Arquitetura Técnica - Sistema de Atendimento ao Cliente com IA

Visão Geral

Este documento descreve a arquitetura técnica do sistema de atendimento ao cliente automatizado, detalhando componentes, integrações e fluxos de dados.

Componentes

1. Frontend

- **Dashboard Cliente**: Aplicação React/Next.js
  - Gerenciamento de agentes
  - Upload de documentos
  - Visualização de métricas
  - Gestão de créditos
- **Integrações de Canais**:
  - WhatsApp Business API (prioridade inicial)
  - Preparação para canais adicionais futuros

2. Backend

- **API Gateway** (Node.js/Express):
  - Autenticação e autorização
  - Roteamento de requisições
  - Rate limiting
  - Logging
- **Serviço de Orquestração** (Node.js/Express):
  - Gerenciamento de conversas
  - Delegação para serviços especializados
  - Monitoramento de créditos
- **Processador de Documentos** (Python/Flask com docling):
  - Extração de texto de diversos formatos (PDF, DOCX, TXT)
  - Normalização de conteúdo
  - Geração de embeddings para indexação
- **Engine IA** (Node.js):
  - Integração com APIs de LLM
  - Sistema RAG

    - Histórico de contexto
    - Geração de respostas
- **Serviço de Créditos** (Node.js/Express):
    - Controle de saldo
    - Registro de consumo
    - Integração com gateway de pagamentos

3. Armazenamento

- **Vector Database** (Pinecone/pgvector):
    - Armazenamento de embeddings
    - Busca semântica
- **Banco Relacional** (PostgreSQL):
    - Dados de clientes
    - Configurações
    - Registro de transações
    - Métricas e estatísticas
- **Object Storage** (S3):
    - Documentos originais
    - Backups

4. Infraestrutura

- **Deployment**: Docker/Kubernetes
- **CI/CD**: GitHub Actions
- **Observabilidade**: Prometheus/Grafana
- **Logs**: ELK Stack
- **Monitoramento**: Sentry

Fluxos Principais

1. Processamento de Documentos

mermaid

```
sequenceDiagram
    participant Cliente
    participant API Gateway
    participant Processador de Documentos
    participant Vector Database
    participant Banco Relacional

    Cliente->>API Gateway: Upload de documento
    API Gateway->>Processador de Documentos: Encaminha documento
    Processador de Documentos->>Processador de Documentos: Extrai texto usando docling
    Processador de Documentos->>Processador de Documentos: Divide em chunks
    Processador de Documentos->>Processador de Documentos: Gera embeddings
    Processador de Documentos->>Vector Database: Armazena embeddings
    Processador de Documentos->>Banco Relacional: Armazena metadados
    Processador de Documentos->>API Gateway: Confirma processamento
    API Gateway->>Cliente: Notifica conclusão
```

2. Atendimento ao Cliente

mermaid

```
sequenceDiagram
    participant Usuário
    participant WhatsApp API
    participant API Gateway
    participant Orquestrador
    participant Engine IA
    participant Vector Database
    participant Serviço de Créditos

    Usuário->>WhatsApp API: Envia mensagem
    WhatsApp API->>API Gateway: Encaminha mensagem
    API Gateway->>Orquestrador: Processa mensagem
    Orquestrador->>Serviço de Créditos: Verifica saldo
    Serviço de Créditos->>Orquestrador: Confirma disponibilidade
    Orquestrador->>Engine IA: Solicita resposta
    Engine IA->>Vector Database: Busca informações relevantes
    Vector Database->>Engine IA: Retorna contexto
    Engine IA->>Engine IA: Gera resposta com LLM
    Engine IA->>Orquestrador: Entrega resposta
    Orquestrador->>Serviço de Créditos: Registra consumo
    Orquestrador->>API Gateway: Encaminha resposta
    API Gateway->>WhatsApp API: Envia resposta
    WhatsApp API->>Usuário: Entrega resposta
```

Considerações de Escalabilidade

- Serviços stateless para facilitar escala horizontal
- Uso de cache (Redis) para reduzir latência
- Processamento assíncrono para tarefas demoradas
- Filas de mensagens para garantir processamento em caso de picos
- Arquitetura de microserviços para escalar componentes independentemente

Considerações de Segurança

- Autenticação JWT
- Encriptação de dados sensíveis
- HTTPS em todas as comunicações
- Isolamento de ambientes
- Auditoria de acessos
- Proteção contra ataques comuns (CSRF, XSS, SQL Injection)

Integração Específica com docling (Python)

```
mermaid

sequenceDiagram
    participant API Gateway (Node.js)
    participant Queue Service (RabbitMQ)
    participant Document Processor (Python/Flask)
    participant docling Library
    participant Storage

    API Gateway (Node.js)->>Queue Service (RabbitMQ): Envia job com documento
    Queue Service (RabbitMQ)->>Document Processor (Python/Flask): Consome job
    Document Processor (Python/Flask)->>docling Library: Processa documento
    docling Library->>docling Library: Extrai texto e metadados
    docling Library->>Document Processor (Python/Flask): Retorna conteúdo estruturado
    Document Processor (Python/Flask)->>Storage: Armazena resultado
    Document Processor (Python/Flask)->>Queue Service (RabbitMQ): Notifica conclusão
    Queue Service (RabbitMQ)->>API Gateway (Node.js): Atualiza status
```

Componentes Específicos para docling

- **Serviço Python Dedicado**: Microserviço em Python/Flask para encapsular o processamento docling
- **Fila de Mensagens**: RabbitMQ para comunicação assíncrona entre Node.js e Python
- **API REST**: Interface para comunicação síncrona quando necessário

- **Monitoramento Específico**: Métricas de performance e qualidade de extração

Requisitos de Infraestrutura

Para MVP

- **Frontend**: Vercel ou similar (plano inicial)
- **Backend Node.js**: AWS EC2 t3.medium (2 instâncias) ou equivalente
- **Serviço Python**: AWS EC2 t3.medium (1 instância) ou equivalente
- **Banco de Dados**: PostgreSQL (AWS RDS t3.micro) ou equivalente
- **Vector Store**: Pinecone (plano inicial) ou pgvector em PostgreSQL
- **Object Storage**: AWS S3 (plano básico)
- **Cache**: Redis (AWS ElastiCache t2.micro) ou equivalente
- **Fila de Mensagens**: RabbitMQ (AWS t3.micro)

Para Escala (base de 50+ clientes)

- Autoscaling para todos os componentes
- Balanceadores de carga
- Réplicas de banco de dados
- Clusters para cache e filas

Considerações para Desenvolvimento

Padrões de Comunicação

- **Síncrona**: REST para operações que exigem resposta imediata
- **Assíncrona**: Mensageria para operações de longa duração

Estrutura de Código

- **Monorepo**: Facilita coordenação entre serviços
- **Clean Architecture**: Separação clara de responsabilidades
- **Dependency Injection**: Facilita testes e manutenção

Estratégia de Testes

- **Unitários**: Para lógica de negócio isolada
- **Integração**: Para comunicação entre serviços
- **End-to-End**: Para fluxos completos
- **Mocks**: Para APIs externas e LLMs

Evoluções Futuras

- Implementação de LLMs próprios para casos específicos
- Adição de novos canais de comunicação
- Expansão para funcionalidades de SDR e vendas
- Implementação de análise avançada de conversas
- Aprendizado contínuo com base em feedback

Glossário

- **RAG**: Retrieval Augmented Generation
- **LLM**: Large Language Model
- **SDR**: Sales Development Representative
- **Vector Database**: Banco de dados otimizado para busca semântica
- **Embedding**: Representação vetorial de texto para processamento semântico