🏛️ Buscador de Jurisprudencia con IA - Provincia de Jujuy

📋 Resumen del Proyecto

Sistema de búsqueda inteligente de jurisprudencia para la Provincia de Jujuy que utiliza IA para clasificar, etiquetar y buscar fallos judiciales de forma semántica, superando las limitaciones del buscador oficial actual.

Inspirado en: JurisprudenciaARG.com pero optimizado específicamente para Jujuy con capacidades de IA más avanzadas.

🎯 Objetivos del Proyecto

Problema a Resolver

- El buscador oficial de Jujuy es básico y difícil de usar
- No tiene clasificación automática ni etiquetado
- Búsqueda limitada a palabras exactas
- Sin resúmenes ni análisis de los fallos

Solución Propuesta

Buscador inteligente que permite:

- ✅ Búsqueda en lenguaje natural ("casos de despido por embarazo")
- ✅ Clasificación y etiquetado automático con IA
- ✅ Resúmenes ejecutivos generados por IA
- ✅ Búsqueda semántica (encuentra conceptos similares)
- ✅ Chat contextual con los fallos
- ✅ Análisis comparativo entre fallos

🚀 Roadmap - Fase 1: MVP (2-4 semanas)

Semana 1-2: Scraping y Base de Datos

- [ ] Desarrollar scraper para jurisprudencia.justiciajujuy.gov.ar
- [ ] Diseñar esquema de base de datos
- [ ] Implementar PostgreSQL con extensión pgvector
- [ ] Crear pipeline de ingesta de datos

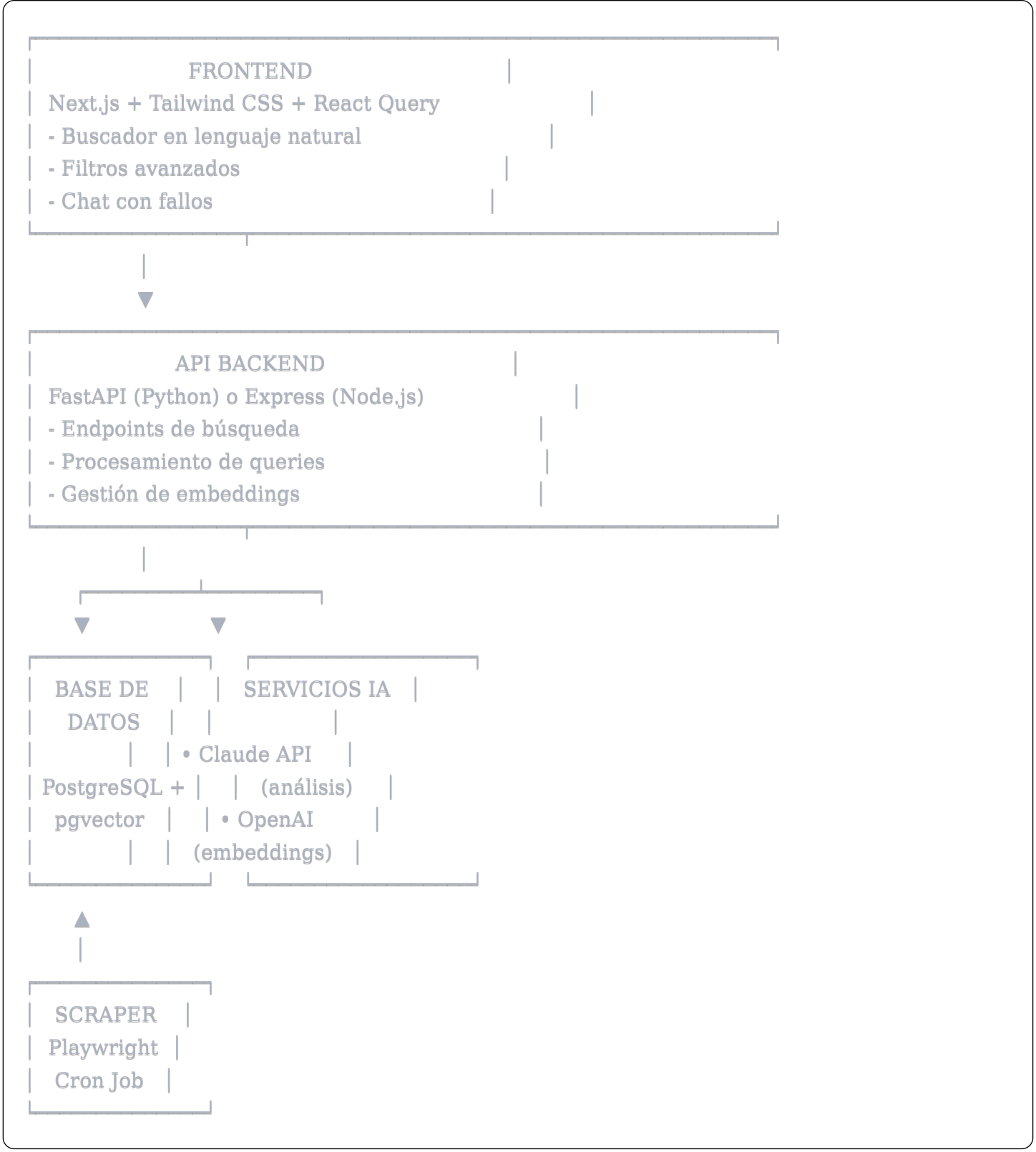
Semana 2-3: Procesamiento con IA

- [ ] Integrar Claude API para análisis de fallos
- [ ] Implementar sistema de etiquetado automático
- [ ] Generar embeddings para búsqueda semántica
- [ ] Crear resúmenes automáticos

Semana 3-4: Interfaz y Búsqueda

- [ ] Desarrollar frontend con Next.js
- [ ] Implementar búsqueda en lenguaje natural
- [ ] Crear sistema de filtros avanzados
- [ ] Diseñar vista de resultados con resúmenes

🏗️ Arquitectura del Sistema



💾 Modelo de Datos

Tabla: `fallos`

```sql
CREATE TABLE fallos (
    id SERIAL PRIMARY KEY,
    caratula TEXT NOT NULL,
    fecha_fallo DATE,
    tribunal VARCHAR(255),
    expediente VARCHAR(100),
    materia VARCHAR(100),
    tipo_proceso VARCHAR(100),
    juez VARCHAR(255),
    texto_completo TEXT,
    resumen_ia TEXT,
    resultado VARCHAR(50),
    url_original TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

Tabla: etiquetas

```sql
CREATE TABLE etiquetas (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) UNIQUE NOT NULL,
    categoria VARCHAR(50),
    descripcion TEXT
);
```

Tabla: fallo_etiquetas

```sql
CREATE TABLE fallo_etiquetas (
    fallo_id INTEGER REFERENCES fallos(id),
    etiqueta_id INTEGER REFERENCES etiquetas(id),
    confianza FLOAT,
    PRIMARY KEY (fallo_id, etiqueta_id)
);
```

Tabla: embeddings

```sql
CREATE TABLE embeddings (
    fallo_id INTEGER PRIMARY KEY REFERENCES fallos(id),
    embedding vector(1536),
    modelo VARCHAR(50)
);

CREATE INDEX ON embeddings USING ivfflat (embedding vector_cosine_ops);
```

🤖 Sistema de Etiquetado con IA

Prompt para Claude API

python

```
PROMPT_ETIQUETADO = """
Analiza el siguiente fallo judicial de la Provincia de Jujuy y extrae:

1. **Resumen ejecutivo** (máximo 150 palabras): Describe los hechos principales,
   el conflicto legal y la resolución del tribunal.

2. **Palabras clave** (máximo 10): Términos jurídicos relevantes del caso.

3. **Categorías principales**:
   - Materia: (Ej: Civil, Penal, Laboral, Familia, Contencioso Administrativo)
   - Tipo de proceso: (Ej: Amparo, Recurso de Apelación, Juicio Ordinario)
   - Subtemas: (Ej: Despido, Daños y Perjuicios, Inconstitucionalidad)

4. **Resultado**: (Ej: Se hizo lugar, Se rechazó, Se confirmó, Se revocó)

5. **Partes**: Actor/Demandante y Demandado

6. **Normas citadas**: Leyes, artículos o códigos mencionados

Responde ÚNICAMENTE en formato JSON siguiendo esta estructura:
{
  "resumen": "...",
  "palabras_clave": ["...", "..."],
  "materia": "...",
  "tipo_proceso": "...",
  "subtemas": ["...", "..."],
  "resultado": "...",
  "actor": "...",
  "demandado": "...",
  "normas_citadas": ["...", "..."]
}

FALLO:
{texto_fallo}
"""
```

Implementación en Python

```python
import anthropic
import json

def etiquetar_fallo(texto_fallo: str) -> dict:
    client = anthropic.Anthropic(api_key=os.environ.get("ANTHROPIC_API_KEY"))

    message = client.messages.create(
        model="claude-sonnet-4-5-20250929",
        max_tokens=2000,
        messages=[{
            "role": "user",
            "content": PROMPT_ETIQUETADO.format(texto_fallo=texto_fallo)
        }]
    )

    # Extraer JSON de la respuesta
    respuesta = message.content[0].text
    # Limpiar posibles markdown
    respuesta = respuesta.replace("```json", "").replace("```", "").strip()

    return json.loads(respuesta)
```

🔍 Sistema de Búsqueda Semántica

Generación de Embeddings

```python
import openai

def generar_embedding(texto: str) -> list:
    """Genera embedding vectorial del texto"""
    response = openai.embeddings.create(
        model="text-embedding-3-small",
        input=texto
    )
    return response.data[0].embedding
```

Búsqueda por Similitud

```python
def buscar_fallos_similares(query: str, limit: int = 10):
    """Busca fallos similares usando embeddings"""

    # Generar embedding de la consulta
    query_embedding = generar_embedding(query)

    # Búsqueda vectorial en PostgreSQL
    sql = """
        SELECT
            f.id,
            f.caratula,
            f.resumen_ia,
            f.fecha_fallo,
            1 - (e.embedding <=> %s::vector) as similitud
        FROM fallos f
        JOIN embeddings e ON f.id = e.fallo_id
        ORDER BY e.embedding <=> %s::vector
        LIMIT %s
    """

    return ejecutar_query(sql, (query_embedding, query_embedding, limit))
```

🎨 Stack Tecnológico

Backend

- **Lenguaje**: Python 3.11+
- **Framework**: FastAPI
- **Base de datos**: PostgreSQL 15+ con pgvector
- **IA**:
  - Claude API (Sonnet 4.5) para análisis
  - OpenAI API para embeddings
- **Scraping**: Playwright
- **Cache**: Redis (opcional para MVP)

Frontend

- **Framework**: Next.js 14+ (App Router)
- **UI**: Tailwind CSS + shadcn/ui
- **Estado**: React Query / TanStack Query
- **Gráficos**: Recharts (para estadísticas)

DevOps

- **Hosting**: Vercel (Frontend) + Railway/Render (Backend)
- **CI/CD**: GitHub Actions
- **Monitoreo**: Sentry (errores) + Posthog (analytics)

🎯 Diferenciadores vs JurisprudenciaARG

Característica	JurisprudenciaARG	Nuestro MVP
Búsqueda en lenguaje natural	❌	✅
Chat con fallos	❌	✅
Búsqueda semántica	Limitada	✅ Avanzada
Análisis comparativo	❌	✅
Resúmenes con IA	✅	✅
Etiquetado automático	✅	✅
Alertas personalizadas	✅	✅ (Fase 2)
Enfoque geográfico	Nacional	Jujuy (especializado)

💡 Características Innovadoras

1. Chat Contextual con Fallos

```
Usuario: "¿Qué dice este fallo sobre la carga de la prueba?"
IA: "El tribunal estableció que en casos de despido discriminatorio,
    la carga de la prueba se invierte, correspondiendo al empleador
    demostrar que el despido no fue discriminatorio..."
```

2. Comparador de Fallos

```
Usuario: Selecciona 3 fallos similares
Sistema: Genera tabla comparativa con:
- Argumentos principales
- Normas citadas
- Resultados
- Diferencias clave
```

3. Análisis de Tendencias

```
"En los últimos 6 meses, el 73% de los fallos sobre despidos
 sin causa en Jujuy han favorecido al trabajador, un aumento
 del 15% respecto al año anterior."
```

📊 Modelo de Negocio

Plan Gratuito

- 5 búsquedas por mes
- Acceso a resúmenes
- Filtros básicos

Plan Pro ($9.99/mes)

- Búsquedas ilimitadas
- Chat con fallos
- Alertas personalizadas
- Exportación a PDF
- Comparador de fallos

Plan Enterprise (Precio personalizado)

- Todo lo de Pro +
- API para integración
- Análisis de tendencias
- Soporte prioritario
- Capacitación del equipo

🔧 Instalación y Configuración

Requisitos Previos

```bash
- Python 3.11+
- Node.js 18+
- PostgreSQL 15+
- Redis (opcional)
```

Backend Setup

bash

```
# Clonar repositorio
git clone https://github.com/tu-usuario/jurisprudencia-jujuy-ia.git
cd jurisprudencia-jujuy-ia/backend

# Crear entorno virtual
python -m venv venv
source venv/bin/activate  # Linux/Mac
# venv\Scripts\activate  # Windows

# Instalar dependencias
pip install -r requirements.txt

# Configurar variables de entorno
cp .env.example .env
# Editar .env con tus API keys

# Crear base de datos
createdb jurisprudencia_jujuy
psql jurisprudencia_jujuy < schema.sql

# Instalar extensión pgvector
psql jurisprudencia_jujuy -c "CREATE EXTENSION vector;"

# Ejecutar migraciones
alembic upgrade head

# Iniciar servidor
uvicorn main:app --reload
```

Frontend Setup

bash

```
cd ../frontend

# Instalar dependencias
npm install

# Configurar variables de entorno
cp .env.example .env.local

# Iniciar desarrollo
npm run dev
```

📝 Ejemplo de Uso del Scraper

python

```python
# scraper/jujuy_scraper.py
from playwright.sync_api import sync_playwright
import time

def scrape_fallos_jujuy(max_pages=10):
    """Scraper para jurisprudencia de Jujuy"""

    with sync_playwright() as p:
        browser = p.chromium.launch(headless=True)
        page = browser.new_page()

        base_url = "https://jurisprudencia.justiciajujuy.gov.ar/public/buscador"
        fallos = []

        for i in range(max_pages):
            page.goto(f"{base_url}?index={i}")
            page.wait_for_selector(".resultado-fallo")

            # Extraer información de cada fallo
            items = page.query_selector_all(".resultado-fallo")

            for item in items:
                fallo = {
                    'caratula': item.query_selector('.caratula').inner_text(),
                    'fecha': item.query_selector('.fecha').inner_text(),
                    'tribunal': item.query_selector('.tribunal').inner_text(),
                    'url': item.query_selector('a').get_attribute('href')
                }

                # Ir al detalle del fallo
                detail_page = browser.new_page()
                detail_page.goto(fallo['url'])
                fallo['texto_completo'] = detail_page.query_selector('.texto-fallo').inner_text()
                detail_page.close()

                fallos.append(fallo)
                time.sleep(1)  # Respetar el servidor

        browser.close()
        return fallos
```

🧪 Testing

```bash
# Backend tests
pytest tests/ -v

# Frontend tests
npm run test

# E2E tests
npm run test:e2e
```

📈 Métricas de Éxito (KPIs)

Técnicos

- Tiempo de respuesta de búsqueda: < 2 segundos
- Precisión de etiquetado: > 85%
- Disponibilidad del sistema: > 99%

Negocio

- Usuarios activos mensuales: Meta 500 en 3 meses
- Tasa de conversión Free → Pro: > 5%
- NPS (Net Promoter Score): > 50

🗓️ Fases Futuras

Fase 2: Expansión (Mes 2-3)

- [ ] Alertas por email/WhatsApp
- [ ] Análisis de tendencias jurídicas
- [ ] Exportación a Word/PDF con formato
- [ ] API pública para desarrolladores

Fase 3: Avanzado (Mes 4-6)

- [ ] Predicción de resultados con ML
- [ ] Integración con otras provincias
- [ ] Asistente legal virtual completo
- [ ] Generación de escritos básicos

🤝 Contribución

```bash
# Fork el proyecto
# Crea una rama para tu feature
git checkout -b feature/nueva-funcionalidad

# Commit tus cambios
git commit -m "Añade nueva funcionalidad"

# Push a la rama
git push origin feature/nueva-funcionalidad

# Abre un Pull Request
```

📄 Licencia

Este proyecto está bajo la Licencia MIT. Ver archivo `LICENSE` para más detalles.

📞 Contacto

- **Desarrollador**: Tu Nombre
- **Email**: tu@email.com
- **LinkedIn**: Tu perfil
- **GitHub**: @tu-usuario

🙏 Agradecimientos

- Inspiración: JurisprudenciaARG.com
- Datos: Poder Judicial de Jujuy
- IA: Claude API y OpenAI

¿Preguntas? Abre un issue en GitHub o contacta directamente.

📚 Recursos Adicionales

- Documentación de Claude API
- Guía de pgvector
- Best Practices para Embeddings
- FastAPI Documentation
- Next.js Documentation