**Especificação Técnica Completa - Sistema Jurídico com IA**

**1. Visão Geral da Arquitetura**

**1.1 Arquitetura Híbrida Cloud-First**

O sistema adota uma arquitetura híbrida moderna, priorizando SaaS com componentes desktop opcionais:

**Core Platform (SaaS)**

* **Frontend**: React 18+ com TypeScript, Next.js 14, TailwindCSS
* **Backend**: Node.js com NestJS framework, PostgreSQL, Redis
* **Infraestrutura**: AWS/Azure com Kubernetes, auto-scaling
* **API Gateway**: Kong/AWS API Gateway para roteamento e rate limiting

**Desktop Agent (Opcional)**

* **Framework**: Electron com React renderer
* **IA Local**: Ollama/LM Studio para modelos locais (Llama 3.1, Phi-3)
* **Storage**: SQLite com encryption AES-256
* **Sync**: Event-driven sync com cloud platform

**1.2 Microserviços Especializados**

graph TB

A[API Gateway] --> B[Auth Service]

A --> C[Document Service]

A --> D[AI Engine Service]

A --> E[Legal Database Service]

A --> F[Deadline Service]

A --> G[Case Management Service]

A --> H[Court Integration Service]

D --> I[RAG Engine]

D --> J[Citation Validator]

D --> K[Anti-Hallucination Engine]

E --> L[Jurisprudence DB]

E --> M[Legislation DB]

E --> N[Templates DB]

**2. Sistema de IA - Motor Anti-Alucinação**

**2.1 RAG (Retrieval-Augmented Generation) Jurídico**

**Base de Conhecimento Jurídico**

* **Fontes Primárias**: Constituição, códigos, leis federais/estaduais/municipais
* **Jurisprudência**: STF, STJ, TST, TRFs, TJs, TRTs (últimos 10 anos)
* **Documentos Normativos**: Súmulas, temas repetitivos, IRDR, IAC
* **Atualizações**: Pipeline automatizado via APIs e webscraping ético

**Indexação Semântica**

interface LegalChunk {

id: string;

content: string;

metadata: {

source: 'legislation' | 'jurisprudence' | 'doctrine';

document\_type: string;

court?: string;

date: Date;

status: 'active' | 'revoked' | 'superseded';

confidence\_score: number;

};

embeddings: number[];

legal\_references: string[];

}

**Embedding Strategy**

* **Modelo Principal**: multilingual-e5-large para português jurídico
* **Chunking**: Semântico baseado em estrutura jurídica (artigos, parágrafos, ementas)
* **Vector Store**: Weaviate/Pinecone com filtros de metadados

**2.2 Sistema Anti-Alucinação Multicamadas**

**Layer 1: Context Validation**

def validate\_context(query: str, retrieved\_docs: List[Document]) -> ValidationResult:

"""Valida se o contexto recuperado é suficiente para responder a consulta"""

relevance\_scores = calculate\_relevance(query, retrieved\_docs)

coverage\_score = calculate\_coverage(query, retrieved\_docs)

if coverage\_score < MINIMUM\_COVERAGE\_THRESHOLD:

return ValidationResult(

status="insufficient\_context",

message="Informações insuficientes na base de conhecimento",

suggested\_action="request\_more\_information"

)

**Layer 2: Citation Verification**

class CitationValidator:

def verify\_citations(self, generated\_text: str) -> List[CitationStatus]:

"""Verifica todas as citações no texto gerado"""

citations = extract\_citations(generated\_text)

results = []

for citation in citations:

verification = self.\_verify\_against\_source(citation)

if not verification.is\_valid:

results.append(CitationStatus(

citation=citation,

status="invalid",

error=verification.error\_message,

suggested\_correction=verification.suggestion

))

return results

**Layer 3: Confidence Scoring**

class ConfidenceScorer:

def calculate\_confidence(self,

query: str,

context: List[Document],

generated\_response: str) -> float:

"""Calcula score de confiança da resposta gerada"""

context\_relevance = self.\_calculate\_context\_relevance(query, context)

citation\_accuracy = self.\_validate\_citations(generated\_response)

factual\_consistency = self.\_check\_factual\_consistency(context, generated\_response)

return weighted\_average([context\_relevance, citation\_accuracy, factual\_consistency])

**2.3 Modelos de IA Especializados**

**Orchestrator LLM**: GPT-4o/Claude-3.5-Sonnet para coordenação geral **Specialized Models**:

* **Legal Extraction**: Fine-tuned BERT para extração de entidades jurídicas
* **Document Classification**: Custom model para classificação de peças
* **Deadline Calculator**: Rule-based system com ML para casos complexos
* **Citation Formatter**: Regex + NLP para formatação ABNT/jurídica

**3. Módulos Funcionais Detalhados**

**3.1 Editor Jurídico Inteligente**

**Arquitetura do Editor**

interface DocumentStructure {

header: {

court: string;

process\_number?: string;

parties: Party[];

};

sections: {

qualification: QualificationSection;

facts: FactsSection;

legal\_grounds: LegalGroundsSection;

requests: RequestsSection;

evidence: EvidenceSection;

closure: ClosureSection;

};

metadata: DocumentMetadata;

}

class SmartEditor {

private aiAssistant: AIAssistant;

private templateEngine: TemplateEngine;

private citationValidator: CitationValidator;

async generateSection(sectionType: string, context: CaseContext): Promise<Section> {

const template = await this.templateEngine.getTemplate(sectionType);

const aiSuggestion = await this.aiAssistant.generateContent({

template,

context,

constraints: {

max\_tokens: 2000,

require\_citations: true,

confidence\_threshold: 0.8

}

});

const validatedContent = await this.citationValidator.validate(aiSuggestion);

return this.createSection(validatedContent);

}

}

**Templates Jurídicos Estruturados**

* **Petições Iniciais**: Cível, trabalhista, penal (20+ variações)
* **Procurações**: Ad judicia, ad negotia, com poderes específicos
* **Contestações**: Por rito e área do direito
* **Recursos**: Apelação, agravo, embargos, especial, extraordinário
* **Documentos Auxiliares**: Memoriais, pareceres, ofícios, requerimentos

**3.2 Sistema de Gestão de Prazos**

**Calculadora de Prazos Inteligente**

class DeadlineCalculator:

def \_\_init\_\_(self):

self.court\_calendars = CourtCalendarManager()

self.legal\_rules = LegalDeadlineRules()

def calculate\_deadline(self,

trigger\_event: Event,

deadline\_type: str,

court: str,

process\_type: str) -> DeadlineResult:

base\_rule = self.legal\_rules.get\_rule(deadline\_type, process\_type)

court\_calendar = self.court\_calendars.get\_calendar(court)

# CPC Art. 219 - Dias úteis

working\_days = base\_rule.days

start\_date = trigger\_event.date

# Calcula considerando feriados forenses, férias coletivas

end\_date = self.\_calculate\_business\_days(

start\_date,

working\_days,

court\_calendar.holidays,

court\_calendar.collective\_vacations

)

return DeadlineResult(

end\_date=end\_date,

working\_days=working\_days,

confidence=0.95,

legal\_basis=base\_rule.legal\_reference,

warnings=self.\_check\_warnings(end\_date, court\_calendar)

)

**Monitoramento Automatizado**

* **Integração com Diários**: Webscraping ético + APIs quando disponíveis
* **OCR de Publicações**: Tesseract + Azure Document Intelligence
* **Alertas Multi-canal**: Email, SMS, WhatsApp API, Teams/Slack
* **Calendário Sincronizado**: Outlook, Google Calendar, CalDAV

**3.3 Pesquisa Jurídica Avançada**

**Query Engine Híbrido**

class LegalSearchEngine:

def \_\_init\_\_(self):

self.vector\_db = WeaviateClient()

self.full\_text\_db = ElasticsearchClient()

self.query\_expander = QueryExpander()

async def search(self, query: str, filters: SearchFilters) -> SearchResults:

# Expandir query com sinônimos jurídicos

expanded\_query = await self.query\_expander.expand(query)

# Busca vetorial semântica

vector\_results = await self.vector\_db.search(

expanded\_query,

filters=filters,

limit=50

)

# Busca textual exata

text\_results = await self.full\_text\_db.search(

query,

filters=filters,

limit=50

)

# Fusão de resultados com re-ranking

merged\_results = self.\_merge\_and\_rerank(vector\_results, text\_results)

# Validação de relevância

validated\_results = await self.\_validate\_relevance(merged\_results, query)

return SearchResults(

results=validated\_results,

total\_found=len(validated\_results),

search\_metadata=SearchMetadata(

query\_expansion=expanded\_query,

search\_time=time.time() - start\_time,

sources\_consulted=filters.sources

)

)

**3.4 Sistema de Gestão de Casos**

**Estrutura de Dados do Caso**

interface Case {

id: string;

metadata: {

number?: string;

court: string;

subject: string;

area\_of\_law: string[];

parties: Party[];

lawyers: Lawyer[];

status: CaseStatus;

created\_at: Date;

updated\_at: Date;

};

folders: {

initial\_documents: Document[];

evidence: Document[];

expert\_reports: Document[];

court\_decisions: Document[];

correspondence: Document[];

internal\_memos: Document[];

};

timeline: TimelineEvent[];

deadlines: Deadline[];

tasks: Task[];

financial: FinancialData;

ai\_insights: AIInsight[];

}

**Document Intelligence**

class DocumentProcessor:

def \_\_init\_\_(self):

self.ocr\_engine = TesseractOCR()

self.nlp\_processor = SpacyNLP()

self.classifier = DocumentClassifier()

async def process\_document(self, document: UploadedFile) -> ProcessedDocument:

# OCR se necessário

text = await self.\_extract\_text(document)

# Classificação automática

classification = await self.classifier.classify(text)

# Extração de entidades

entities = await self.nlp\_processor.extract\_entities(text)

# Análise de relevância para o caso

relevance\_score = await self.\_calculate\_relevance(text, case\_context)

return ProcessedDocument(

original\_file=document,

extracted\_text=text,

classification=classification,

entities=entities,

relevance\_score=relevance\_score,

suggested\_folder=classification.suggested\_folder,

metadata=self.\_extract\_metadata(text, entities)

)

**4. Segurança e Conformidade**

**4.1 Criptografia End-to-End**

**Arquitetura de Segurança**

class SecurityManager:

def \_\_init\_\_(self):

self.encryption = AESEncryption(key\_size=256)

self.key\_manager = HashiCorpVault() # Gestão de chaves

self.audit\_logger = AuditLogger()

def encrypt\_sensitive\_data(self, data: bytes, user\_id: str) -> EncryptedData:

user\_key = self.key\_manager.get\_user\_key(user\_id)

encrypted = self.encryption.encrypt(data, user\_key)

# Log de auditoria sem dados sensíveis

self.audit\_logger.log\_encryption\_event(

user\_id=user\_id,

data\_type="case\_document",

timestamp=datetime.now(),

operation="encrypt"

)

return encrypted

**4.2 Conformidade LGPD**

**Gestão de Consentimento e Dados**

interface DataSubject {

id: string;

name: string;

email: string;

consent\_records: ConsentRecord[];

data\_processing\_purposes: ProcessingPurpose[];

retention\_policy: RetentionPolicy;

}

class LGPDCompliance {

async exportUserData(userId: string): Promise<DataExport> {

const userData = await this.collectAllUserData(userId);

const anonymizedData = await this.anonymizeData(userData);

return {

personal\_data: userData.personal,

case\_data: anonymizedData.cases,

generated\_documents: userData.documents,

ai\_interactions: anonymizedData.ai\_logs,

export\_date: new Date(),

retention\_expires: this.calculateRetentionExpiry(userId)

};

}

async forgetUser(userId: string): Promise<ForgetResult> {

// Soft delete with crypto-shredding for immediate effect

await this.cryptoShred(userId);

// Schedule hard delete after legal retention period

await this.scheduleHardDelete(userId, retention\_period);

return { status: 'forgotten', scheduled\_deletion: retention\_period };

}

}

**4.3 Auditoria e Compliance**

**Sistema de Auditoria Completo**

class AuditSystem:

def \_\_init\_\_(self):

self.blockchain = EthereumClient() # Para imutabilidade opcional

self.logger = StructuredLogger()

def log\_ai\_generation(self,

user\_id: str,

document\_type: str,

sources\_used: List[str],

confidence\_score: float,

generated\_content\_hash: str):

audit\_record = AuditRecord(

timestamp=datetime.now(),

user\_id=user\_id,

action="ai\_document\_generation",

metadata={

"document\_type": document\_type,

"sources\_count": len(sources\_used),

"confidence\_score": confidence\_score,

"content\_hash": generated\_content\_hash,

"model\_version": self.get\_current\_model\_version()

}

)

# Log local + blockchain opcional para casos críticos

self.logger.log(audit\_record)

if document\_type in ["petition", "appeal"]:

self.blockchain.store\_hash(audit\_record.hash)

**5. Integrações e Conectores**

**5.1 Tribunais e Sistemas Processuais**

**Connector Framework**

class CourtConnector(ABC):

@abstractmethod

async def search\_processes(self, query: ProcessQuery) -> List[Process]:

pass

@abstractmethod

async def download\_process\_documents(self, process\_id: str) -> List[Document]:

pass

@abstractmethod

async def submit\_petition(self, petition: Petition, credentials: Credentials) -> SubmissionResult:

pass

class PJeConnector(CourtConnector):

"""Conector para sistema PJe com automação assistida"""

async def submit\_petition(self, petition: Petition, credentials: Credentials) -> SubmissionResult:

# Automação com Playwright para preenchimento assistido

browser = await self.get\_authenticated\_browser(credentials)

# Preenchimento de formulários

result = await self.\_fill\_petition\_form(browser, petition)

# Confirmação humana obrigatória antes do envio

if not petition.human\_confirmation:

return SubmissionResult(

status="awaiting\_confirmation",

message="Revisão humana necessária antes do protocolo"

)

return await self.\_submit\_with\_monitoring(browser, petition)

**5.2 Integração com Escritório Digital**

**ERP Jurídico Integration**

interface ERPIntegration {

// Integração com sistemas como Astrea, Themis, SAJ ADV

sync\_client\_data(clients: Client[]): Promise<SyncResult>;

sync\_financial\_data(billing: BillingData[]): Promise<SyncResult>;

sync\_calendar\_events(events: CalendarEvent[]): Promise<SyncResult>;

// Webhooks para sincronização bidirecional

handle\_erp\_webhook(event: ERPWebhookEvent): Promise<void>;

}

class UniversalERPConnector {

private adapters: Map<string, ERPAdapter> = new Map();

constructor() {

this.adapters.set('astrea', new AstreaAdapter());

this.adapters.set('themis', new ThemisAdapter());

this.adapters.set('saj', new SAJAdapter());

}

async syncWithERP(erpType: string, syncConfig: SyncConfig): Promise<SyncResult> {

const adapter = this.adapters.get(erpType);

if (!adapter) throw new Error(`ERP ${erpType} not supported`);

return await adapter.performSync(syncConfig);

}

}

**6. Performance e Escalabilidade**

**6.1 Otimizações de Performance**

**Caching Strategy**

class LegalCacheManager:

def \_\_init\_\_(self):

self.redis = Redis()

self.local\_cache = TTLCache(maxsize=10000, ttl=3600)

async def get\_jurisprudence(self, query\_hash: str) -> Optional[SearchResult]:

# L1: Cache local (mais rápido)

local\_result = self.local\_cache.get(query\_hash)

if local\_result:

return local\_result

# L2: Redis (compartilhado)

redis\_result = await self.redis.get(f"jurisprudence:{query\_hash}")

if redis\_result:

cached\_result = pickle.loads(redis\_result)

self.local\_cache[query\_hash] = cached\_result

return cached\_result

return None

**Database Optimization**

-- Índices especializados para consultas jurídicas

CREATE INDEX idx\_jurisprudence\_court\_date ON jurisprudence(court, decision\_date DESC);

CREATE INDEX idx\_legislation\_article\_gin ON legislation USING gin(to\_tsvector('portuguese', article\_content));

CREATE INDEX idx\_cases\_client\_status ON cases(client\_id, status) WHERE status IN ('active', 'pending');

-- Particionamento por ano para tabelas grandes

CREATE TABLE jurisprudence\_2024 PARTITION OF jurisprudence

FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');

**6.2 Arquitetura para Escala**

**Event-Driven Architecture**

class EventBus:

def \_\_init\_\_(self):

self.kafka = KafkaProducer()

self.handlers = defaultdict(list)

async def publish(self, event: Event):

await self.kafka.send(event.topic, event.to\_json())

def subscribe(self, event\_type: str, handler: Callable):

self.handlers[event\_type].append(handler)

# Eventos do sistema

class DocumentGeneratedEvent(Event):

document\_id: str

user\_id: str

document\_type: str

ai\_confidence: float

class DeadlineCalculatedEvent(Event):

case\_id: str

deadline\_date: datetime

deadline\_type: str

urgency\_level: str

**7. Monitoramento e Observabilidade**

**7.1 Health Monitoring**

class SystemHealthMonitor:

def \_\_init\_\_(self):

self.prometheus = PrometheusClient()

self.jaeger = JaegerTracer()

@trace

async def check\_ai\_model\_health(self) -> HealthStatus:

try:

# Teste de geração simples

test\_response = await self.ai\_client.generate\_simple\_text(

"Teste de saúde do modelo",

max\_tokens=10

)

latency = self.measure\_response\_time()

self.prometheus.histogram('ai\_model\_latency').observe(latency)

self.prometheus.counter('ai\_model\_health\_checks').inc()

return HealthStatus(

status='healthy',

latency=latency,

model\_version=self.get\_model\_version()

)

except Exception as e:

self.prometheus.counter('ai\_model\_failures').inc()

return HealthStatus(status='unhealthy', error=str(e))

**7.2 Business Intelligence**

class LegalAnalytics:

def generate\_case\_insights(self, user\_id: str, period: timedelta) -> CaseInsights:

analytics = self.query\_analytics\_db(user\_id, period)

return CaseInsights(

total\_cases=analytics.case\_count,

documents\_generated=analytics.document\_count,

ai\_assistance\_usage=analytics.ai\_usage\_rate,

most\_used\_templates=analytics.popular\_templates,

deadline\_compliance\_rate=analytics.deadline\_success\_rate,

average\_case\_resolution\_time=analytics.avg\_resolution\_time,

predictive\_insights={

'workload\_forecast': self.predict\_workload(user\_id),

'deadline\_risk\_cases': self.identify\_risky\_deadlines(user\_id),

'efficiency\_recommendations': self.suggest\_improvements(analytics)

}

)

**8. Deployment e DevOps**

**8.1 Container Strategy**

# Dockerfile para microserviços

FROM node:18-alpine AS builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci --only=production

FROM node:18-alpine AS runtime

RUN addgroup -g 1001 -S nodejs

RUN adduser -S nextjs -u 1001

WORKDIR /app

COPY --from=builder --chown=nextjs:nodejs /app/node\_modules ./node\_modules

COPY --chown=nextjs:nodejs . .

USER nextjs

EXPOSE 3000

CMD ["npm", "start"]

**8.2 CI/CD Pipeline**

# .github/workflows/deploy.yml

name: Deploy Legal AI System

on:

push:

branches: [main]

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Run AI Model Tests

run: |

python -m pytest tests/ai/ --cov=src/ai

python scripts/validate\_legal\_citations.py

security-scan:

runs-on: ubuntu-latest

steps:

- name: OWASP Dependency Check

run: dependency-check --project "Legal-AI" --scan .

deploy:

needs: [test, security-scan]

runs-on: ubuntu-latest

steps:

- name: Deploy to Kubernetes

run: |

kubectl apply -f k8s/

kubectl rollout status deployment/legal-ai-api

**9. Considerações Finais de Arquitetura**

**9.1 Princípios de Design**

1. **Fail-Safe**: Sistema sempre falha para posição segura (não gerar se houver dúvida)
2. **Transparency-First**: Toda ação da IA é auditável e explicável
3. **Human-Centric**: IA como assistente, nunca como substituto do advogado
4. **Performance-Conscious**: Sub-segundo para consultas, < 30s para geração de documentos
5. **Privacy-by-Design**: Dados sensíveis nunca deixam controle do usuário sem consentimento explícito

**9.2 Roadmap de Evolução**

**Fase 1 (MVP - 6 meses)**

* Core SaaS platform
* RAG básico com verificação de citações
* Templates essenciais
* Calculadora de prazos

**Fase 2 (12 meses)**

* Desktop agent para IA local
* Conectores básicos de tribunais
* Analytics avançado
* Mobile app

**Fase 3 (18 meses)**

* IA preditiva com explicabilidade
* Automação processual assistida
* Integração completa com ERPs
* Blockchain para auditoria

Esta arquitetura garante que o sistema seja não apenas tecnicamente robusto, mas principalmente confiável e adequado às exigências rigorosas da prática jurídica brasileira.