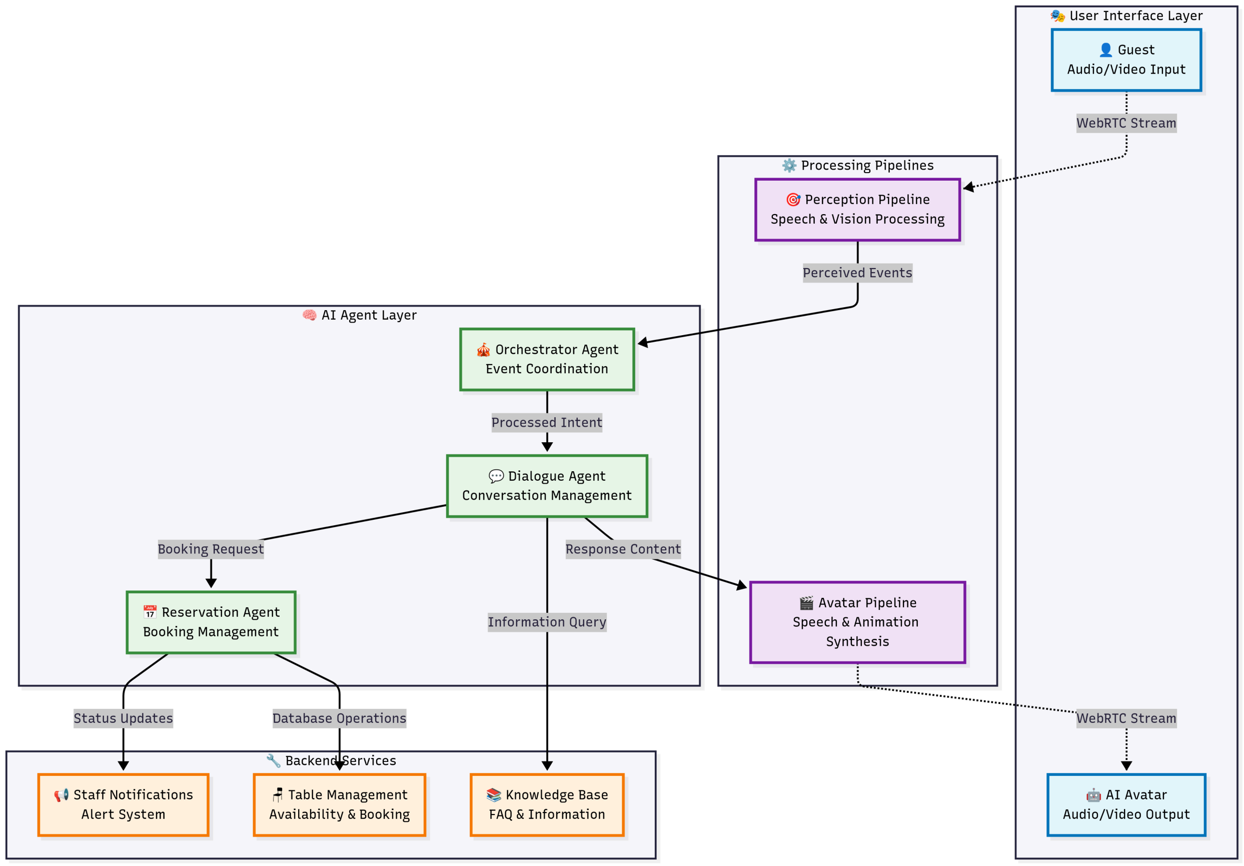
System Architecture

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.

Overall Architecture


🎭 User Interface Layer
👤 Guest Audio/Video Input
WebRTC Stream
⚙️ Processing Pipelines
🎯 Perception Pipeline Speech & Vision Processing
Perceived Events
🧠 AI Agent Layer
🎪 Orchestrator Agent Event Coordination
Processed Intent
💬 Dialogue Agent Conversation Management
Booking Request
Response Content
📅 Reservation Agent Booking Management
Information Query
🎬 Avatar Pipeline Speech & Animation Synthesis
Status Updates
Database Operations
WebRTC Stream
🔧 Backend Services
📢 Staff Notifications Alert System
🪑 Table Management Availability & Booking
📚 Knowledge Base FAQ & Information
🤖 AI Avatar Audio/Video Output

User: Hôm nay nhà hàng còn món gì?

USer: Đặt món:⇒

Agent: Status: món nào còn

Table: Số bàn đã đặt.

Super-Receptionist:

- QnA expert (prompt cho restaurant) ⇒ common questions.
- RAG expert ⇒ Chat with file
- Agent expert ⇒ Supervisers (agent nhỏ,....)
  - 1 questions:

        - câu trả lời nằm ở nhiều DB ⇒ thì phải query vào nhiều DB để lấy (MCP,...)
    - Reservations

Target Architecture (high level)

Pipelines (deterministic processing)

- **Perception Pipeline** → turns live audio/video into structured events (text + timestamps + speaker + sentiment).
- **Avatar Pipeline** → turns agent replies into speech + lip-sync + render (WebRTC).

Agents (LLM + decisions)

- **Dialogue Agent** → goals, policy, tool use, memory, guardrails.
- **Reservation Agent** → table logic, alternatives, staff pings, retries.
- **Orchestrator Agent** → session routing, supervision, recovery, rate limiting.

This split mirrors best practice: *use agents only where reasoning/branching is needed; keep I/O as pipelines* (and orchestrate agents with **LangGraph**, which is purpose-built for stateful agent workflows).

---

1) Perception Pipeline (no LLM)

Goal: Low-latency, reliable "who said what, when, and how" in a busy restaurant.

Flow

1. **WebRTC Ingest (browser ↔ edge)**
    - Use **SFU** (Selective Forwarding Unit) for scalable, low-cost fan-out of audio/video.
2. **VAD (voice activity detection)** → gate compute and segment utterances

    - Proven real-time VAD options: **NeMo MarbleNet** / Realtime APIs with built-in VAD.
3. **Streaming ASR** (timestamps)
    - Whisper can be adapted for streaming (community impls). If you need sub-second latency, consider a native streaming ASR.
4. **Speaker diarization** (optional for multi-party)
    - ECAPA-TDNN embeddings + clustering is a standard recipe.
5. **Prosody/emotion** (optional but helpful) → coarse states (neutral / stressed / happy).
6. **Pack event** → PerceptionEvent{ session_id, text, start_ts, end_ts, speaker_id?, emotion?, asr_conf }
7. **Emit to bus** (Kafka/Redis streams) → consumed by **Orchestrator Agent**.

> Why pipeline? This is deterministic signal processing; no planning/branching is required. The
>
> **Dialogue Agent**

---

2) Avatar Pipeline (no LLM)

Goal: Natural, low-latency output back to guests.

Flow

1. **Input**: AgentUtterance{ text, style, ssml?, gestures?, emotion? }
2. **TTS** (neural, SSML) → waveform
3. **Lip-sync/animation** → drive face mesh / avatar (browser or server), then
4. **WebRTC out** via the same SFU.

> Why pipeline? It deterministically renders what agents decide. Any “decision” (tone/pace/gesture) is a parameter decided upstream by the
>
> **Dialogue Agent**

---

3) Dialogue Agent (LLM + tools)

Goal: Understand intents (reservation, walk-in, FAQ), decide next actions, call tools, and talk naturally.

- **Built with LangGraph** for *stateful, branching* behaviour (retries, loops, tool decisions). LangChain provides the *tools* surface the agent can call.
- **Inputs**: PerceptionEvent stream (batched by session).
- **Policy** (examples):
  - If **intent=reservation** → call **Reservation Agent**.
  - If **FAQ** → do **RAG** against menu/policies.
- **Memory**: session state (party size, name, ETA, preferences).
- **Guardrails**: prompt/runtime checks (deny payments, etc.).
- **Output**: AgentUtterance to **Avatar Pipeline** + action messages to tools/agents.

---

4) Reservation Agent (LLM + domain rules)

Goal: Own the table-allocation problem end-to-end.

- **State machine** (LangGraph): find table → hold → confirm → timebox → release.
- **Decisions**: alternatives (bar waiting, split tables, future slot), wait time estimates, overbook policy.
- **Tools**:
  - tables.lookup(date, party_size)
  - tables.place_hold(table_id, ttl)
  - notify.staff(channel, msg)
- **Output**: structured result to **Dialogue Agent** (success/alt/retry) + staff notifications.

---

5) Orchestrator Agent (LLM optional; mostly control logic)

Goal: Keep everything synchronized per guest/session and recover from issues.

- **Responsibilities**
  - Route **PerceptionEvent → Dialogue Agent** (by session_id).
  - Backpressure & timeouts (e.g., if ASR stalls, ask guest to repeat).
  - Health checks, retries, circuit breakers.
- **Why an agent?** It manages *long-lived*, *concurrent* sessions with branching control; it benefits from LangGraph's state and transitions even without an LLM call on every step.