
69. Spring AI Concepts Tutorial With Examples - Chat Model API - JavaTechOnline, acessado em maio 2, 2025, https://javatechonline.com/spring-ai-concepts-tutorial-with-examples/
70. How to build an AI bot with Spring AI and Unstructured.io - BellSoft, acessado em maio 2, 2025, https://bell-sw.com/blog/creating-a-bot-for-documentation-assistance-with-spring-ai-and-unstructured-io/
71. lucasnscr/SpringAI - GitHub, acessado em maio 2, 2025, https://github.com/lucasnscr/SpringAI
72. acessado em dezembro 31, 1969, https://www.baeldung.com/java-langchain-mongodb?ref=dailydev
73. How to generate and read QR code with Java using ZXing Library - GeeksforGeeks, acessado em maio 2, 2025, https://www.geeksforgeeks.org/how-to-generate-and-read-qr-code-with-java-using-zxing-library/
74. ZXing ("Zebra Crossing") barcode scanning library for Java, Android - GitHub, acessado em maio 2, 2025, https://github.com/zxing/zxing
75. Find QR code in image and decode it using Zxing - Stack Overflow, acessado em maio 2, 2025, https://stackoverflow.com/questions/36210537/find-qr-code-in-image-and-decode-it-using-zxing
76. Reading QRCode with Zxing in Java - Stack Overflow, acessado em maio 2, 2025, https://stackoverflow.com/questions/18863466/reading-qrcode-with-zxing-in-java