

## ARQUITETURA COMPLETA ERP-UZZAI v2.0

### Documento Técnico Consolidado

Este documento consolida TODA a arquitetura técnica do ERP-UzzAI, um sistema de automação empresarial com IA integrada para processamento de reuniões, extração de entidades e gestão de projetos.

**Visão:** "Vault.exe com IA Integrada" — Sistema que automatiza o trabalho operacional de gestão de projetos mantendo o Obsidian como interface principal.

### ÍNDICE

1. Visão Geral do Sistema
2. Arquitetura de Alto Nível
3. Modelo de Domínio
4. Arquitetura Multi-Agente
5. Pipeline de Processamento
6. Sistema RAG
7. Banco de Dados
8. API REST
9. Integração Financeira
10. Stack Tecnológico
11. Estrutura de Código
12. Roadmap de Implementação

## 1. VISÃO GERAL DO SISTEMA

### 1.1 Propósito

O ERP-UzzAI é um sistema de automação empresarial que:

- Processa transcrições de reuniões automaticamente
- Extrai entidades (decisões, ações, riscos, kaizens) via IA
- Gera documentação estruturada (atas, dashboards)
- Gerencia projetos, sprints e pessoas
- Integra com Obsidian Vault existente
- Utiliza RAG para contexto histórico e anti-duplicação

### 1.2 Problema Resolvido

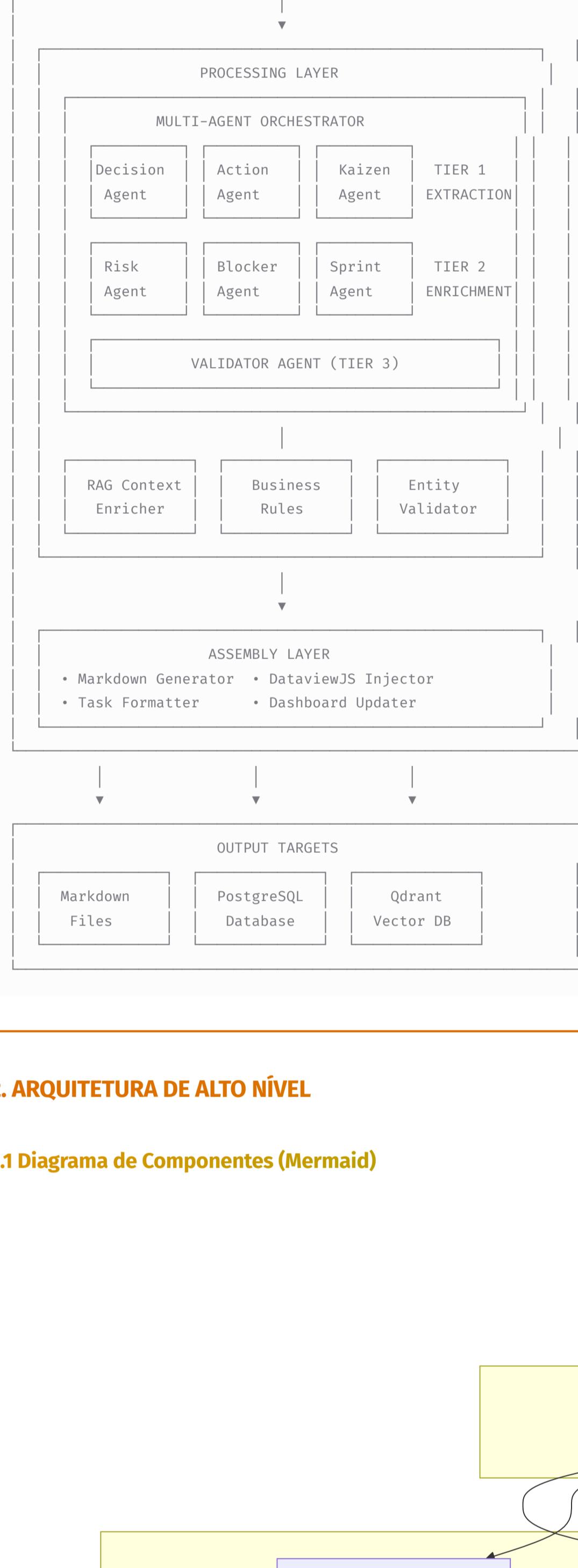
#### ANTES (Manual):

Reunião de 90 min → 4-6 horas de documentação manual  
Decisões duplicadas entre reuniões  
Ações perdidas ou não atribuídas  
Dashboards desatualizados

#### DEPOIS (Automatizado):

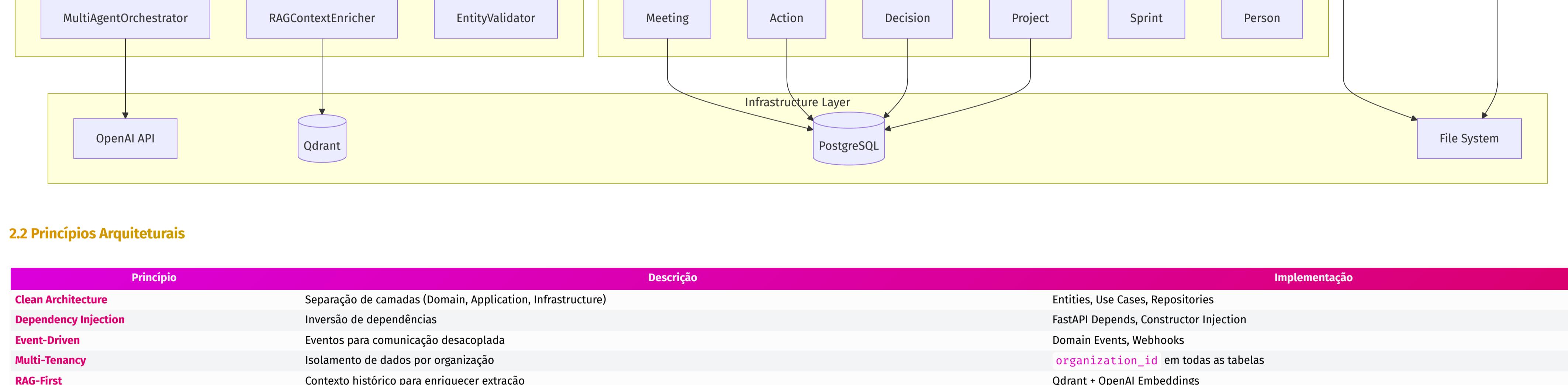
Reunião de 90 min → 5 min de processamento automático  
Detecção de decisões duplicadas via RAG  
Ações atribuídas com prazos e responsáveis  
Dashboards atualizados automaticamente

### 1.3 Arquitetura Conceitual



## 2. ARQUITETURA DE ALTO NÍVEL

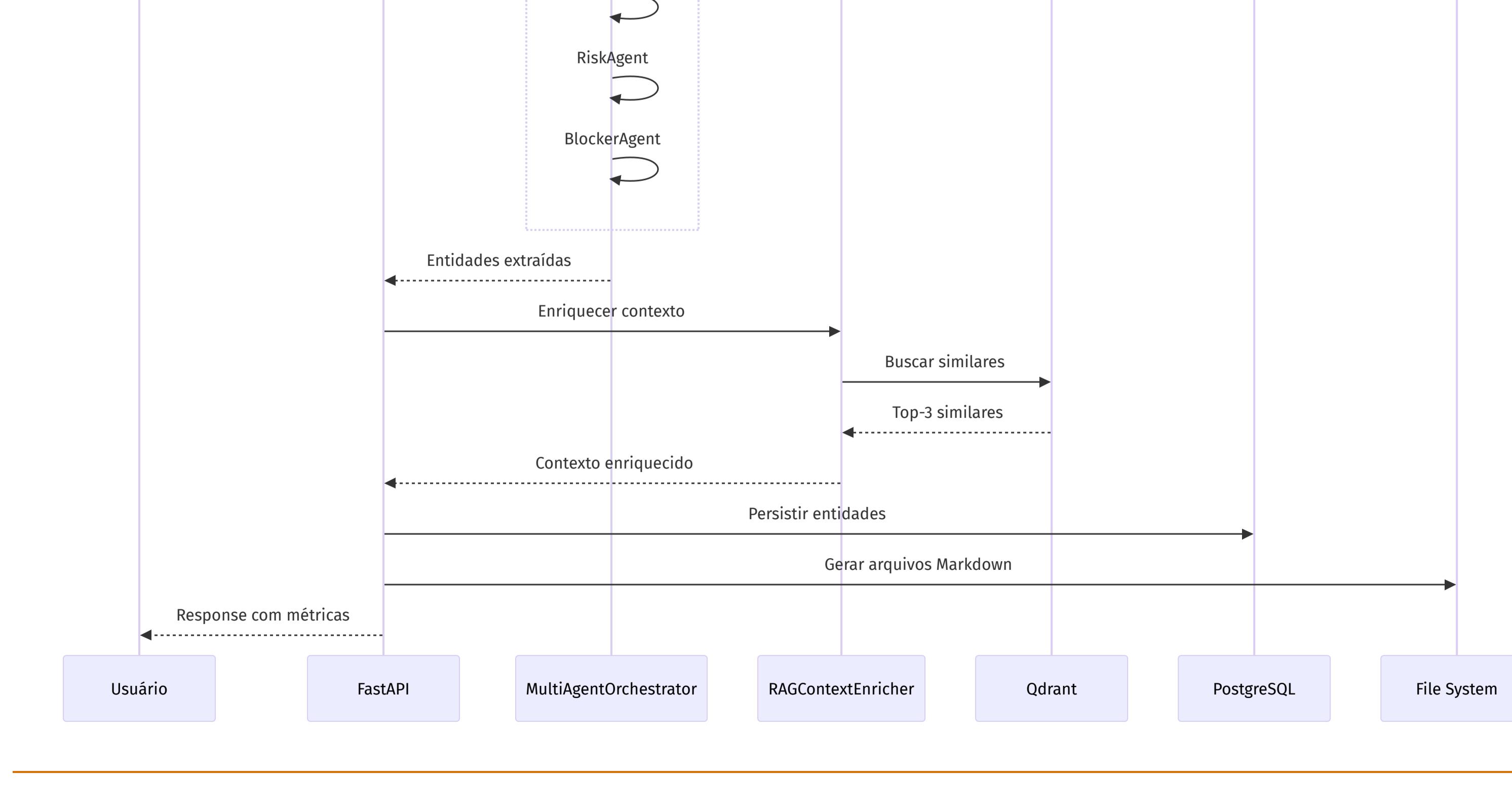
### 2.1 Diagrama de Componentes (Mermaid)



### 2.2 Princípios Arquiteturais

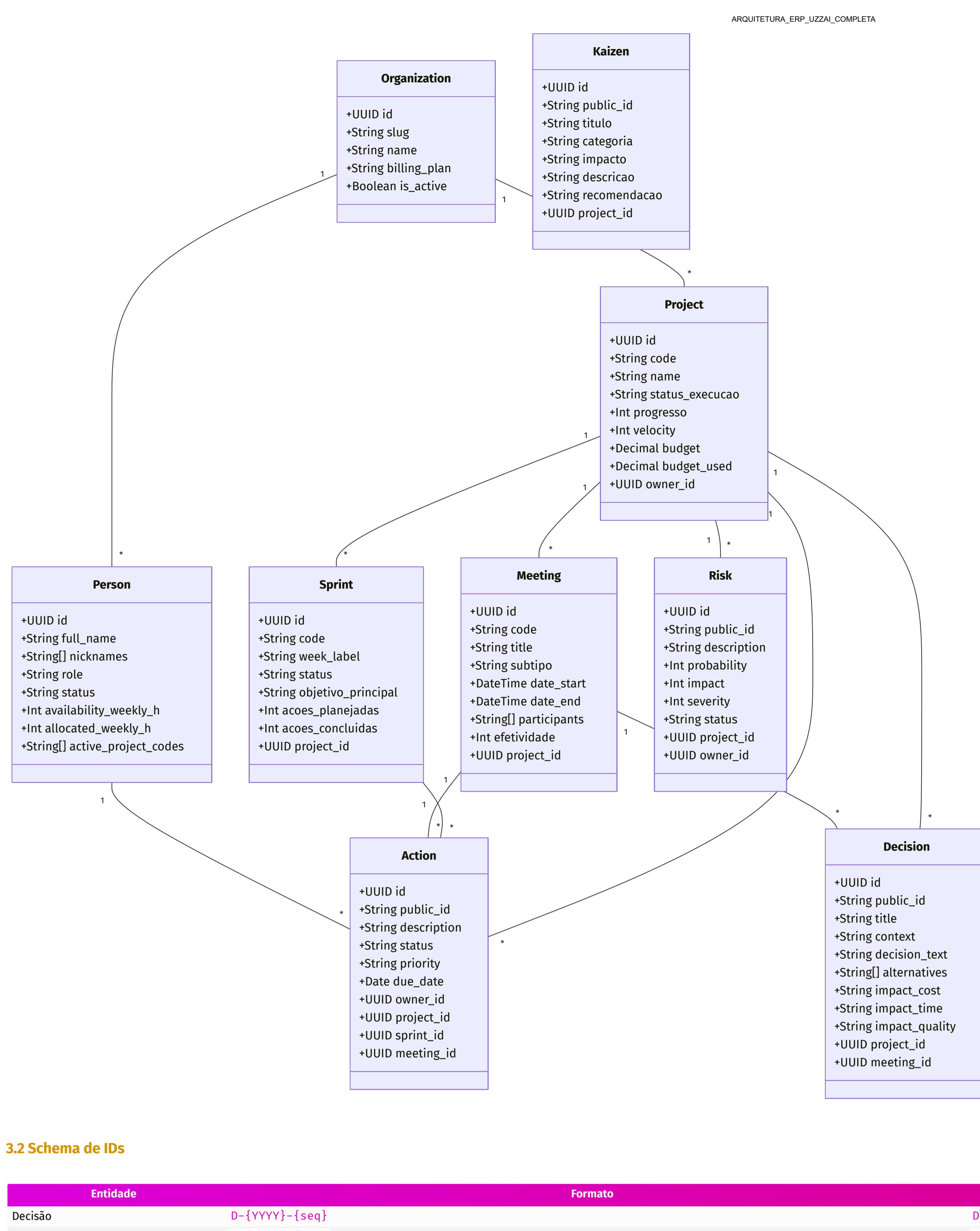
Princípio	Descrição	Implementação
Clean Architecture	Separação de camadas (Domain, Application, Infrastructure)	Entities, Use Cases, Repositories
Dependency Injection	Inversão de dependências	FastAPI Depends, Constructor Injection
Event-Driven	Eventos para comunicação desacoplada	Domain Events, Webhooks
Multi-Tenancy	Isolamento de dados por organização	<code>organization_id</code> em todas as tabelas
RAG-First	Contexto histórico para enriquecer extração	Qdrant + OpenAI Embeddings

### 2.3 Fluxo de Dados Principal



## 3. MODELO DE DOMÍNIO

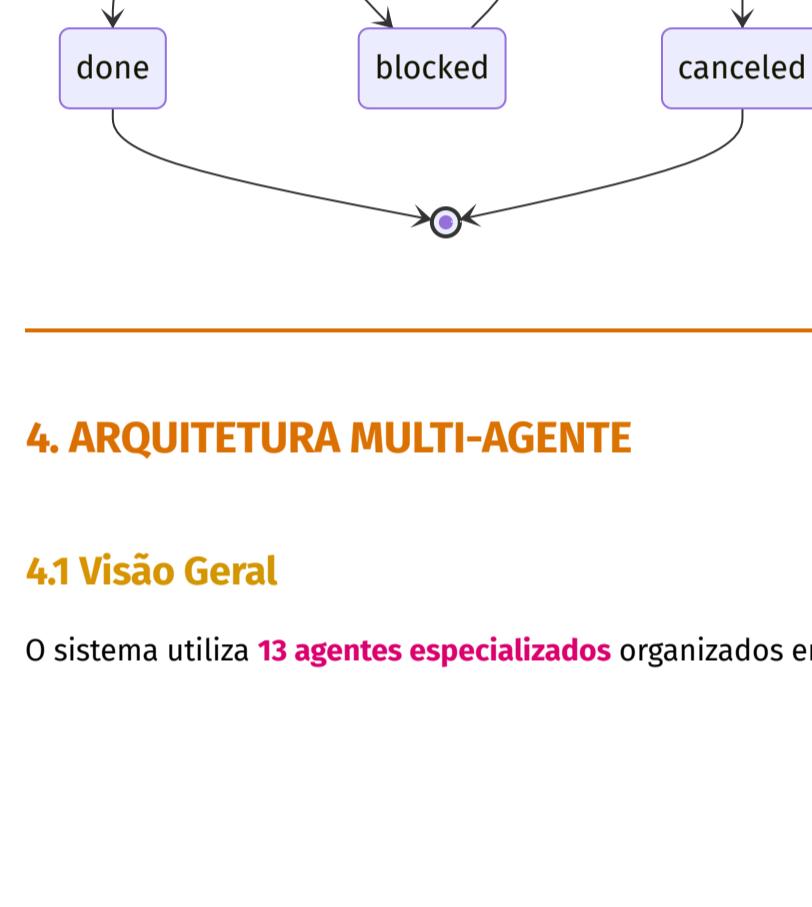
### 3.1 Entidades Principais



### 3.2 Schema de IDs

Entidade	Formato	Exemplo
Decisão	D-{YYYY}-{seq}	D-2025-042
Ação	A-{YYYY}-{seq}	A-2025-123
Kaizen	K-{tip[0]}-{seq}	K-T-015
Bloqueio	B-{projeto}-{seq}	B-CHATBOT-003
Risco	R-{projeto}-{seq}	R-UZZAI-007
Meeting	MTG-{YYYY-MM-DD}-{projeto}	MTG-2025-11-24-CHATBOT
Sprint	Sprint-{YYYY}-{W{nn}}	Sprint-2025-W48

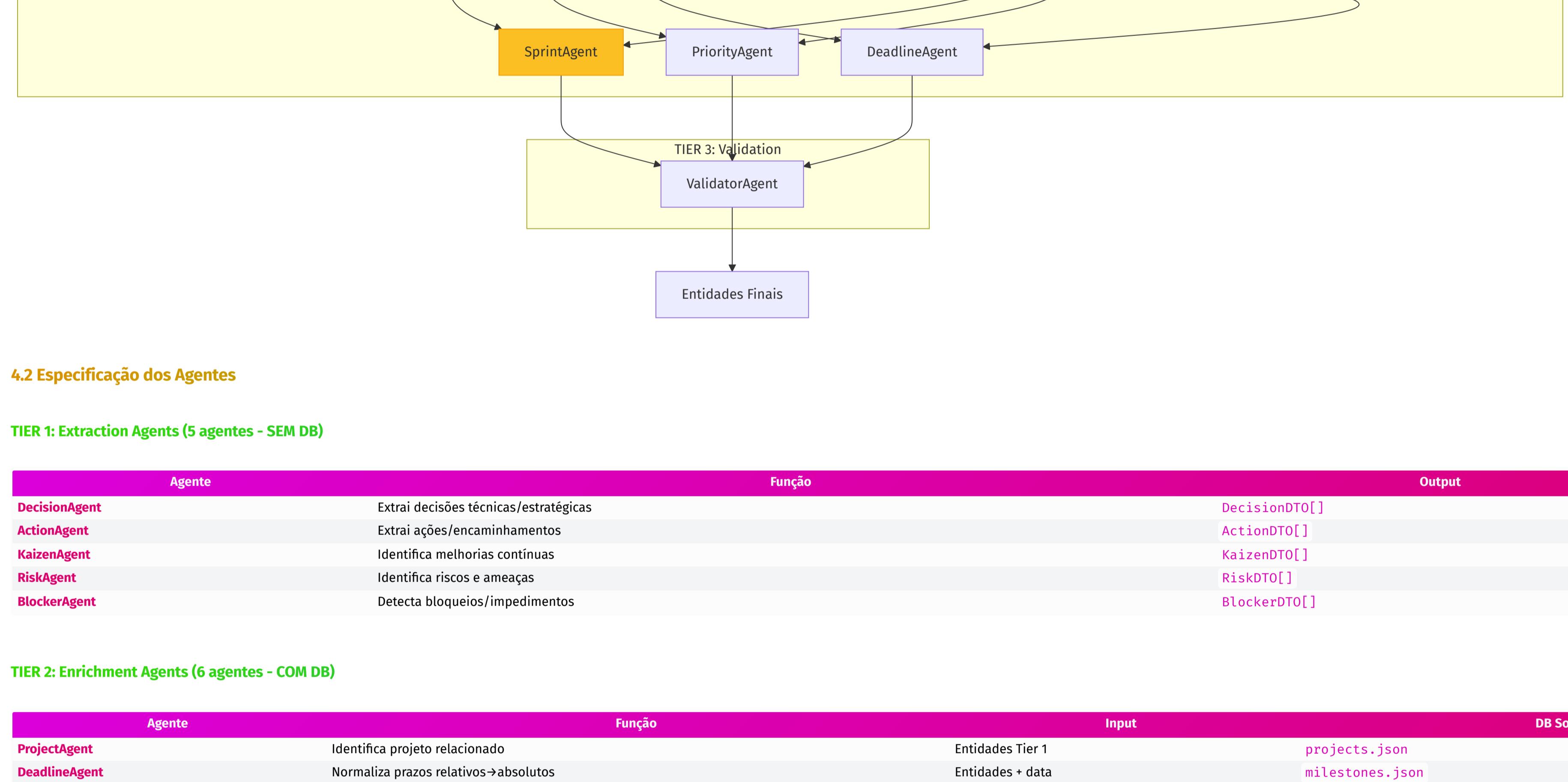
### 3.3 Estados de Entidades



## 4. ARQUITETURA MULTI-AGENTE

### 4.1 Visão Geral

O sistema utiliza **13 agentes especializados** organizados em **3 tiers** para processamento de transcrições:



### 4.2 Especificação dos Agentes

#### TIER 1: Extraction Agents (5 agentes - SEM DB)

Agente	Função	Output	DB
DecisionAgent	Extrai decisões técnicas/estratégicas	DecisionDTO[]	X
ActionAgent	Extrai ações/encaminhamentos	ActionDTO[]	X
KaizenAgent	Identifica melhorias contínuas	KaizenDTO[]	X
RiskAgent	Identifica riscos e ameaças	RiskDTO[]	X
BlockerAgent	Detecta bloqueios/impedimentos	BlockerDTO[]	X

#### TIER 2: Enrichment Agents (6 agentes - COM DB)

Agente	Função	Input	DB Source
ProjectAgent	Identifica projeto relacionado	Entidades Tier 1	projects.json
DeadlineAgent	Normaliza prazos relativos→absolutos	Entidades + data	milestones.json
PriorityAgent	Infere prioridade (high/medium/low)	Entidades + contexto	business_context.json
SprintAgent	Identifica sprint semanal	Entidades + data	sprints.json
PerformanceAgent	Análise desempenho individual/equipe	Entidades + pessoas	performance_metrics.json
TeamHealthAgent	Avalia saúde da equipe	Entidades + contexto	workload.json

#### TIER 3: Validation Agent

Agente	Função	Input	Output
ValidatorAgent	Dedupe e valida entidades	Entidades enriquecidas	Entidades finais validadas

### 4.3 Exemplo: ActionAgent

```

class ActionAgent(BaseSpecializedAgent):
    """Agente especializado em extração de ações."""

    SYSTEM_PROMPT = """
    Você é um especialista em extração de ações/encaminhamentos de transcrições.

    LISTA DE PESSOAS VÁLIDAS:
    {person_aliases.json}

    PROJETOS VÁLIDOS:
    {valid_projects.json}

    REGRAS:
    1. Uma ação DEVE ter: descrição, responsável, prazo
    2. Responsável DEVE estar na lista de pessoas
    3. Prazo DEVE ser convertido para data absoluta
    4. Prioridade: urgente/criticó=high, importante=medium, padrão=low

    OUTPUT FORMAT (JSON):
    {
        "actions": [
            {
                "description": "Descrição clara da ação",
                "owner": "Nome Completo",
                "due_date": "YYYY-MM-DD",
                "priority": "high|medium|low",
                "project": "CÓDIGO_PROJETO"
            }
        ]
    }
    """

    async def extract(self, transcript: str, context: Dict) -> List[ActionDTO]:
        prompt = self.SYSTEM_PROMPT.format(
            person_aliases=json.dumps(context["persons"]),
            valid_projects=json.dumps(context["projects"])
        )

        response = await self.llm.generate(
            system=prompt,
            user=transcript,
            response_format={"type": "json_object"}
        )

        return [ActionDTO(**a) for a in response["actions"]]
  
```

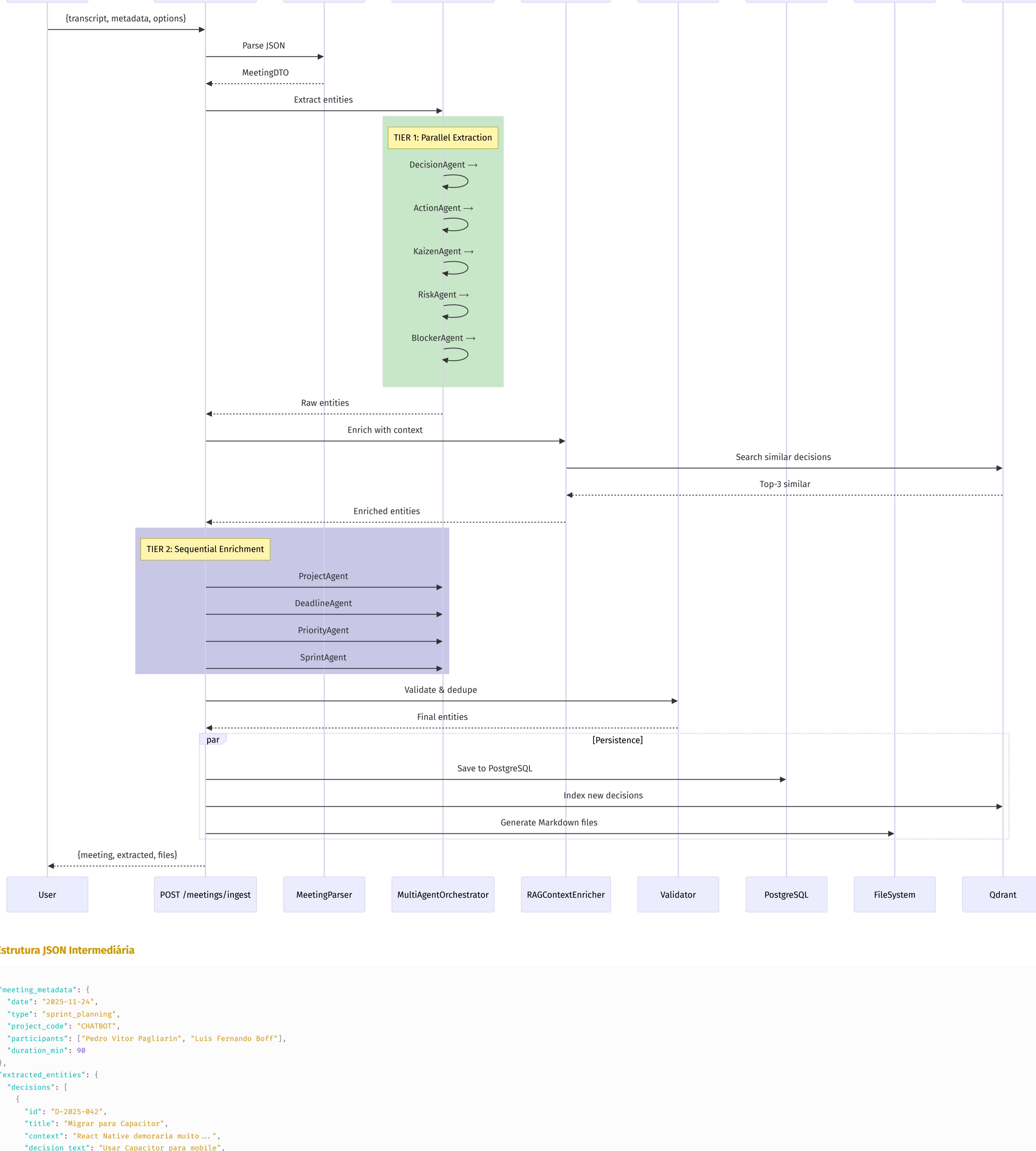
### 4.4 Métricas de Sucesso por Tier

Tier	Métrica	Target
Tier 1	Recall vs ata manual	≥ 85%
Tier 1	Precision	≥ 80%
Tier 1	Tempo por agente	≤ 60s
Tier 2	Project Accuracy	≥ 90%
Tier 2	Deadline Accuracy	≥ 85%
Tier 2	Priority Accuracy	≥ 75%

Tier	Métrica	Target
Tier 2	Sprint Accuracy	≥ 95%
Tier 3	Deduplication Rate	≥ 95%
Tier 3	Validation Pass Rate	≥ 90%

## 5. PIPELINE DE PROCESSAMENTO

### 5.1 Fluxo Completo de Ingestão



### 5.2 Estrutura JSON Intermediária

```
{
  "meeting_metadata": {
    "date": "2025-11-24",
    "type": "sprint_planning",
    "project_code": "CHATBOT",
    "participants": ["Pedro Vitor Pagliarin", "Luis Fernando Boff"],
    "duration_min": 90
  },
  "extracted_entities": [
    {
      "decisions": [
        {
          "id": "D-2025-042",
          "title": "Migrar para Capacitor",
          "context": "React Native demoraria muito...",
          "decision_text": "Usar Capacitor para mobile",
          "alternatives": ["React Native", "Flutter"],
          "impact": {"custo": "Alto", "prazo": "+2 dias", "qualidade": "Melhora"},
          "project": "CHATBOT",
          "similar_decisions": [],
          "requires_review": false
        }
      ],
      "actions": [
        {
          "id": "A-2025-123",
          "description": "Implementar login OAuth",
          "owner": "[Luis Fernando Boff]",
          "due_date": "2025-12-01",
          "priority": "high",
          "project": "CHATBOT",
          "sprint": "Sprint-2025-W48",
          "tags": ["encaminhamento", "técnico"]
        }
      ],
      "kaizens": [
        {
          "id": "K-T-015",
          "titulo": "Commits pequenos evitam conflitos",
          "categoria": "technical",
          "impacto": "Alto",
          "recomendacao": "Adotar feature branches"
        }
      ],
      "risks": [
        {
          "id": "R-CHATBOT-003",
          "description": "LOI SEDETEC atrasada",
          "probability": 4,
          "impact": 5,
          "severity": 20,
          "owner": "Pedro Vitor Pagliarin"
        }
      ],
      "blockers": [
        {
          "id": "B-CHATBOT-001",
          "titulo": "Falta de números WhatsApp",
          "severidade": "Crítico",
          "bloqueado_por": "Sócios não forneceram números",
          "bloqueando": "Lançamento piloto"
        }
      ]
    },
    "rag_context": {
      "similar_decisions_found": 1,
      "duplicate_alerts": false
    }
  }
}
```

## 6. SISTEMA RAG

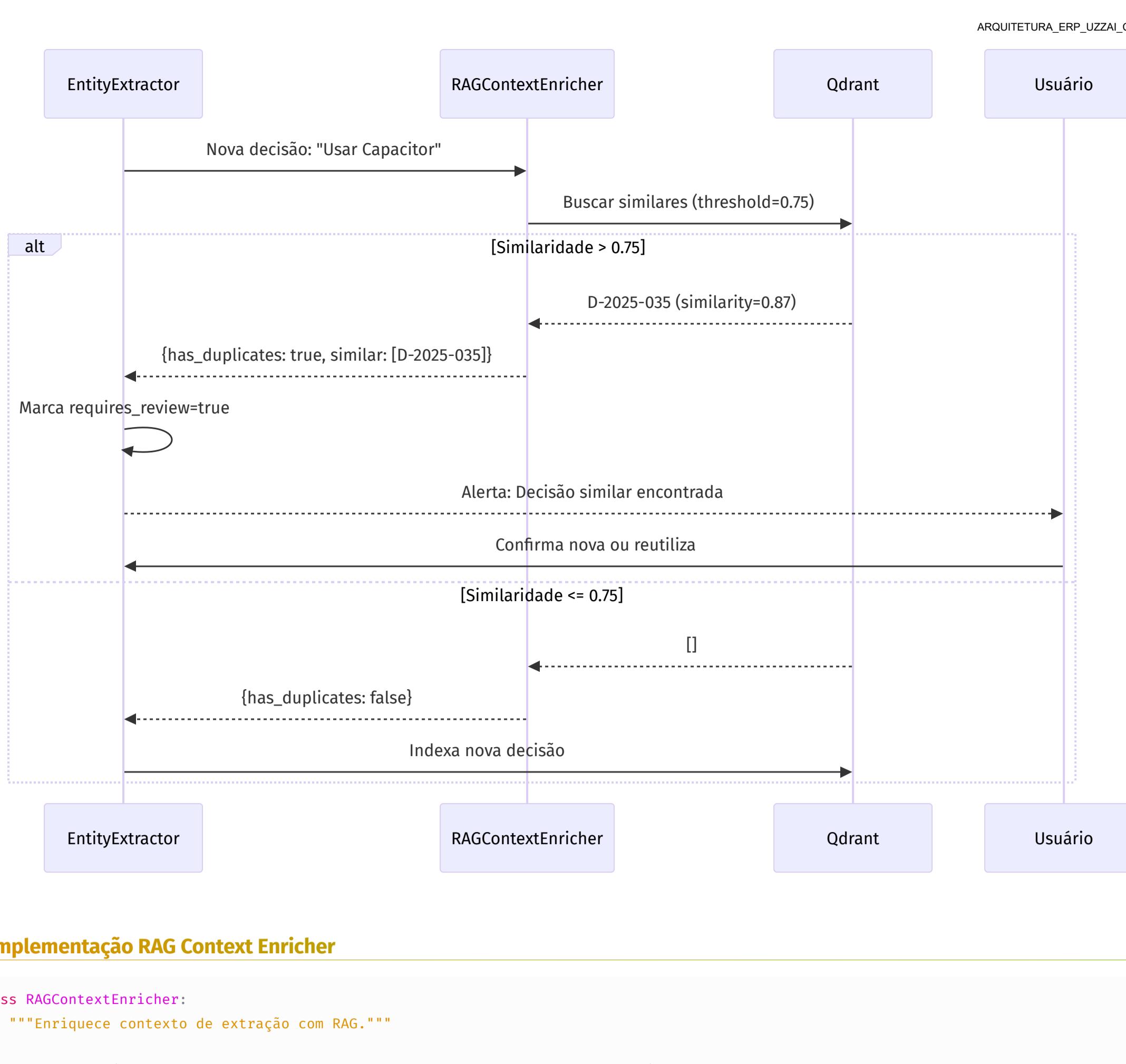
### 6.1 Arquitetura RAG



### 6.2 Estrutura de Coleções Qdrant

```
COLLECTIONS = {
  "decisions": {
    "vector_size": 3072, # text-embedding-3-large
    "payload_schema": {
      "decision_id": "STRING", # D-2025-XXX
      "project_code": "STRING", # CHATBOT, SITE-BUILDER
      "title": "STRING",
      "context": "STRING",
      "status": "STRING", # active, archived
      "created_at": "DATETIME",
      "organization_id": "STRING"
    }
  },
  "actions": {
    "vector_size": 3072,
    "payload_schema": {
      "action_id": "STRING", # A-2025-XXX
      "project_code": "STRING",
      "description": "STRING",
      "status": "STRING", # open, completed, canceled
      "success_score": "INTEGER", # 1-10
      "owner": "STRING",
      "created_at": "DATETIME",
      "organization_id": "STRING"
    }
  },
  "kaizens": {
    "vector_size": 3072,
    "payload_schema": {
      "kaizen_id": "STRING", # K-T-XXX
      "project_code": "STRING",
      "lesson": "STRING",
      "category": "STRING",
      "roi_impact": "STRING", # CRÍTICO, ALTO, MÉDIO
      "created_at": "DATETIME",
      "organization_id": "STRING"
    }
  }
}
```

### 6.3 Fluxo de Anti-Duplicação



#### 6.4 Implementação RAG Context Enricher

```

class RAGContextEnricher:
    """Enriquece contexto de extração com RAG."""

    def __init__(self, qdrant: QdrantClientWrapper, embedding: EmbeddingService):
        self.qdrant = qdrant
        self.embedding = embedding

    async def enrich_decision_context(
        self,
        decision_text: str,
        project_code: str,
        organization_id: UUID
    ) -> Dict:
        """Busca decisões similares para evitar duplicação."""
        # 1. Gera embedding da nova decisão
        query_embedding = self.embedding.embed_text(decision_text)

        # 2. Busca decisões similares
        similar = self.qdrant.search_similar_decisions(
            query_embedding=query_embedding,
            project_code=project_code,
            organization_id=str(organization_id),
            limit=3,
            score_threshold=0.75
        )

        return {
            "similar_decisions": similar,
            "has_duplicates": len(similar) > 0
        }

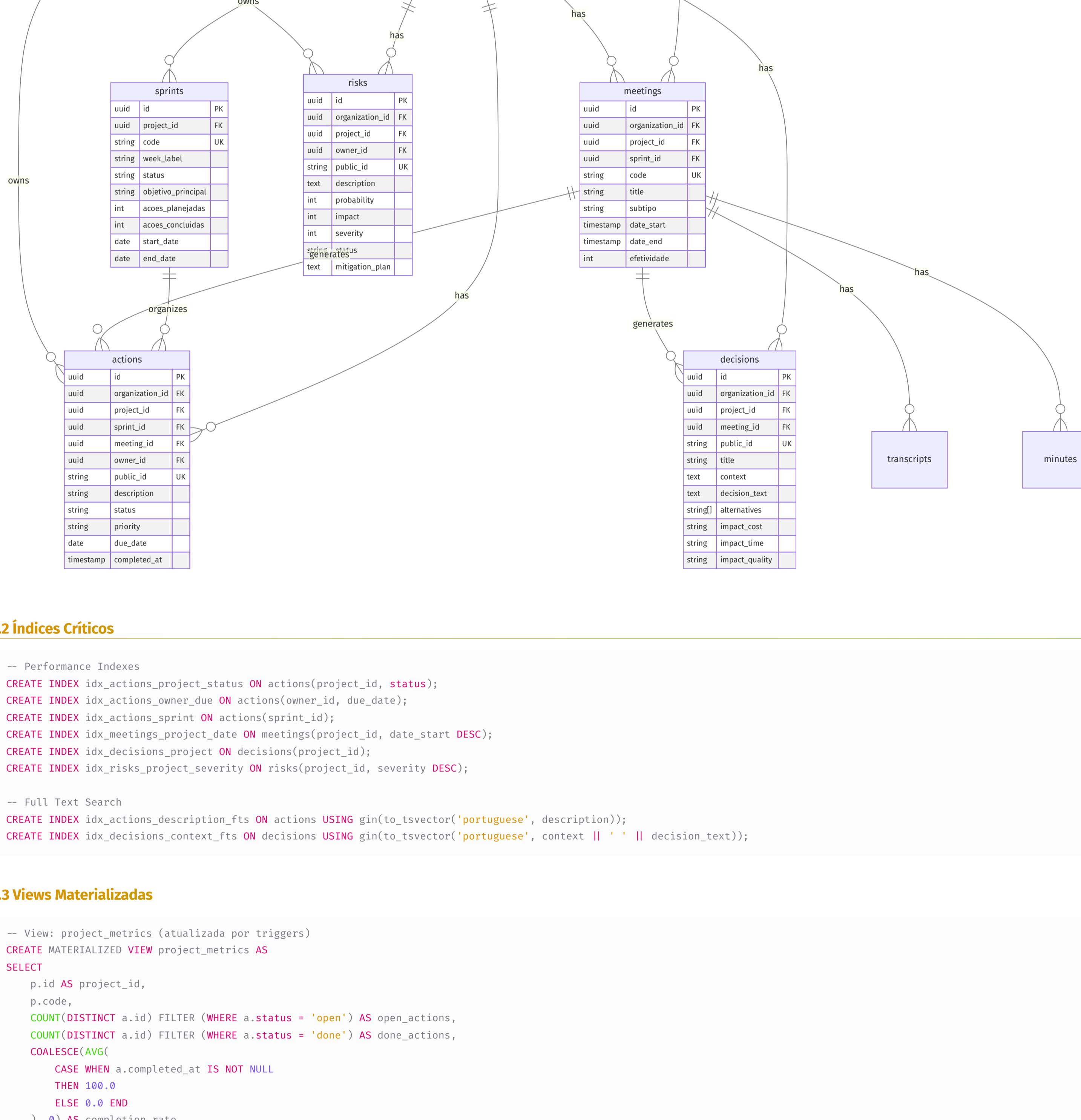
    async def enrich_action_context(
        self,
        action_description: str,
        project_code: str,
        organization_id: UUID
    ) -> Dict:
        """Busca ações similares bem-sucedidas."""
        query_embedding = self.embedding.embed_text(action_description)

        # Filtra por ações concluídas com success_score ≥ 7
        similar = self.qdrant.search_similar_actions(
            query_embedding=query_embedding,
            project_code=project_code,
            organization_id=str(organization_id),
            min_success_score=7,
            limit=5,
            score_threshold=0.70
        )

        return {
            "similar_actions": similar,
            "suggestions_count": len(similar)
        }
    
```

#### 7. BANCO DE DADOS

##### 7.1 Diagrama ER



##### 7.2 Índices Críticos

```

-- Performance Indexes
CREATE INDEX idx_actions_project_status ON actions(project_id, status);
CREATE INDEX idx_actions_owner_due ON actions(owner_id, due_date);
CREATE INDEX idx_actions_sprint_id ON actions(sprint_id);
CREATE INDEX idx_meetings_project_date ON meetings(project_id, date_start DESC);
CREATE INDEX idx_decisions_project ON decisions(project_id);
CREATE INDEX idx_risks_project_severity ON risks(project_id, severity DESC);

-- Full Text Search
CREATE INDEX idx_actions_description_fts ON actions USING gin(to_tsvector('portuguese', description));
CREATE INDEX idx_decisions_context_fts ON decisions USING gin(to_tsvector('portuguese', context || ' ' || decision_text));

```

##### 7.3 Views Materializadas

```

-- View: project_metrics (atualizada por triggers)
CREATE MATERIALIZED VIEW project_metrics AS
SELECT
    p.id AS project_id,
    p.code,
    COUNT(DISTINCT a.id) FILTER (WHERE a.status = 'open') AS open_actions,
    COUNT(DISTINCT a.id) FILTER (WHERE a.status = 'done') AS done_actions,
    COALESCE(AVG(
        CASE WHEN a.completed_at IS NOT NULL
        THEN 100.0
        ELSE 0.0 END
    ), 0) AS completion_rate,
    COALESCE(SUM(a.net_total), 0) AS budget_used,
    MAX(r.severity) AS max_risk_severity
FROM projects p
LEFT JOIN actions a ON a.project_id = p.id
LEFT JOIN expenses e ON e.project_id = p.id AND e.status != 'canceled'
LEFT JOIN risks r ON r.project_id = p.id AND r.status = 'open'
GROUP BY p.id, p.code;

```

#### 8. API REST

##### 8.1 Endpoints Principais

Método	Endpoint	Descrição
POST	/api/v1/meetings/ingest	Processa nova reunião
GET	/api/v1/meetings	Lista reuniões
GET	/api/v1/meetings/{id}	Detalhe de reunião
GET	/api/v1/projects	Lista projetos
GET	/api/v1/projects/{id}	Detalhe de projeto
GET	/api/v1/actions	Lista ações
POST	/api/v1/actions/{id}/complete	Marca ação concluída
GET	/api/v1/decisions	Lista decisões
GET	/api/v1/decisions/similar	Busca decisões similares (RAG)
GET	/api/v1/sprints	Lista sprints
POST	/api/v1/sprints/{id}/close	Fecha sprint

##### 8.2 Endpoint Principal: Ingest Meeting

```
POST /api/v1/meetings/ingest
```

Request:

```
{ "transcript": { "raw_text": "Reunião de alinhamento do Chatbot ... ", "source": "fathom", "source_url": "https://fathom.ai/meeting/123", "language": "pt-BR" }, "metadata": { "title": "Reunião Chatbot - Sprint 48", "date_start": "2025-11-24T14:00:00Z", "date_end": "2025-11-24T15:30:00Z", "participants": ["Pedro Vitor Pagliarin", "Luis Fernando Boff"], "subtopic": "sprint_planning" } }
```

```

ARQUITETURA_ERP_UZZAI_COMPLETA

{
  "project_code": "CHATBOT",
  "sprint_code": "Sprint-2025-W48"
},
{
  "options": {
    "auto_extract": true,
    "generate_minutes": true,
    "update_dashboard": true,
    "create_actions": true
  }
}

```

#### Response 201:

```

{
  "meeting": {
    "id": "70e8400-e29b-41d4-a716-44665540000",
    "code": "MTG-2025-11-24-CHATBOT",
    "title": "Reunião Chatbot - Sprint 48",
    "status": "processed",
    "duration_min": 90,
    "efetividade": 8
  },
  "extracted": {
    "decisions": 3,
    "actions": 7,
    "risks": 2,
    "kaizens": 1,
    "blockers": 0
  },
  "rag_suggestions": {
    "duplicate_decisions": [],
    "similar_actions": [
      {"action_id": "A-2025-100", "similarity": 0.82}
    ]
  },
  "files_generated": {
    "ata": "40-Reuniões/2025-11-24-Reunião-Chatbot-Sprint-48.md",
    "tasks": "20-Projetos/CHATBOT/99 - REUNIÕES PROJETO/2025-11-24-Tasks.md"
  }
}

```

### 8.3 Autenticação JWT

`Authorization: Bearer <jwt_token>`

#### Roles e Permissões:

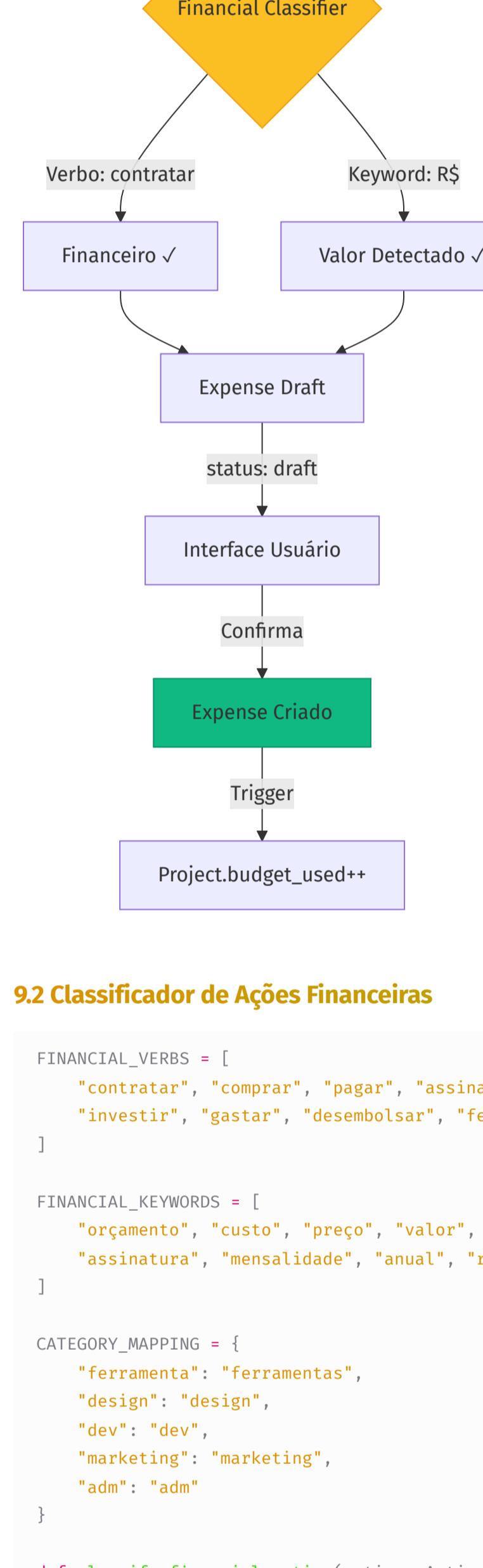
Role	Permissões
owner	Full access (CRUD tudo)
manager	CRUD em projetos/sprints/ações, read-only em org
member	CRUD em ações próprias, read-only no resto
guest	Read-only em tudo

### 8.4 Rate Limiting

Endpoint	Límite	Janela
/meetings/ingest	10 req/hora	Por usuário
/auth/login	5 req/minuto	Por IP
Outros	100 req/minuto	Por usuário

## 9. INTEGRAÇÃO FINANCEIRA

### 9.1 Fluxo Reunião → Despesa



### 9.2 Classificador de Ações Financeiras

```

FINANCIAL_VERBS = [
  "contratar", "comprar", "pagar", "assinar", "adquirir",
  "investir", "gastar", "desembolsar", "fechar plano"
]

FINANCIAL_KEYWORDS = [
  "orçamento", "custo", "preço", "valor", "R$", "reais",
  "assinatura", "mensalidade", "anual", "recorrente"
]

CATEGORY_MAPPING = {
  "ferramentas": "ferramentas",
  "design": "design",
  "dev": "dev",
  "marketing": "marketing",
  "adm": "adm"
}

def classify_financial_action(action: Action) -> Optional[ExpenseDraft]:
    """Classifica se ação gera despesa."""
    text = action.description.lower()

    # Deteta verbos financeiros
    is_financial = any(v in text for v in FINANCIAL_VERBS)

    # Extrai valor se mencionado
    value_match = re.search(r'R\$?\s*(\d,)?(\d)', action.description)
    value = float(value_match.group(1).replace(',', '.')) if value_match else None

    if is_financial:
        return ExpenseDraft(
            description=action.description,
            project_code=action.project,
            action_id=action.id,
            gross_total=value,
            category=infer_category(text),
            status="draft"
        )
    return None

```

## 10. STACK TECNOLÓGICO

### 10.1 Backend

Linguagem	Componente	Tecnologia	Versão
Framework API	Python	FastAPI	3.10+
ORM	SQLAlchemy	2.0+	
Validação	Pydantic	2.0+	
LLM	OpenAI API	gpt-4-0613	
Embeddings	OpenAI	text-embedding-001	
Vector DB	Qdrant	1.7+	
Database	PostgreSQL	15+	
Migrations	Alembic	1.12+	
CLI	Typer	0.9+	
Templates	Jinja2	3.1+	
Async	asyncio	stdlib	

### 10.2 Frontend

Framework	Componente	Tecnologia	Versão
Interface atual	Next.js (futuro)		14+
Plugins	Obsidian	Dataview, Tasks, Calendar	-

### 10.3 Infraestrutura

Container	Componente	Tecnologia
CI/CD		GitHub Actions
Hosting		Vercel (API) + Supabase (DB)
Vector DB		Qdrant Cloud

### 10.4 Docker Compose

```

version: '3.8'

services:
  api:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgres://user:pass@postgres:5432/uzzai
      - QDRANT_URL=http://qdrant:6333
      - OPENAI_API_KEY=${OPENAI_API_KEY}
    depends_on:
      - postgres
      - qdrant

```

```

  postgres:
    image: postgres:15
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: uzzai
    volumes:
      - postgres_data:/var/lib/postgresql/data

```

```

qdrant:
  image: qdrant/qdrant:v1.7.0
  ports:
    - "6333:6333"
  volumes:
    - qdrant_data:/qdrant/storage

volumes:
  postgres_data:
  qdrant_data:

```

## 11. ESTRUTURA DE CÓDIGO

### 11.1 Estrutura de Diretórios

```

uzzai-erp-core/
├── src/
│   ├── domain/
│   │   ├── entities/
│   │   │   ├── __init__.py
│   │   │   ├── meeting.py
│   │   │   ├── action.py
│   │   │   ├── decision.py
│   │   │   ├── project.py
│   │   │   ├── sprint.py
│   │   │   ├── person.py
│   │   │   ├── risk.py
│   │   │   └── kaizen.py
│   │   ├── repositories/
│   │   │   ├── __init__.py
│   │   │   ├── meeting_repository.py
│   │   │   ├── action_repository.py
│   │   │   └── project_repository.py
│   │   ├── services/
│   │   │   └── entity_extractor.py
│   │   └── events/
│   │       └── domain_events.py
│
│   ├── application/
│   │   ├── use_cases/
│   │   │   ├── __init__.py
│   │   │   ├── ingest_meeting.py
│   │   │   ├── update_sprint.py
│   │   │   └── sync_dashboard.py
│   │   └── services/
│   │       ├── rag_context_enricher.py
│   │       └── financial_classifier.py
│
│   ├── infrastructure/
│   │   ├── persistence/
│   │   │   ├── sql/
│   │   │   │   ├── database.py
│   │   │   │   ├── models.py
│   │   │   │   └── repositories/
│   │   │   └── file/
│   │   │       ├── markdown_assembler.py
│   │   │       └── vault_reader.py
│   │   └── rag/
│   │       ├── qdrant_client.py
│   │       ├── embedding_service.py
│   │       ├── decision_indexer.py
│   │       └── action_indexer.py
│
│   ├── llm/
│   │   ├── openai_client.py
│   │   └── prompts/
│   │       ├── action_extraction.py
│   │       └── decision_extraction.py
│
│   ├── agents/
│   │   ├── base_agent.py
│   │   ├── decision_agent.py
│   │   ├── action_agent.py
│   │   ├── kaizen_agent.py
│   │   ├── risk_agent.py
│   │   ├── blocker_agent.py
│   │   ├── project_agent.py
│   │   ├── deadline_agent.py
│   │   ├── priority_agent.py
│   │   ├── sprint_agent.py
│   │   ├── performance_agent.py
│   │   ├── team_health_agent.py
│   │   └── validator_agent.py
│
│   └── orchestrator.py
│
└── interfaces/
    ├── api/
    │   ├── main.py
    │   ├── dependencies.py
    │   └── routes/
    │       ├── meetings.py
    │       ├── projects.py
    │       ├── actions.py
    │       ├── decisions.py
    │       └── auth.py
    └── schemas/
        ├── meeting_schemas.py
        └── action_schemas.py
    └── cli/
        ├── main.py
        └── commands/
            ├── ingest.py
            ├── sprint.py
            └── dashboard.py
└── tests/
    ├── unit/
    └── integration/
    └── e2e/
└── migrations/
    └── versions/
└── templates/
    ├── meeting_template.md.j2
    ├── sprint_template.md.j2
    └── dashboard_template.md.j2
└── docs/
    └── architecture/
└── docker-compose.yml
└── Dockerfile
└── pyproject.toml
└── requirements.txt
└── alembic.ini
└── .env.example

```

### 11.2 Exemplo: Entity Classes

```

# src/domain/entities/action.py
from pydantic import BaseModel, Field
from datetime import date, datetime
from typing import Optional, List, Literal
from uuid import UUID, uuid4

class Action(BaseModel):
    """Entidade de domínio: Ação/Encaminhamento."""

    id: UUID = Field(default_factory=uuid4)
    public_id: str # A-2025-XXX
    description: str
    status: Literal["open", "in_progress", "done", "canceled"] = "open"
    priority: Literal["high", "medium", "low"] = "medium"
    due_date: date
    owner_id: UUID
    project_id: Optional[UUID] = None
    sprint_id: Optional[UUID] = None
    meeting_id: Optional[UUID] = None
    organization_id: UUID
    tags: List[str] = Field(default_factory=list)
    completed_at: Optional[datetime] = None
    created_at: datetime = Field(default_factory=datetime.utcnow)
    updated_at: datetime = Field(default_factory=datetime.utcnow)

    # Metadados RAG
    similar_actions: List[dict] = Field(default_factory=list)

    def to_markdown_task(self) -> str:
        """Converte para formato do plugin Tasks."""
        prio_icon = {"high": "🔴", "medium": "🟡", "low": "🟢"}
        checkbox = "*" if self.status == "done" else " "
        return (
            f"- [{checkbox}] **{self.public_id}: {self.description}**\n"
            f"  [{self.owner_name}] [{prio_icon[self.priority]}]\n"
            f"  [{self.due_date.isoformat()}]\n"
            f"  {prio_icon[self.priority]} project:{self.project_code} #encaminhamento"
        )

    class Config:
        from_attributes = True

```

## 12. ROADMAP DE IMPLEMENTAÇÃO

### 12.1 Fases



### 12.2 Checklist por Fase

#### Fase 0: Lab UzzAI (Fundação)

- Análise do Vault Atual
- Definição do Modelo de Domínio
- Definição da Arquitetura
- Especificação de Automação
- Setup projeto Python (`uzzai-erp-core`)
- Implementar `MarkdownAssembler` básico
- Implementar `TaskFormatter`
- Implementar Multi-Agent Architecture
- Setup Qdrant (Docker)
- RAG Context Enricher (decisões)

#### Fase 1: ERP Interno (Colaboração)

- Configurar múltiplos outputs (Vaults)
- Banco de dados PostgreSQL
- FastAPI MVP (3 endpoints)
- Autenticação JWT
- Multi-tenancy real
- RAG completo (ações + kaizens)
- Automação de Sprints

#### Fase 2: Produto SaaS

- API REST completa
- Frontend Next.js
- Multi-tenancy real
- Billing (Stripe)
- Integrações (Slack, WhatsApp)

### 12.3 Estimativas

Fase	Duração	Horas	Status
Fase 0	4-6 semanas	~120h	Em Progresso
Fase 1	4-5 semanas	~100h	Planejado

Fase	Duração	Horas	Status
Fase 2	6-8 semanas	-160h	Futuro

## MÉTRICAS DE SUCESSO

### Fase 0

- Recall ≥ 85% em extração de decisões/ações
- Tempo de processamento ≤ 60s
- Zero duplicatas em decisões (RAG)

### Fase 1

- 10 reuniões processadas automaticamente
- 3 projetos piloto migrados
- Latência API < 200ms

### Fase 2

- 5 clientes externos ativos
- ARR > R\$ 50.000
- NPS > 40

## REFERÊNCIAS

### Documentos Base

- [00\\_Master\\_Plan.md](#) — Plano mestre do projeto
- [01\\_Domain\\_Model.md](#) — Modelo de domínio
- [02\\_Architecture.md](#) — Arquitetura de alto nível
- [03\\_Roadmap.md](#) — Roadmap detalhado
- [05\\_Automation\\_Specs.md](#) — Especificações de automação
- [05\\_3\\_Multi\\_Agent\\_Architecture\\_R02.md](#) — Arquitetura multi-agente
- [08\\_Database\\_Schema.md](#) — Schema do banco de dados
- [09\\_API\\_SPEC.md](#) — Especificação da API REST
- [12\\_RAG\\_Implementation\\_Plan.md](#) — Plano de implementação RAG

### Templates

- [ARQUITETURA.md](#) — Template de arquitetura

Última Atualização: 2025-11-29T15:00

Autor: Sistema de Documentação ERP-UzzAI

Versão: 2.0.0

Próxima Revisão: Após implementação da Fase 0