Relatório do Trabalho Prático de PLN

Aplicação Web de Chat com Django, Transformers e MongoDB

Equipe:

- Bruno Santos (Função: Ex: Frontend e UI/UX)
- Luccas Lohan (Função: Ex: Backend e Banco de Dados)
- Artur Revollo (Função: Ex: Lógica de IA e Integração)

Data: 01 de Novembro de 2025

1. Introdução

Este relatório detalha a arquitetura, as decisões técnicas e os resultados do desenvolvimento da aplicação "IDE de IA", como parte da avaliação da disciplina de Processamento de Linguagem Natural. O objetivo principal do projeto foi construir uma aplicação web completa que permite a interação com um modelo de linguagem natural (LLM) da Hugging Face, com persistência de dados em MongoDB.

O projeto foi desenvolvido utilizando o framework Django para o backend, pymongo para a interação com a base de dados, e a biblioteca transformers para o carregamento e inferência local do modelo de IA, cumprindo todos os requisitos obrigatórios solicitados.

A aplicação final permite que um usuário converse com um modelo de IA, veja seu histórico de conversas, filtre esse histórico por texto ou data, e exporte os dados em formatos CSV ou JSON.

2. Arquitetura da Solução

A aplicação segue uma arquitetura de projeto Django padrão, separando claramente as responsabilidades (Separation of Concerns) para garantir a manutenibilidade e escalabilidade.

- **Project (project/)**: Contém as configurações globais do Django (settings.py), roteamento principal (urls.py) e o ponto de entrada (wsgi.py). As configurações sensíveis (chaves secretas, conexão do DB) são carregadas de um arquivo .env através da biblioteca python-dotenv.
- **App Core (core/)**: Responsável por servir as páginas estruturais, como o template base (base.html) e a página inicial do chat (index.html), que é a raiz do site.
- **App Chat (chat/)**: É o cérebro da aplicação. Contém:
    - views.py: Controladores que recebem os pedidos HTTP (para gerar respostas, ver histórico, exportar) e retornam as respostas (HTML ou JSON).
    - urls.py: Define as rotas específicas da aplicação de chat (ex: /chat/gerar/,

/chat/historico/).
    - services/: Uma pasta dedicada para a lógica de negócio, desacoplada das *views*:
        - nlp_service.py: Responsável por carregar o modelo de IA (Qwen/Qwen2-0.5B-Instruct) da Hugging Face na memória (como um *singleton*) e por gerar as respostas.
        - mongo_service.py: Abstrai toda a comunicação com o MongoDB, fornecendo funções como create_chat, add_message, get_all_chats_paginated, get_all_chats_for_export, etc.
- **Templates**: O sistema utiliza a herança de templates do Django (base.html) para manter uma interface de usuário consistente entre a página de Chat (index.html) e a página de Histórico (historico.html).
- **Static (static/)**: Pasta na raiz do projeto que armazena os arquivos CSS e JavaScript, servidos pelo Django.

3. Decisões Técnicas

Durante o desenvolvimento, várias decisões importantes foram tomadas para cumprir os requisitos dentro das limitações de um ambiente de desenvolvimento local.

- **Framework (Django)**:
    - **Decisão:** Utilizar Django, conforme solicitado pelo professor. O projeto foi migrado de uma prova de conceito inicial feita em FastAPI.
    - **Justificativa:** O Django, apesar de mais robusto que o FastAPI, facilitou enormemente a gestão de templates (com base.html), arquivos estáticos, paginação (com Paginator) e a organização geral do projeto com a sua estrutura de "apps" (chat, core), que se provou ideal para separar as funcionalidades.
- **Modelo de IA (Qwen/Qwen2-0.5B-Instruct)**:
    - **Decisão:** O requisito era usar um modelo transformers. Após tentativas com modelos maiores (como o microsoft/Phi-3-mini-4k-instruct) que causaram erros críticos de memória (Segmentation fault) em máquinas com 8GB de RAM, optamos pelo Qwen2-0.5B-Instruct (0.5 Bilhão de parâmetros).
    - **Justificativa:** Este modelo provou ser o equilíbrio perfeito: é um modelo de chat *instruct* (feito para perguntas e respostas), é extremamente leve (cerca de 1GB de download), responde bem em português para tarefas simples e, o mais importante, **roda de forma estável em CPU** (device_map="cpu") sem esgotar os recursos do sistema.
- **Banco de Dados (PyMongo Direto vs. ORM)**:
    - **Decisão:** O requisito era usar MongoDB. Em vez de usar bibliotecas de abstração como djongo (que mapeia o ORM do Django para o MongoDB), optamos por usar pymongo diretamente, isolado num módulo de serviço (mongo_service.py).
    - **Justificativa:** Esta abordagem nos deu controle total sobre as *queries* NoSQL. Foi crucial para implementar os filtros de busca de forma eficiente, usando a sintaxe nativa do MongoDB ($regex para busca de texto *case-insensitive* e $gte/$lte para filtros de data complexos), o que seria muito mais complexo de fazer através de um

ORM tradicional.
- **Testes (mongomock)**:
  - **Decisão:** Para cumprir o requisito de testes, foi necessário simular a base de dados.
  - **Justificativa:** Para isolar os testes da base de dados real, utilizamos a biblioteca mongomock. Ela simula a API do pymongo em memória, permitindo testes unitários rápidos, isolados e confiáveis (como visto em chat/tests.py), sem a necessidade de um servidor MongoDB rodando during a execução dos testes.

4. Limitações Conhecidas

Apesar de funcional, o projeto tem limitações inerentes às escolhas técnicas feitas:

- **Qualidade das Respostas**: O Qwen2-0.5B-Instruct é um modelo pequeno. Embora rápido, sua base de conhecimento é limitada e a qualidade de suas respostas não se compara a modelos de ponta (como GPT-4 ou Llama 3).
- **Ausência de Streaming**: A resposta da IA é enviada ao frontend apenas quando está 100% completa. A implementação de *streaming* (listado como requisito bônus) melhoraria muito a percepção de velocidade do usuário, pois o texto apareceria palavra por palavra.
- **Segurança (CSRF)**: O *endpoint* da API /chat/gerar/ usa o decorador @csrf_exempt para facilitar a chamada via JavaScript (fetch). Numa aplicação de produção real, isto seria um risco de segurança. A solução correta seria implementar a passagem do token CSRF do Django no cabeçalho da requisição JavaScript.

5. Considerações Éticas

O desenvolvimento de aplicações de IA levanta questões éticas importantes, mesmo em pequena escala:

- **Viés (Bias)**: O modelo foi treinado com uma grande quantidade de dados da internet. Ele pode (e provavelmente irá) reproduzir vieses sociais, culturais ou de linguagem presentes nesses dados.
- **Alucinações e Desinformação**: Sendo um modelo pequeno, ele é suscetível a "alucinar" ou inventar informações que parecem corretas, mas são factualmente erradas. A aplicação não avisa ativamente o usuário sobre esta possibilidade.
- **Privacidade dos Dados (Ponto Positivo)**: Ao rodar o modelo 100% localmente (em vez de usar uma API paga) e ao usar uma base de dados local (ou uma instância Atlas controlada pela equipe), garantimos a **privacidade total** das conversas dos usuários. As perguntas não são enviadas para empresas externas, o que é uma vantagem significativa em termos de segurança de dados.

6. Conclusão

Este projeto foi um sucesso na integração de três tecnologias centrais (Django, Transformers e MongoDB) para criar uma aplicação web de PLN funcional. Os desafios principais—como a migração de framework, a escolha de um modelo de IA estável para CPU, a depuração de conexões de banco de dados e a implementação de filtros NoSQL complexos—foram

superados.

O resultado é uma aplicação que cumpre todos os requisitos funcionais solicitados pelo professor: uma interface de chat funcional, persistência de dados, visualização de histórico com paginação, filtros por texto e data, export