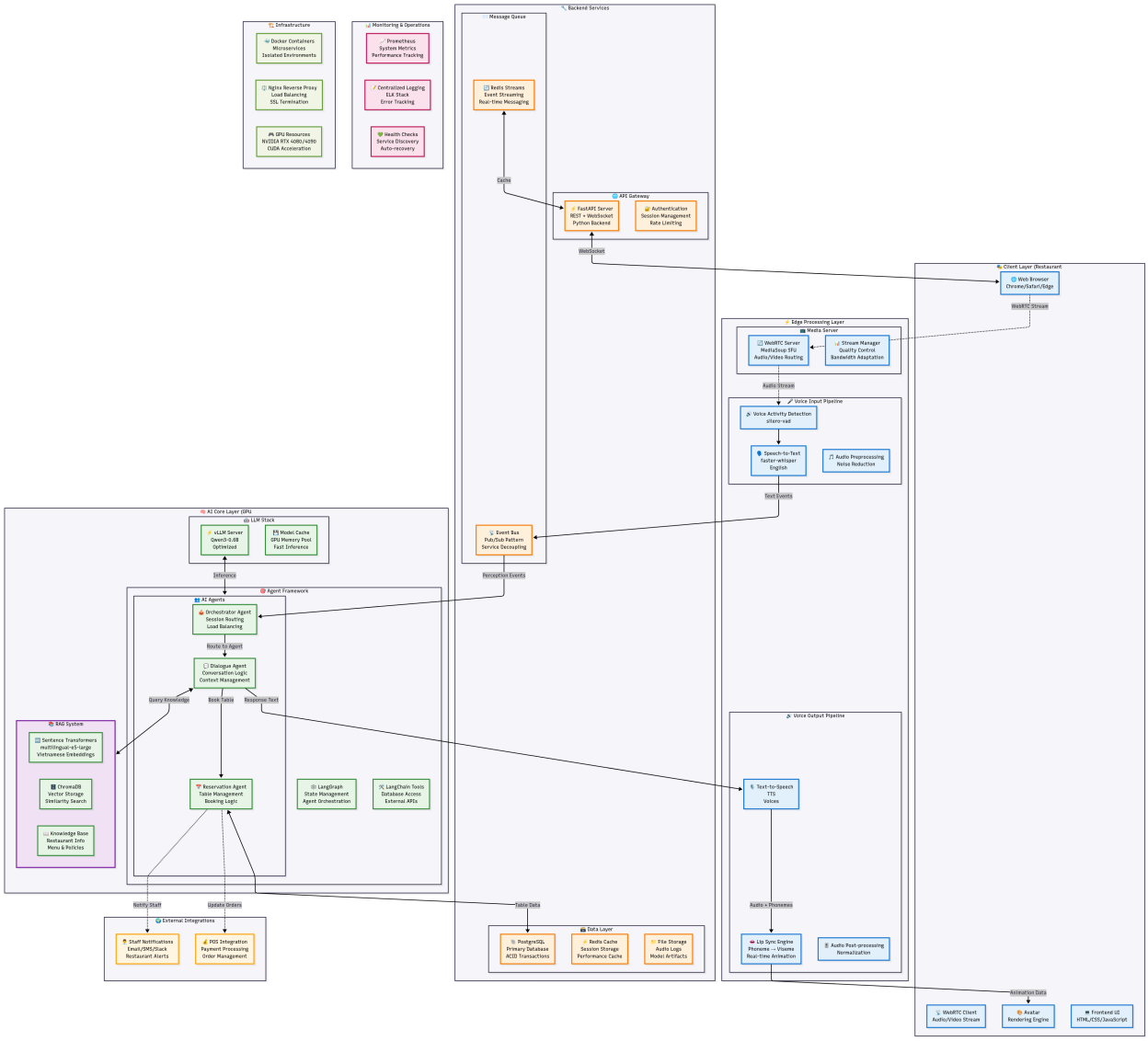
Tech Stack Architecture

Type	Document
Status	Under Review
Priority	Urgent
Assigned To	 Thinh Hung Ho  Minh Hoàng
Due Date	@September 10, 2025
Dependencies	None
Description	Comprehensive documentation of the entire receptionist system architecture, including component interactions, data flows, and integration points.

Tech Stack Architecture Workflow



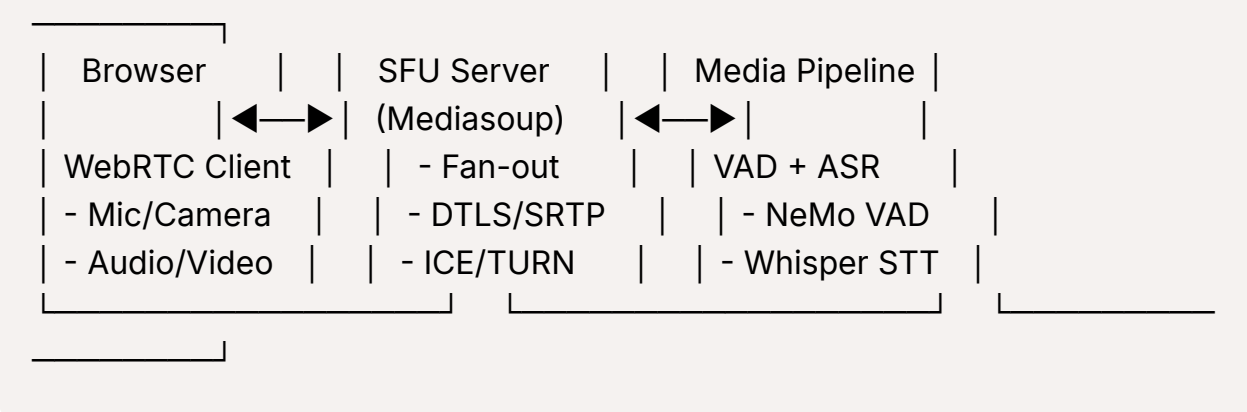
Tech Stack Architecture & Source Structure

1. Detailed Tech Stack Architecture

1.1 Perception Pipeline Stack

WebRTC & Media Processing

Edge Node Architecture:



Voice Activity Detection (VAD)

- **Primary**: NeMo MarbleNet (NVIDIA)
- **Fallback**: WebRTC VAD or Silero VAD
- **Output**: Audio segments with confidence scores
- **Latency Target**: <50ms detection

Automatic Speech Recognition (ASR)

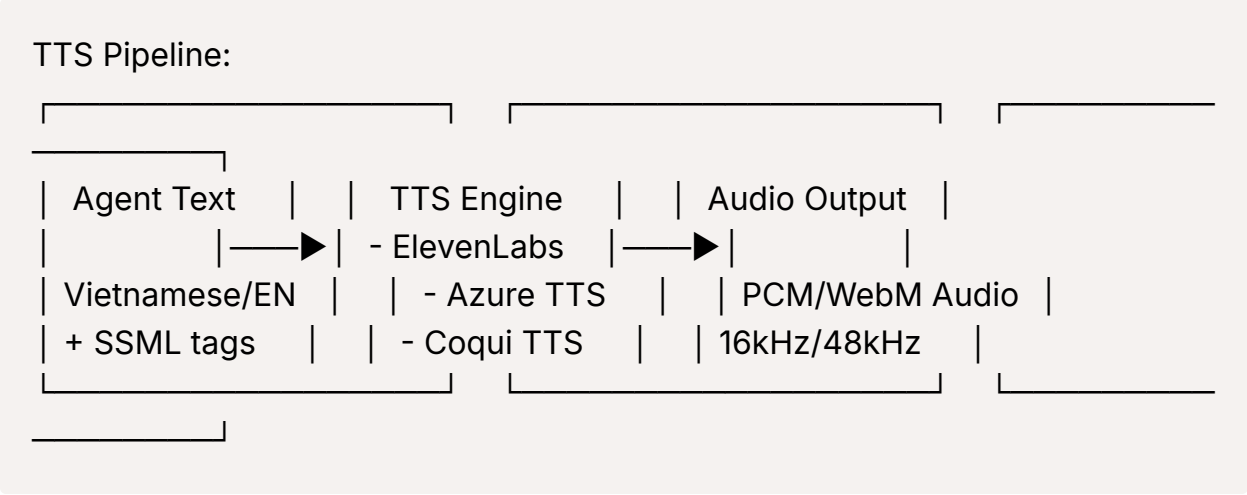
- **Primary**: OpenAI Whisper (streaming adaptation)
- **Alternative**: Google Speech-to-Text Streaming API
- **Features**: Timestamps, confidence scores, Vietnamese + English
- **Latency Target**: <200ms for short utterances

Speaker Diarization

- **Model**: ECAPA-TDNN embeddings
- **Clustering**: Online clustering (incremental)
- **Output**: Speaker IDs with timestamp ranges

1.2 Avatar Pipeline Stack

Text-to-Speech (TTS)



Lip Sync & Animation

- **3D Avatar**: Ready Player Me or custom Three.js model
- **Lip Sync**: Rhubarb Lip Sync or OVRLipSync
- **Animation**: Mixamo animations + facial expressions
- **Rendering**: Three.js WebGL in browser

WebRTC Output

- **Codec**: Opus for audio, VP9/H.264 for video
- **Streaming**: MediaStream API to WebRTC peer connection
- **Latency Target**: <150ms end-to-end

1.3 Agent Layer Stack (LangGraph + LangChain)

LangGraph State Machine Architecture

```
# Agent State Flow
Orchestrator Agent:
  ├── SessionRouter (route events by session_id)
  ├── BackpressureController (rate limiting)
  ├── HealthChecker (timeout/retry logic)
  └── EventDispatcher (emit to dialogue agent)
```

```
Dialogue Agent:
  ├── IntentClassifier (reservation/faq/chitchat)
  ├── ContextManager (session memory)
  ├── ToolSelector (which tools to call)
  ├── ResponseGenerator (craft natural replies)
  └── GuardrailsChecker (safety/policy validation)

Reservation Agent:
  ├── TableMatcher (find available tables)
  ├── HoldManager (temporary reservations)
  ├── AlternativeGenerator (backup options)
  ├── StaffNotifier (alert restaurant staff)
  └── BookingConfirmer (finalize reservation)
```

1.4 Backend Services Stack

Message Bus

- **Primary**: Apache Kafka (for durability)
- **Alternative**: Redis Streams (for simplicity)
- **Topics**: `perception-events`, `agent-utterances`, `booking-actions`

Database Layer

```
-- PostgreSQL with pgvector extension
Tables:
├── sessions (active conversations)
├── reservations (booking data)
├── tables (restaurant layout/availability)
├── knowledge_base (FAQ embeddings)
├── conversation_logs (audit trail)
└── user_profiles (preferences/history)
```

Vector Database (RAG)

- **Primary**: Postgres + pgvector
- **Alternative**: Qdrant or FAISS
- **Embeddings**: OpenAI text-embedding-ada-002
- **Use Cases**: Menu items, policies, common questions

2. Source Structure

```
src/
├── README.md
├── docker-compose.yml
├── .env.example
├── .gitignore
└── services/
    ├── edge-node/                # Media processing at edge
    │   ├── Dockerfile
    │   ├── requirements.txt
    │   ├── src/
    │   │   ├── __init__.py
    │   │   ├── main.py           # FastAPI app entry point
    │   │   ├── webrtc/
    │   │   │   ├── __init__.py
    │   │   │   ├── sfu_server.py    # Mediasoup SFU wrapper
    │   │   │   ├── ice_config.py    # STUN/TURN configuration
    │   │   │   └── peer_manager.py  # WebRTC peer connections
    │   │   ├── perception/
    │   │   │   ├── __init__.py
    │   │   │   ├── pipeline.py      # Main perception orchestrator
    │   │   │   ├── vad/
    │   │   │   │   ├── __init__.py
    │   │   │   │   ├── nemo_vad.py  # NeMo MarbleNet integration
    │   │   │   │   └── fallback_vad.py
    │   │   │   ├── asr/
    │   │   │   │   ├── __init__.py
    │   │   │   │   ├── whisper_streaming.py
```

```
│   │   │   │   └── google_stt.py
│   │   │   ├── diarization/
│   │   │   │   ├── __init__.py
│   │   │   │   ├── ecapa_tdnn.py
│   │   │   │   └── clustering.py
│   │   │   └── emotion/
│   │   │       ├── __init__.py
│   │   │       └── prosody_analyzer.py
│   │   ├── avatar/
│   │   │   ├── __init__.py
│   │   │   ├── pipeline.py      # Avatar rendering orchestrator
│   │   │   ├── tts/
│   │   │   │   ├── __init__.py
│   │   │   │   ├── elevenlabs_tts.py
│   │   │   │   ├── azure_tts.py
│   │   │   │   └── coqui_tts.py
│   │   │   ├── lipsync/
│   │   │   │   ├── __init__.py
│   │   │   │   ├── rhubarb_sync.py
│   │   │   │   └── ovr_lipsync.py
│   │   │   └── animation/
│   │   │       ├── __init__.py
│   │   │       ├── facial_animation.py
│   │   │       └── gesture_controller.py
│   │   ├── streaming/
│   │   │   ├── __init__.py
│   │   │   ├── kafka_producer.py
│   │   │   ├── redis_producer.py
│   │   │   └── event_schemas.py  # Pydantic models for events
│   │   └── utils/
│   │       ├── __init__.py
│   │       ├── audio_utils.py
│   │       ├── video_utils.py
│   │       └── logging_config.py
│   └── tests/
│       ├── test_perception.py
```

```
│       ├── test_avatar.py
│       └── test_webrtc.py
│
├── agent-core/                # LangGraph agents
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── src/
│   │   ├── __init__.py
│   │   ├── main.py            # FastAPI + LangGraph server
│   │   ├── graphs/
│   │   │   ├── __init__.py
│   │   │   ├── orchestrator_graph.py
│   │   │   ├── dialogue_graph.py
│   │   │   └── reservation_graph.py
│   │   ├── agents/
│   │   │   ├── __init__.py
│   │   │   ├── base_agent.py    # Abstract base class
│   │   │   ├── orchestrator/
│   │   │   │   ├── __init__.py
│   │   │   │   ├── agent.py
│   │   │   │   ├── session_router.py
│   │   │   │   ├── backpressure_controller.py
│   │   │   │   └── health_checker.py
│   │   │   ├── dialogue/
│   │   │   │   ├── __init__.py
│   │   │   │   ├── agent.py
│   │   │   │   ├── intent_classifier.py
│   │   │   │   ├── context_manager.py
│   │   │   │   ├── response_generator.py
│   │   │   │   └── guardrails.py
│   │   │   └── reservation/
│   │   │       ├── __init__.py
│   │   │       ├── agent.py
│   │   │       ├── table_matcher.py
│   │   │       ├── hold_manager.py
│   │   │       ├── alternative_generator.py
```

```
│   │   │       └── staff_notifier.py
│   │   ├── tools/
│   │   │   ├── __init__.py
│   │   │   ├── table_tools.py   # LangChain tools for table ops
│   │   │   ├── knowledge_tools.py  # RAG search tools
│   │   │   ├── notification_tools.py
│   │   │   └── calendar_tools.py
│   │   ├── memory/
│   │   │   ├── __init__.py
│   │   │   ├── session_memory.py
│   │   │   ├── conversation_buffer.py
│   │   │   └── long_term_memory.py
│   │   ├── rag/
│   │   │   ├── __init__.py
│   │   │   ├── embeddings.py
│   │   │   ├── vector_store.py
│   │   │   ├── retrievers.py
│   │   │   └── knowledge_base.py
│   │   ├── streaming/
│   │   │   ├── __init__.py
│   │   │   ├── kafka_consumer.py
│   │   │   ├── kafka_producer.py
│   │   │   └── event_handlers.py
│   │   └── utils/
│   │       ├── __init__.py
│   │       ├── llm_clients.py   # OpenAI, Claude, etc.
│   │       ├── prompt_templates.py
│   │       └── validation.py
│   └── tests/
│       ├── test_orchestrator.py
│       ├── test_dialogue.py
│       └── test_reservation.py
│
├── backend-services/           # Supporting microservices
│   ├── table-management/
│   │   ├── Dockerfile
```

```
│   │   ├── src/
│   │   │   ├── main.py         # FastAPI CRUD service
│   │   │   ├── models.py       # SQLAlchemy models
│   │   │   ├── schemas.py      # Pydantic schemas
│   │   │   ├── crud.py         # Database operations
│   │   │   ├── database.py     # DB connection
│   │   │   └── api/
│   │   │       ├── __init__.py
│   │   │       ├── tables.py   # Table endpoints
│   │   │       └── reservations.py
│   │   └── migrations/         # Alembic migrations
│   ├── knowledge-base/
│   │   ├── Dockerfile
│   │   ├── src/
│   │   │   ├── main.py         # FastAPI RAG service
│   │   │   ├── embeddings.py
│   │   │   ├── vector_db.py
│   │   │   ├── indexing.py
│   │   │   └── search.py
│   │   └── data/
│   │       ├── menu.json
│   │       ├── policies.md
│   │       └── faqs.json
│   └── notification-service/
│       ├── Dockerfile
│       ├── src/
│       │   ├── main.py         # FastAPI notification service
│       │   ├── sms_sender.py   # SMS notifications
│       │   ├── email_sender.py # Email notifications
│       │   ├── slack_sender.py # Staff Slack alerts
│       │   └── webhook_sender.py
│       └── templates/
│           ├── sms_templates.json
│           └── email_templates.html
│
├── web-frontend/               # React.js frontend
```

```
│   ├── package.json
│   ├── tsconfig.json
│   ├── vite.config.ts
│   ├── public/
│   │   ├── index.html
│   │   └── assets/
│   │       ├── avatars/        # 3D avatar models
│   │       └── animations/     # Animation files
│   ├── src/
│   │   ├── main.tsx
│   │   ├── App.tsx
│   │   ├── components/
│   │   │   ├── Avatar/
│   │   │   │   ├── AvatarRenderer.tsx
│   │   │   │   ├── LipSyncController.tsx
│   │   │   │   └── GestureController.tsx
│   │   │   ├── WebRTC/
│   │   │   │   ├── MediaCapture.tsx
│   │   │   │   ├── PeerConnection.tsx
│   │   │   │   └── StreamManager.tsx
│   │   │   ├── UI/
│   │   │   │   ├── ChatInterface.tsx
│   │   │   │   ├── ReservationForm.tsx
│   │   │   │   └── LoadingSpinner.tsx
│   │   │   └── Admin/
│   │   │       ├── TableManagement.tsx
│   │   │       ├── ConversationLogs.tsx
│   │   │       └── SystemHealth.tsx
│   │   ├── hooks/
│   │   │   ├── useWebRTC.ts
│   │   │   ├── useAvatar.ts
│   │   │   └── useReservation.ts
│   │   ├── services/
│   │   │   ├── api.ts          # API client
│   │   │   ├── websocket.ts    # WebSocket connection
│   │   │   └── webrtc.ts       # WebRTC utilities
```

```
│   │   ├── stores/            # State management
│   │   │   ├── conversationStore.ts
│   │   │   ├── reservationStore.ts
│   │   │   └── avatarStore.ts
│   │   ├── types/
│   │   │   ├── conversation.ts
│   │   │   ├── reservation.ts
│   │   │   └── webrtc.ts
│   │   └── utils/
│   │       ├── audioUtils.ts
│   │       ├── videoUtils.ts
│   │       └── formatters.ts
│   └── dist/
│
└── infrastructure/            # Deployment & DevOps
    ├── kubernetes/
    │   ├── namespace.yaml
    │   ├── edge-node/
    │   │   ├── deployment.yaml
    │   │   ├── service.yaml
    │   │   └── configmap.yaml
    │   ├── agent-core/
    │   │   ├── deployment.yaml
    │   │   ├── service.yaml
    │   │   └── hpa.yaml       # Horizontal Pod Autoscaler
    │   ├── backend-services/
    │   │   ├── table-management.yaml
    │   │   ├── knowledge-base.yaml
    │   │   └── notification-service.yaml
    │   ├── databases/
    │   │   ├── postgres.yaml
    │   │   ├── redis.yaml
    │   │   └── kafka.yaml
    │   └── ingress/
    │       ├── nginx-ingress.yaml
    │       └── tls-certificates.yaml
```

```
├── terraform/
│   ├── main.tf
│   ├── variables.tf
│   ├── outputs.tf
│   ├── providers.tf
│   ├── modules/
│   │   ├── gke-cluster/     # Google Kubernetes Engine
│   │   ├── vpc-network/
│   │   ├── load-balancer/
│   │   └── monitoring/
│   └── environments/
│       ├── dev/
│       ├── staging/
│       └── production/
├── monitoring/
│   ├── prometheus/
│   │   ├── config.yaml
│   │   └── rules.yaml
│   ├── grafana/
│   │   ├── dashboards/
│   │   └── datasources.yaml
│   └── jaeger/              # Distributed tracing
│       └── config.yaml
└── scripts/
    ├── build.sh             # Build all services
    ├── deploy.sh            # Deploy to K8s
    ├── seed-data.sh         # Initialize databases
    └── health-check.sh      # System health verification
```

3. Event Schemas & API Contracts

3.1 Core Event Types (Pydantic Models)

```
# services/edge-node/src/streaming/event_schemas.pyfrom pydantic import BaseModel
```

```
from typing import Optional, List, Dict, Any
from datetime import datetime

class PerceptionEvent(BaseModel):
    type: str = "PerceptionEvent"
    session_id: str
    text: str
    start_ts: float
    end_ts: float
    speaker_id: Optional[str] = None
    emotion: Optional[str] = "neutral"
    asr_confidence: float
    language: str = "vi"# Vietnamese default
    metadata: Dict[str, Any] = {}

class AgentUtterance(BaseModel):
    type: str = "AgentUtterance"
    session_id: str
    text: str
    style: str = "friendly"
    ssml: Optional[str] = None
    gestures: List[str] = []
    emotion: str = "neutral"
    language: str = "vi"
    priority: int = 1# 1=normal, 2=urgentclass ReservationRequest(BaseModel):
    session_id: str
    party_size: int
    preferred_time: datetime
    customer_name: Optional[str] = None
    contact_info: Optional[str] = None
    special_requests: List[str] = []

class SystemEvent(BaseModel):
    type: str
    session_id: str
    component: str
```

```
    level: str# INFO, WARN, ERROR
    message: str
    timestamp: datetime
    metadata: Dict[str, Any] = {}
```

3.2 Database Schema

```
-- PostgreSQL schemasCREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgvector";

-- Sessions tableCREATE TABLE sessions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    status VARCHAR(20) DEFAULT 'active',
    guest_count INTEGER DEFAULT 1,
    language VARCHAR(5) DEFAULT 'vi',
    metadata JSONB DEFAULT '{}'
);

-- Reservations tableCREATE TABLE reservations (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    session_id UUID REFERENCES sessions(id),
    table_id INTEGER,
    party_size INTEGER NOT NULL,
    reservation_time TIMESTAMP NOT NULL,
    customer_name VARCHAR(100),
    contact_info VARCHAR(100),
    status VARCHAR(20) DEFAULT 'pending',
    special_requests TEXT[],
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Tables tableCREATE TABLE restaurant_tables (
```

```
    id SERIAL PRIMARY KEY,
    table_number VARCHAR(10) UNIQUE NOT NULL,
    capacity INTEGER NOT NULL,
    location VARCHAR(50),
    status VARCHAR(20) DEFAULT 'available',
    metadata JSONB DEFAULT '{}'
);

-- Knowledge base for RAGCREATE TABLE knowledge_entries (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    title VARCHAR(200) NOT NULL,
    content TEXT NOT NULL,
    category VARCHAR(50),
    embedding vector(1536),-- OpenAI ada-002 dimensions
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_knowledge_embedding ON knowledge_entries
USING ivfflat (embedding vector_cosine_ops);
```

4. Key Integration Points

4.1 WebRTC → Perception Pipeline

- **Protocol**: WebRTC DataChannel + MediaStream
- **Format**: Opus audio, VP9 video
- **Buffering**: Circular buffer for streaming ASR
- **Backpressure**: Drop frames if processing can't keep up

4.2 Perception → Agent Core

- **Transport**: Kafka topics or Redis Streams
- **Serialization**: JSON with Pydantic validation

- **Delivery**: At-least-once with deduplication
- **Batching**: Group events by session_id for context

4.3 Agent Core → Avatar Pipeline

- **Transport**: Direct HTTP/2 calls or message queue
- **Caching**: Cache TTS audio for common phrases
- **Prioritization**: Queue with priority levels
- **Failover**: Fallback to simpler TTS if premium fails

4.4 Cross-Service Authentication

- **API Gateway**: Kong or Nginx with JWT validation
- **Service Mesh**: Istio for mTLS between services
- **Secrets**: Kubernetes secrets + HashiCorp Vault
- **Rate Limiting**: Per-session and per-service limits

5. Deployment Architecture

5.1 Edge Nodes (Low Latency)

- **Location**: Geographically close to restaurant
- **Services**: SFU + Perception + Avatar pipelines
- **Scaling**: Horizontal pod autoscaling based on WebRTC connections
- **Hardware**: GPU nodes for VAD/ASR inference

5.2 Core Nodes (High Compute)

- **Location**: Centralized cloud region
- **Services**: Agent Core + Backend Services
- **Scaling**: Based on message queue depth
- **Hardware**: CPU-optimized for LangGraph state machines

5.3 Data Layer

- **Primary DB**: PostgreSQL with read replicas
- **Cache**: Redis for session state + rate limiting
- **Message Bus**: Kafka cluster with replication
- **Object Storage**: S3/GCS for audio logs + models

This architecture provides a solid foundation for building a production-ready digital human restaurant assistant with clear separation of concerns, scalable components, and robust error handling.