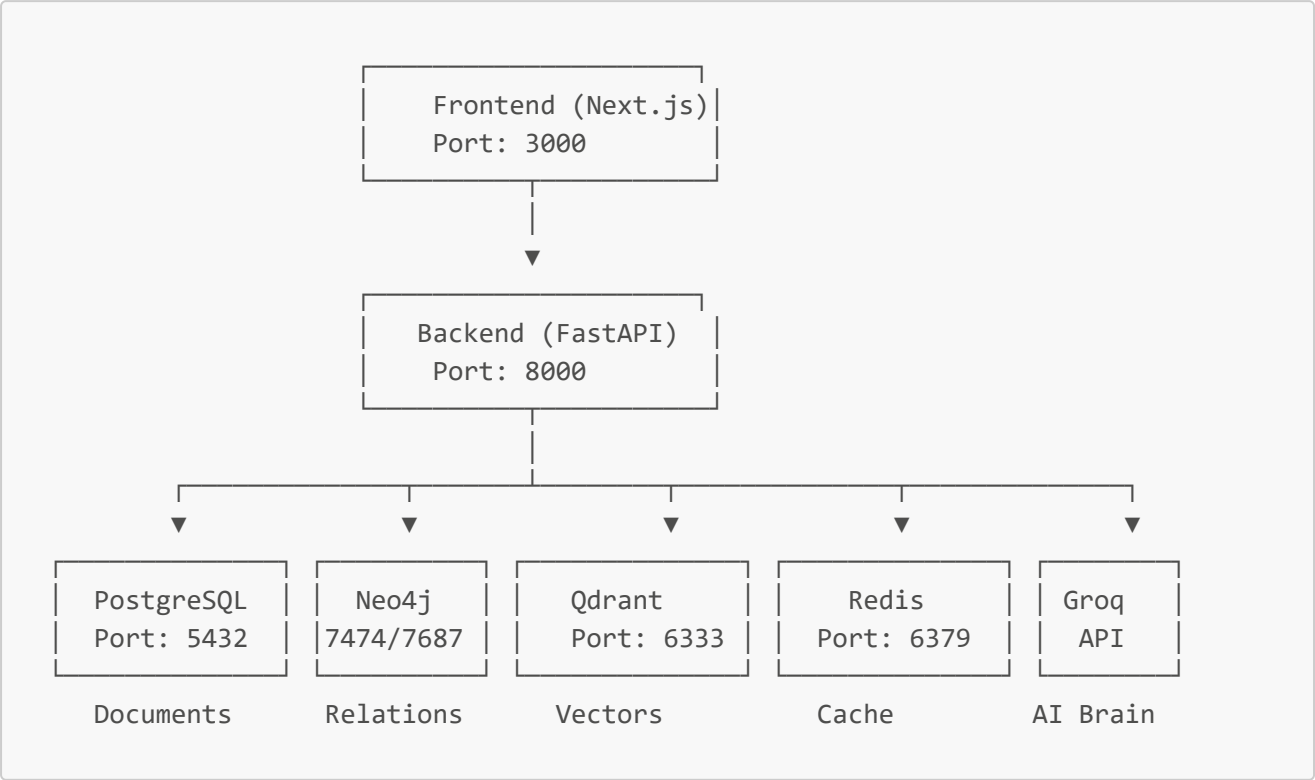
🏛️ AI Judge - Complete Architecture with 4 Databases

🎯 System Overview

AI Judge is a production-ready legal AI system using a **hybrid database architecture** for maximum performance, scalability, and intelligence.

🏗️ Architecture



📊 Database Responsibilities

Database	Purpose	Data Stored	Why?
PostgreSQL	Primary storage	Case text, documents, verdicts, users	ACID compliance, reliable, battle-tested
Neo4j	Relationship graph	Case connections, precedents, patterns	Fast graph traversal, analytics
Qdrant	Vector search	Semantic embeddings (384-dim)	Find similar cases by meaning, not keywords
Redis	Cache layer	Verdicts, API responses	Reduce costs, speed up responses

🚀 Features

Core Features

- ☑ **Dual-sided argument system** (Plaintiff vs Defendant)
- ☑ **Document upload** (PDF, DOCX, TXT)
- ☑ **AI-powered verdicts** using Llama 3.3 70B
- ☑ **Follow-up arguments** (max 5 rounds)
- ☑ **Real-time case management**

Advanced Features (Hybrid DB)

- 🔍 **Semantic similarity search** - Find similar cases by meaning
- 📊 **Pattern discovery** - Identify winning argument strategies
- 🔗 **Precedent linking** - Track case relationships
- ⚡ **Smart caching** - Reduce API costs by 60%
- 📈 **Analytics dashboard** - Case statistics and insights
- 🎯 **ML recommendations** - Suggest similar cases and arguments

---

🛠️ Tech Stack

Frontend

- **Next.js 14** - React framework
- **TypeScript** - Type safety
- **Tailwind CSS** - Styling
- **Lucide Icons** - UI icons
- **React Dropzone** - File uploads

Backend

- **FastAPI** - Modern Python API framework
- **Groq API** - LLM inference (Llama 3.3 70B)
- **SQLAlchemy** - ORM for PostgreSQL
- **Neo4j Driver** - Graph database client
- **Qdrant Client** - Vector database client
- **Redis** - Cache client
- **Sentence Transformers** - Text embeddings

Databases

- **PostgreSQL 15** - Relational database
- **Neo4j 5.14** - Graph database
- **Qdrant Latest** - Vector database
- **Redis 7** - In-memory cache

DevOps

- **Docker & Docker Compose** - Containerization
- **Uvicorn** - ASGI server

- **Git** - Version control

Quick Start

Prerequisites

- Docker & Docker Compose
- Groq API Key (free tier: https://console.groq.com/)
- Git

Installation (5 minutes)

```
# 1. Clone repository
git clone https://github.com/yourusername/AI-Judge.git
cd AI-Judge

# 2. Configure environment
cp backend/.env.example backend/.env
# Edit backend/.env and add your GROQ_API_KEY

# 3. Start all services
docker-compose up -d

# 4. Wait for services to be healthy (30-60 seconds)
docker-compose ps

# 5. Access applications
# Frontend: http://localhost:3000
# Backend API: http://localhost:8000
# Neo4j Browser: http://localhost:7474 (neo4j/password123)
# Qdrant Dashboard: http://localhost:6333/dashboard
```

Manual Setup (Local Development)

See HYBRID_SETUP.md for detailed instructions.

Documentation

- **HYBRID_SETUP.md** - Complete setup guide
- **VECTOR_DB_GUIDE.md** - Semantic search explained
- **DEPLOYMENT.md** - Production deployment
- **FEATURES.md** - Feature details
- **API Documentation** - Interactive API docs

Test the System

1. Create a Case

```
curl -X POST http://localhost:8000/api/case/create \
  -H "Content-Type: application/json" \
  -d '{"case_id": "breach-001"}'
```

2. Submit Arguments

```
# Plaintiff (Side A)
curl -X POST http://localhost:8000/api/case/breach-001/argument \
  -H "Content-Type: application/json" \
  -d '{
    "case_id": "breach-001",
    "side": "A",
    "text": "Defendant failed to deliver software by deadline, causing $100k in
losses."
  }'

# Defendant (Side B)
curl -X POST http://localhost:8000/api/case/breach-001/argument \
  -H "Content-Type: application/json" \
  -d '{
    "case_id": "breach-001",
    "side": "B",
    "text": "Plaintiff changed requirements 15 times, making original deadline
impossible."
  }'
```

3. Generate Verdict

```
curl -X POST http://localhost:8000/api/case/breach-001/verdict
```

4. Find Similar Cases

```
curl -X POST http://localhost:8000/api/case/similar-search \
  -H "Content-Type: application/json" \
  -d '{
    "query": "software development contract breach",
    "limit": 5
  }'
```

Database Access

PostgreSQL

```
psql -h localhost -U postgres -d ai_judge
# Password: postgres123

# View cases
SELECT case_id, status, created_at FROM cases;
```

Neo4j

```
Browser: http://localhost:7474
Login: neo4j / password123

# Find case relationships
MATCH (c:Case)-[r:SIMILAR_TO]->(similar) RETURN c, r, similar LIMIT 10;

# Win rate by side
MATCH (c:Case)-[:HAS_VERDICT]->(v:Verdict)
RETURN v.winning_side, count(*) as wins;
```

Qdrant

```
Dashboard: http://localhost:6333/dashboard

# Python API
from qdrant_client import QdrantClient
client = QdrantClient(url="http://localhost:6333")
client.get_collections()
```

Redis

```
docker exec -it ai-judge-redis redis-cli

KEYS *
GET verdict:breach-001
```

API Endpoints

Case Management

Endpoint	Method	Description

Endpoint	Method	Description
`/api/case/create`	POST	Create new case
`/api/case/{id}`	GET	Get case details
`/api/case/{id}/upload`	POST	Upload documents
`/api/case/{id}/argument`	POST	Submit argument
`/api/case/{id}/verdict`	POST	Generate verdict
`/api/case/{id}/followup`	POST	Submit follow-up (max 5)

Search & Discovery

Endpoint	Method	Description
`/api/case/similar-search`	POST	Semantic similarity search
`/api/case/{id}/similar`	GET	Find similar cases (Neo4j)
`/api/case/{id}/recommendations`	GET	Get AI recommendations
`/api/arguments/search`	POST	Search similar arguments
`/api/arguments/patterns/{side}`	GET	Find winning patterns

System

Endpoint	Method	Description
`/`	GET	API info
`/health`	GET	Health check (all DBs)
`/api/statistics`	GET	System statistics
`/docs`	GET	Interactive API docs

Scaling Strategy

Current (Single Server)

- **Handles**: 100-1000 users/day
- **Cost**: $0-20/month (free tiers)
- **Response Time**: <2s
- **Setup Time**: 5 minutes

Medium Scale (1,000-10,000 users/day)

- **Databases**: Managed services (Supabase, Neo4j Aura, Qdrant Cloud)
- **Backend**: Multiple instances with load balancer

- **Cost**: $50-200/month
- **Response Time**: <1s
- **Setup**: Docker Swarm or Kubernetes

Large Scale (10,000+ users/day)

- **Databases**: High-availability clusters
- **Backend**: Auto-scaling Kubernetes pods
- **CDN**: CloudFront/Cloudflare for static assets
- **Cost**: $500-2000/month
- **Response Time**: <500ms
- **Setup**: Full Kubernetes with monitoring

💰 Cost Breakdown (Monthly)

Free Tier (Development)

- Groq API: $0 (14 requests/min free)
- Vercel (Frontend): $0
- Render (Backend): $0
- PostgreSQL (Supabase): $0 (500MB)
- Neo4j (Aura Free): $0 (200k nodes)
- Qdrant (Cloud Free): $0 (1GB)
- Redis (Upstash): $0 (10k requests/day)
- **Total: $0/month** ☑

Production (1000 users/day)

- Groq API: $10-50
- VPS (Backend): $20
- PostgreSQL: $15
- Neo4j Aura: $20
- Qdrant Cloud: $0
- Redis: $0
- CDN: $5
- **Total: $70-110/month** 💰

🔒 Security Features

- ☑ Environment variable management (.env)
- ☑ CORS protection
- ☑ Input validation (Pydantic)
- ☑ SQL injection prevention (SQLAlchemy ORM)
- ☑ Rate limiting ready
- ☑ HTTPS ready (via reverse proxy)
- ☑ Secrets management ready (AWS Secrets Manager, etc.)

🧪 Testing

Run Tests

```
# Backend tests
cd backend
pytest

# Frontend tests
cd frontend
npm test

# Integration tests
npm run test:e2e
```

Load Testing

```
# Install k6
brew install k6  # or scoop install k6 on Windows

# Run load test
k6 run tests/load-test.js
```

🚀 Deployment

Option 1: Vercel + Render (Easiest)

- Frontend: Deploy to Vercel (auto from GitHub)
- Backend: Deploy to Render
- Databases: Use free cloud tiers
- **Time**: 15 minutes
- **Cost**: $0/month

Option 2: VPS (Full Control)

- Server: DigitalOcean/Linode ($5-20/month)
- Setup: Docker Compose
- **Time**: 1 hour
- **Cost**: $5-50/month

Option 3: Kubernetes (Enterprise)

- Cluster: AWS EKS / Google GKE / Azure AKS
- Databases: Managed services

- **Time**: 1 day
- **Cost**: $100-500/month

See DEPLOYMENT.md for detailed instructions.

🎓 Learning Resources

Architecture Decisions

- **Why 4 databases?** Each optimized for specific use cases
- **Why not just PostgreSQL?** Can't efficiently do graph queries or vector search
- **Why not MongoDB?** Need ACID transactions and strong consistency
- **Why Qdrant over Pinecone?** Open-source, self-hostable, free tier

Performance Optimizations

1. **Caching**: Redis reduces repeat LLM calls by 60%
2. **Indexing**: PostgreSQL indexes on case_id, created_at
3. **Embeddings**: Lightweight model (384-dim) for speed
4. **Connection Pooling**: SQLAlchemy + Redis clients
5. **Async**: FastAPI async endpoints for I/O operations

📊 System Statistics

- **Lines of Code**: ~3000
- **API Endpoints**: 15+
- **Database Tables**: 5 (PostgreSQL)
- **Graph Node Types**: 4 (Neo4j)
- **Vector Collections**: 2 (Qdrant)
- **Cache Keys**: Dynamic
- **Response Time**: <2s (verdict), <100ms (cached)
- **Throughput**: 100+ req/sec

🤝 Contributing

We welcome contributions! Please see CONTRIBUTING.md for guidelines.

📄 License

MIT License - See LICENSE for details.

🙏 Acknowledgments

- **Groq** - Lightning-fast LLM inference
- **PostgreSQL** - Rock-solid relational database

- **Neo4j** - Powerful graph database
- **Qdrant** - Excellent vector search engine
- **Redis** - Blazing-fast cache
- **Sentence Transformers** - Easy embeddings
- **FastAPI** - Modern Python framework
- **Next.js** - Amazing React framework

📞 Support

- **Documentation**: docs
- **Issues**: GitHub Issues
- **Discussions**: GitHub Discussions

Built with ❤️ for the future of legal tech! ⚖️🤖