# Arquitetura RAG para Assistente Interno Empresarial

**Teste Teórico**

## Contexto:

Projetar um sistema que transforme milhares de PDFs, páginas wiki e tickets históricos em um assistente interno inteligente. O objetivo é criar uma solução RAG (Retrieval-Augmented Generation) escalável que domine o conhecimento técnico organizacional.

## 1. Captura, Fatiamento e Indexação de Material

## Pipeline de Ingestão

**Fontes de Dados:**

- PDFs técnicos (manuais, especificações, relatórios)
- Wiki interno (Confluence, Notion, SharePoint)
- Tickets históricos (Jira, ServiceNow, Azure DevOps)
- Documentação de código (GitHub, GitLab)

**Processo de Extração:**

```
# Exemplo simplificado do pipeline
class DocumentProcessor:
    def __init__(self):
        self.pdf_parser = PyPDF2  # Para PDFs simples
        self.layout_parser = DocumentAI  # Para PDFs complexos
        self.ocr_engine = Tesseract  # Para documentos escaneados

    def extract_content(self, document):
        # 1. Detecta tipo de documento
        # 2. Aplica extração apropriada
        # 3. Limpa e normaliza o texto
        # 4. Extrai metadados (autor, data, categoria)
        pass
```

**Estratégia de Chunking:**

- **Chunking Semântico:** 512-1024 tokens por seção lógica
- **Overlap:** 20% entre chunks para manter contexto

- **Hierárquico:** Preserva estrutura (capítulo → seção → parágrafo)
- **Adaptativo:** Tamanhos variáveis por tipo de conteúdo

### Banco Vetorial

**Tecnologia:** Qdrant (produção) ou Chroma (desenvolvimento)

- **Dimensionalidade:** 1536 (text-embedding-3-large) ou 768 (modelos open-source por OLLAMA)
- **Indexação:** HNSW para busca eficiente
- **Similaridade:** Cosine similarity
- **Particionamento:** Collections por departamento/domínio

## 2. Modelos de Embeddings e LLM

### Embedding Model

**Escolha:** `text-embedding-3-large`

**Justificativa:**

- Alta qualidade semântica para conteúdo técnico
- 1536 dimensões balanceiam precisão e performance
- Integração nativa com GPT

### Large Language Model

**Escolha:** GPT-4 ou Claude-3.5 Sonnet

**Justificativa:**

- Context window de 128K tokens para documentos longos
- Baixa taxa de alucinação
- Capacidades de raciocínio complexo
- Suporte a código e documentação técnica

## 3. Fluxo Arquitetural: Do Ingest ao Usuário Final

## Diagrama Mermaid

```
flowchart TD
    A[Usuário] --> B[FastAPI Middleware]
    B --> C[Auth & Rate Limiting]
    C --> D[Query Processor]
    D --> E[Vector Search]
    E --> F[Embeddings Engine]
    F --> G[Vector Database]
    D --> H[LLM Integration]
    E --> H
    H --> I[Response Generator]
    I --> J[User Interface]

    subgraph "Data Ingestion"
      K[PDFs/Wiki/Tickets] --> L[Content Extraction]
      L --> M[Smart Chunking]
      M --> F
    end

    subgraph "Monitoring"
      N[Metrics Collection]
      O[Performance Tracking]
      P[Cost Optimization]
    end

    style H fill:#e6f7ff,stroke:#1890ff,stroke-width:2px
    style F fill:#f6ffed,stroke:#52c41a,stroke-width:2px
    style B fill:#fffbe6,stroke:#faad14,stroke-width:2px
```

## Componentes Principais

### 1. FastAPI Middleware:

```
# Estrutura modular
├── auth_middleware.py      # OAuth2
├── rate_limiter.py         # Throttling
├── query_router.py
├── context_manager.py      # Sessão
└── response_formatter.py   # Formatação
```

### 2. Query Processing Engine:

- Classificação de intenção
- Busca híbrida (vetorial + keyword)
- Re-ranking com cross-encoder
- Prompt engineering dinâmico

**3. Response Generation:**

- Assembly de contexto relevante
- Sistema automático de citações
- Validação de qualidade
- Feedback loop para melhoria

### Fluxo Detalhado

1. **Usuário** envia query via chat/API
2. **Middleware** autentica e aplica rate limiting
3. **Query Processor** analisa intenção e extrai entidades
4. **Vector Search** busca chunks relevantes no Qdrant
5. **LLM** recebe contexto + query e gera resposta
6. **Response Generator** formata com citações
7. **Interface** entrega resposta ao usuário

## 4. Métricas, Custos e Controles

### Monitoramento de Performance

**Métricas Críticas:**

- **Latência P95:** < 5 segundos
- **Time-to-First-Token:** < 500ms
- **Retrieval Precision@5:** > 85%
- **User Satisfaction (NPS):** > 8.0

### Otimização de Custos

**Estratégias Implementadas:**

- **Embedding Caching:** Cache para documentos estáticos
- **Query Caching:** Respostas para FAQs
- **Batch Processing:** Ingestão em lotes
- **Tiered Storage:** Por frequência de acesso

### Segurança e Controle de Acesso

**Implementação Multi-Camadas:**

**1. Autenticação:**

# RBAC com OAuth2

**2. Segurança de Dados:**

- Encryption at rest (AES-256)
- TLS 1.3 para comunicação

**3. Compliance:**

- Query filtering por permissões
- Incident response automatizado

## Conclusão

Esta arquitetura RAG fornece uma base para um assistente interno empresarial, priorizando:

- **Escalabilidade:** Componentes modulares e auto-scaling
- **Segurança:** Multi-layer security com RBAC granular
- **Performance:** Cache inteligente e otimizações de latência
- **Observabilidade:** Monitoramento completo e alertas proativos
- **Custo-efetividade:** Otimizações que reduzem OpEx

A implementação permite evolução incremental e adaptação às necessidades específicas da organização, enquanto mantém conformidade com padrões de segurança.