Documentação Breve: Verificador de Processos Judiciais

1. Introdução

O projeto **Judicial Process Verification Engine** é uma aplicação desenvolvida para automatizar a análise de processos judiciais, determinando sua elegibilidade para aquisição de crédito (`approved`, `rejected` ou `incomplete`) com base em políticas de negócio pré-definidas.

O sistema utiliza um **Large Language Model (LLM)** como motor de decisão, garantindo a explicabilidade e a padronização da saída, conforme exigido pelo teste técnico.

2. Arquitetura e Fluxo de Verificação

A aplicação segue o princípio de **Arquitetura Limpa (Clean Architecture)**, separando as preocupações em camadas:

1. **Camada de API (`app/api`):** Recebe a requisição HTTP POST no endpoint `/verify/`.
2. **Camada de Casos de Uso (`app/use_cases`):** O `VerifyProcessUseCase` orquestra a lógica de negócio.
3. **Camada Externa (`app/external`):** O `LLMService` atua como um adaptador para a API do LLM (Groq/Claude).
4. **Camada de Domínio (`app/domain`):** Contém as entidades de dados e as regras de negócio (Políticas).

Fluxo de Decisão (LLM)

O fluxo de verificação é centrado no LLM e ocorre da seguinte forma:

1. O `LLMService` constrói o **contexto de políticas** a partir do arquivo `app/domain/policies.py`.
2. O **prompt de sistema** é montado, contendo:
    - Instruções de *persona* (Especialista em análise de crédito judicial).
    - O **contexto completo das políticas** de negócio.
    - O **JSON do processo judicial** a ser analisado.
    - A **estrutura JSON de saída** esperada.
3. O `LLMService` utiliza o **LangChain** para invocar o LLM, passando o prompt.
4. O LLM processa as informações e gera a decisão.
5. Um `JsonOutputParser` é aplicado para garantir que a saída do LLM seja um JSON válido e estruturado, contendo a `decision`, `rationale` (justificativa) e `citations` (referências às políticas).

3. Uso do LLM para Decisão Estruturada

O LLM é utilizado de forma obrigatória para **consolidar a decisão** e fornecer a **explicabilidade**.

Saída Estruturada

A saída do LLM é rigidamente controlada para ser um JSON com o seguinte formato, garantindo a integração com sistemas a jusante:

```
{
  "decision": "approved" | "rejected" | "incomplete",
  "rationale": "Justificativa concisa baseada nas políticas.",
  "citations": ["POL-1", "POL-2"],
  "confidence": 0.9,
  "policy_analysis": { ... }
}
```

Políticas de Negócio Injetadas

As políticas de negócio (POL-1 a POL-8) são definidas como dados estruturados e injetadas diretamente no prompt do LLM. Isso garante que o modelo baseie sua decisão nas regras específicas da empresa, transformando o LLM em um **motor de regras explicável**.

ID	Regra	Categoria
POL-1	Trânsito em julgado e fase de execução.	Elegibilidade
POL-2	Exigir valor de condenação informado.	Elegibilidade
POL-3	Valor de condenação < R$ 1.000,00 → não compra.	Exclusão
POL-4	Condenações na esfera trabalhista → não compra.	Exclusão
POL-8	Se faltar documento essencial → incomplete.	Documentação

4. Deploy e Observabilidade

O projeto é totalmente conteinerizado, facilitando o deploy em qualquer ambiente:

- **Docker:** `Dockerfile` e `Dockerfile.ui` para construir as imagens da API (FastAPI) e da UI (Streamlit).
- **Orquestração:** `docker-compose.yml` para subir a API e a UI simultaneamente, com *health checks* e dependência entre os serviços.
- **Observabilidade:** Implementação de **Structured Logging (JSON)** e preparação para integração com **LangSmith** para monitoramento de traces, latência e custos do LLM.

A aplicação é acessível via API REST e uma interface Streamlit para testes manuais.