Kalamna - AI Customer Support Egyptian Service

Supervisors: Dr. Ahmed Yousry & Eng. Mahmoud Sobhy

Faculty of Computer Science, Benha National University

Academic Year: 2025–2026

Team Members: Bassant Hossam, Ahmed Ehab Elattar, Eyad Hesham, Zeyad Wael, Shahda Mohamed, Toni Ehab

1. Idea Description

Our project is an AI-powered, customizable Arabic "Egyptian" customer support platform designed to help SMEs automate their customer service across channels like WhatsApp, Messenger, and web platforms by integrating our service.

Main Features:

- Understanding of the Arabic dialect of Egypt.
- Customization & rule for answers.
- Text support.
- Voice support (speech-to-text & text-to-speech).
- Real-time emotion detection.
- Multi-intent handling.
- Integrations: WhatsApp, Messenger, Websites, Apps.

2. Problem Statement

Local businesses struggle with:

- Providing consistent, 24/7 customer support.
- Repetitive questions.
- Missed messages.
- Slow response times.

All this leads to poor customer satisfaction. Existing solutions are often expensive, rigid, or lack authentic local language and dialect support.

3. Proposed Solution

We aim to build a flexible, API-powered Agent/Services system that:

- Uses advanced AI Models & APIs (Gemini, GPT, Whisper) for text, voice, and emotion-aware replies.
- Supports the Egyptian dialect as a primary focus.
- Detects user emotions and adapts responses accordingly.
- Allows businesses to fully customize the bot's tone, responses, and operating hours.

4. Prototype & Identity

We are currently developing the prototype. The chatbot will be named **Cleo** (short for *Cleopatra*), and the overall service will be called **Kalamna**. Our chosen theme blends **Ancient Egyptian aesthetics** with an **Arabic-focused** design direction. We are now working on a sample prototype that will later be updated to fully reflect our visual identity.

Figma Link: *(Pending link)*

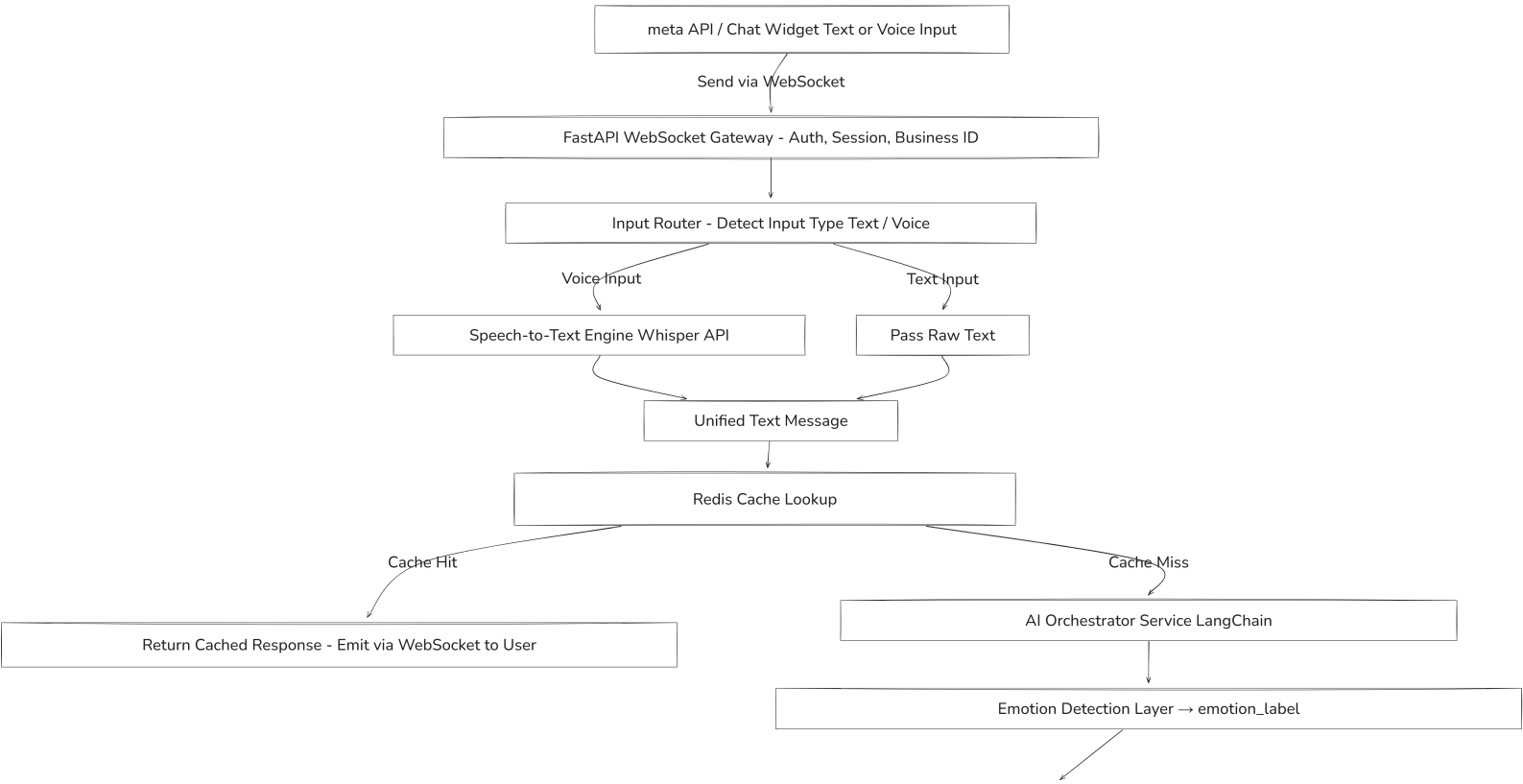
5. System Analysis Diagrams

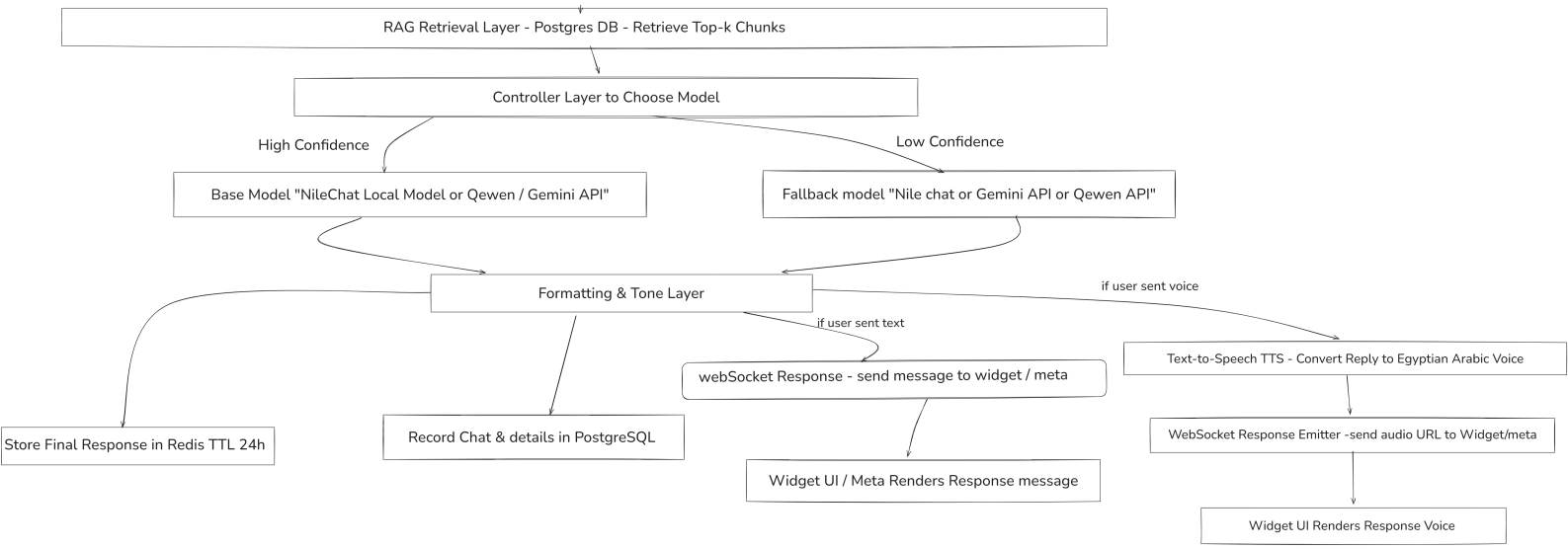
We are currently in the system analysis phase. The following diagrams represent our vision and planned design.

- Flowcharts for user and business Flow
- ERD
- sequence User diagram
- sequence Business diagram
- DFD Level 0 "Context"
- DFD Level 1 "User Flow"
- DFD Level 2 "User Flow"
- DFD Level 1 "Business Flow"
- High level system design & Arch Diagram ¿¿ PNG Version not here
- UseCase User and Business Diagram

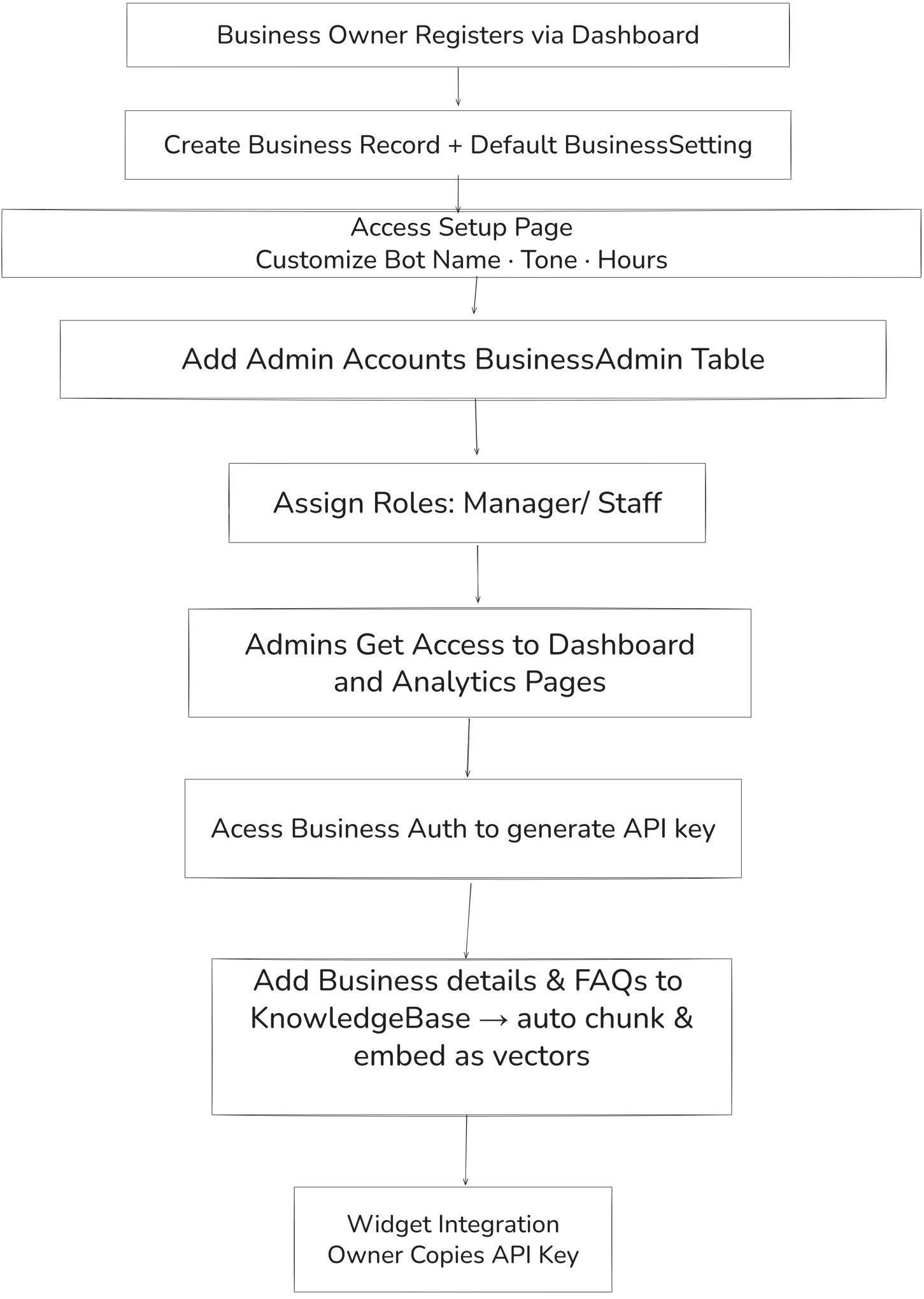
with that we also working on prototype for widget and dashboard

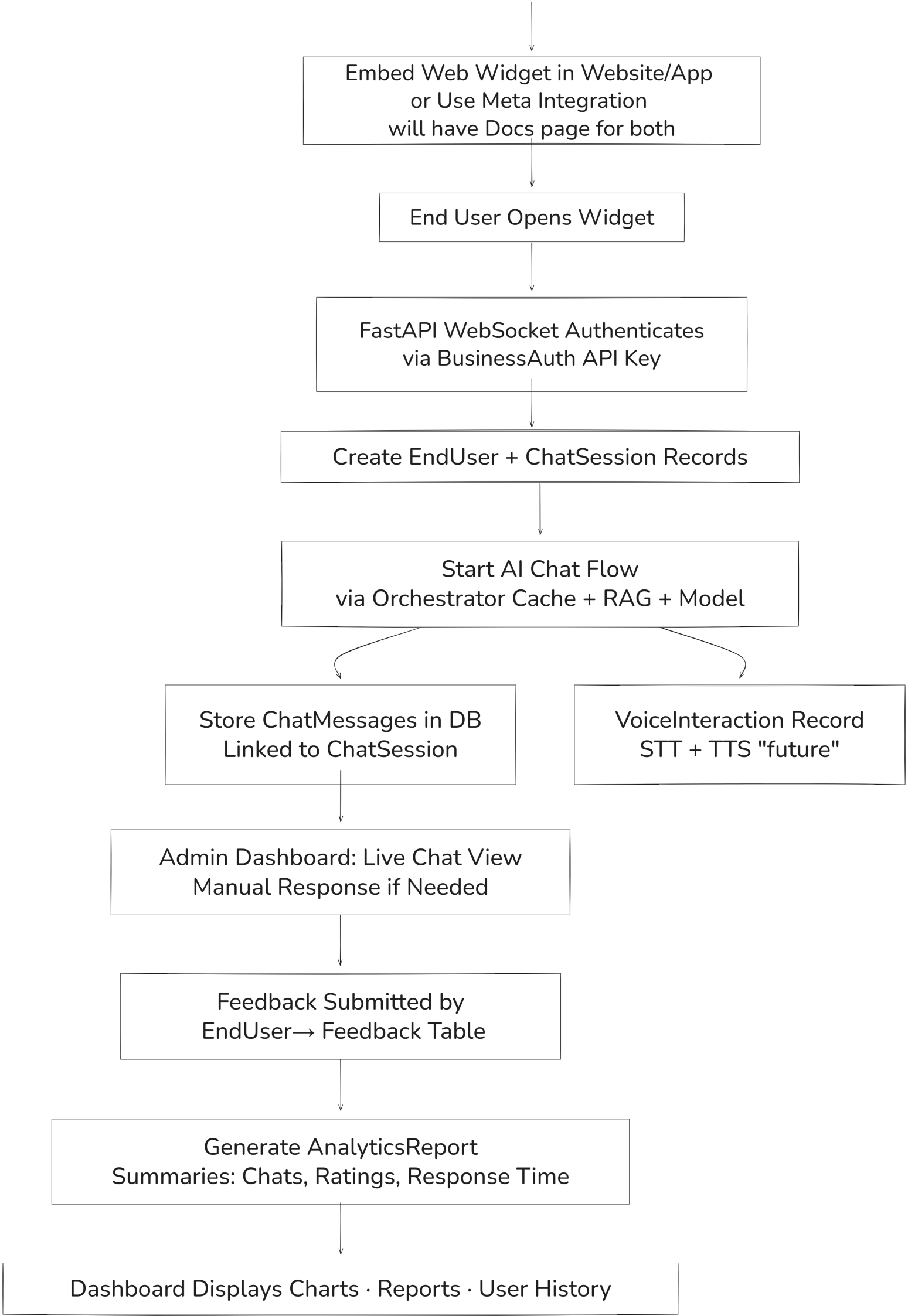
User flow AI Detailed Flowchart

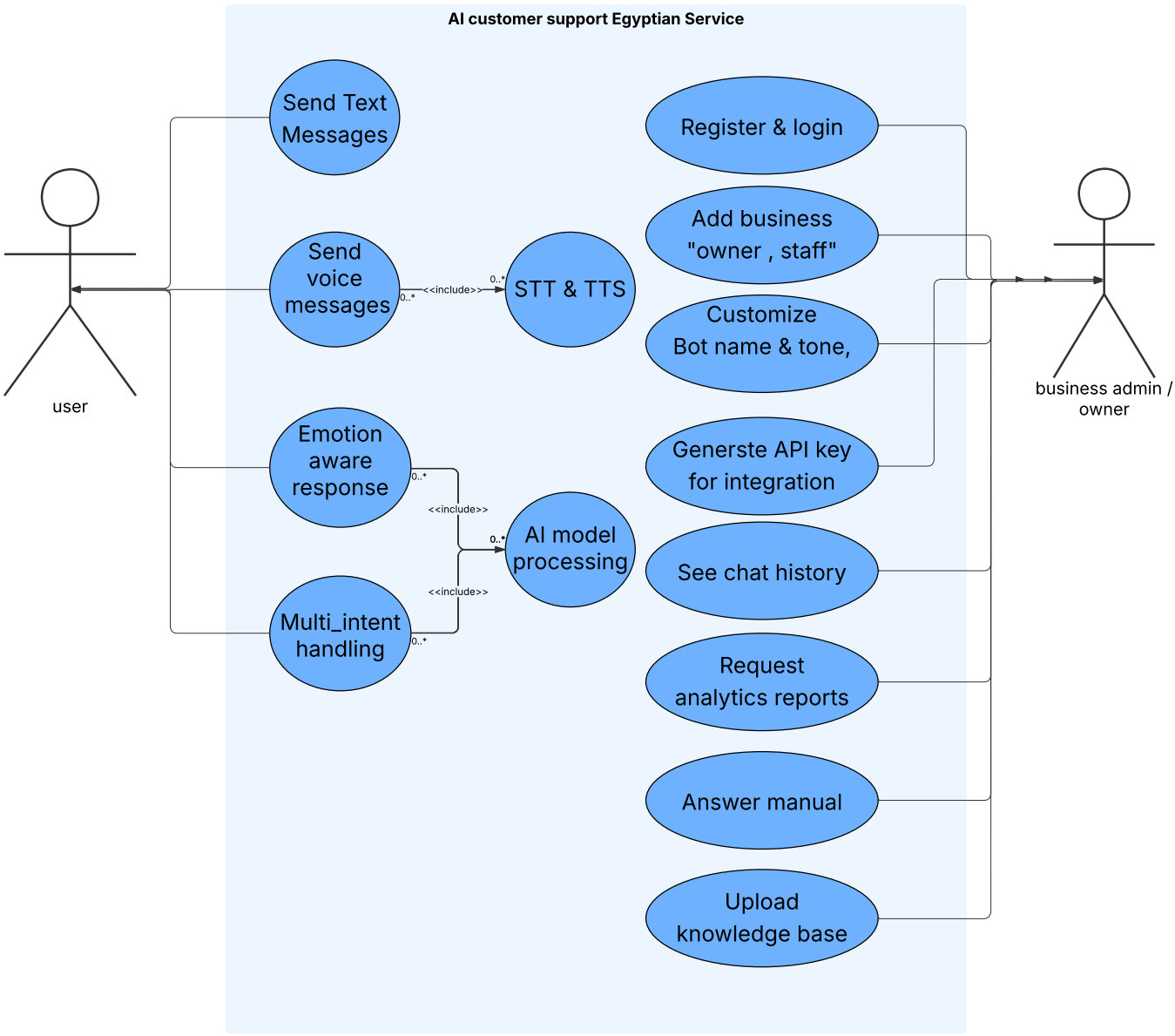


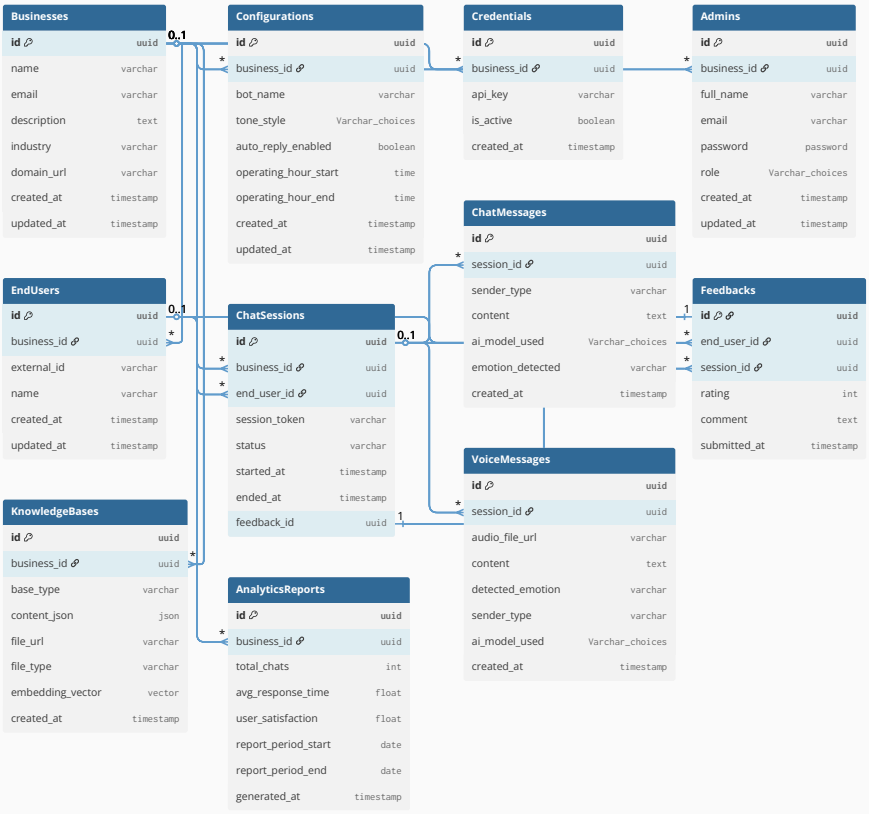

RAG Retrieval Layer - Postgres DB - Retrieve Top-k Chunks
Controller Layer to Choose Model
High Confidence
Low Confidence
Base Model "NileChat Local Model or Qewen / Gemini API"
Fallback model "Nile chat or Gemini API or Qewen API"
Formatting & Tone Layer
if user sent voice
if user sent text
Text-to-Speech TTS - Convert Reply to Egyptian Arabic Voice
webSocket Response - send message to widget / meta
Store Final Response in Redis TTL 24h
Record Chat & details in PostgreSQL
WebSocket Response Emitter -send audio URL to Widget/meta
Widget UI / Meta Renders Response message
Widget UI Renders Response Voice

Business Flowchart detailed diagram

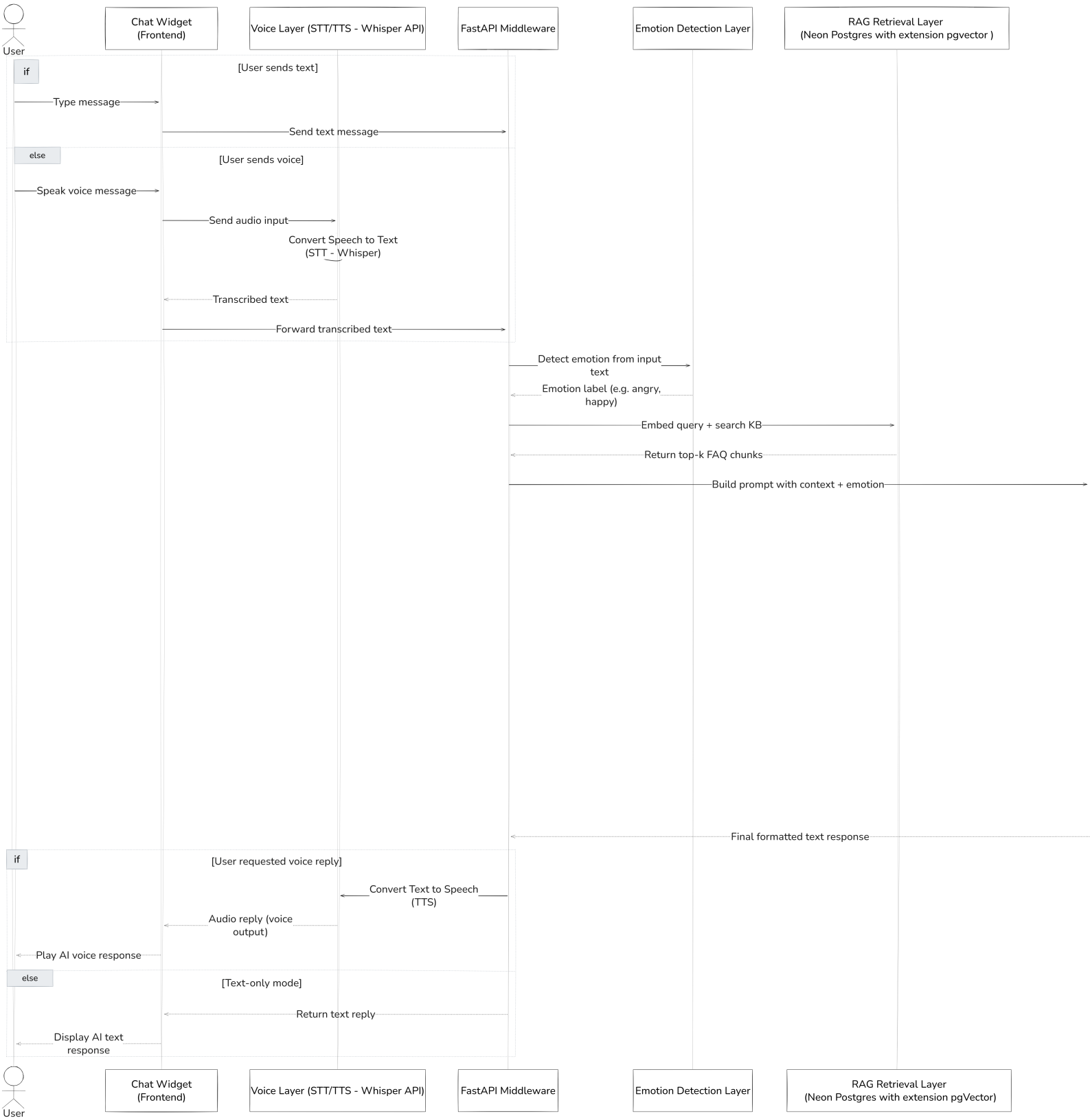


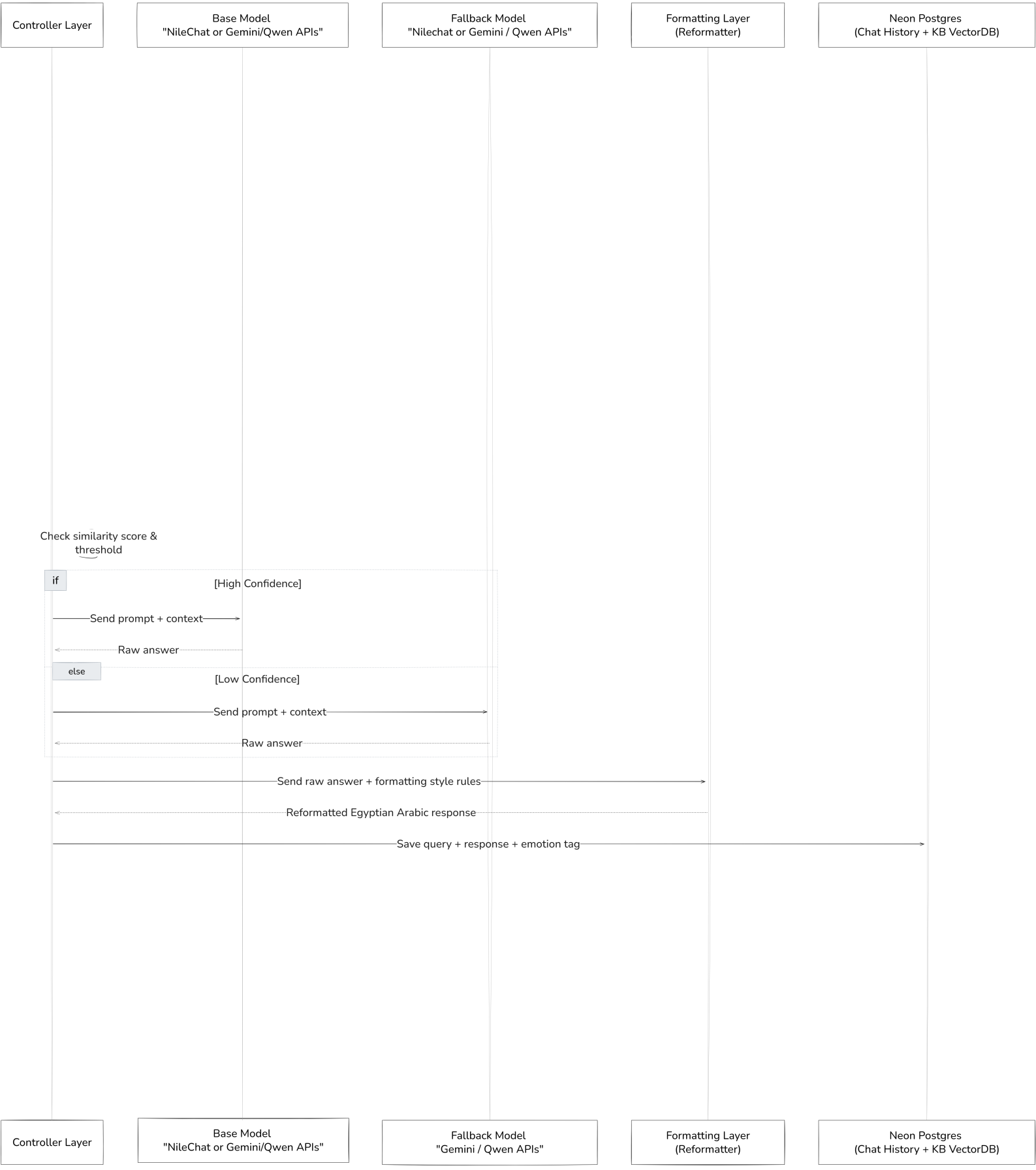

Embed Web Widget in Website/App
or Use Meta Integration
will have Docs page for both
End User Opens Widget
FastAPI WebSocket Authenticates
via BusinessAuth API Key
Create EndUser + ChatSession Records
Start AI Chat Flow
via Orchestrator Cache + RAG + Model
Store ChatMessages in DB
Linked to ChatSession
VoiceInteraction Record
STT + TTS "future"
Admin Dashboard: Live Chat View
Manual Response if Needed
Feedback Submitted by
EndUser→ Feedback Table
Generate AnalyticsReport
Summaries: Chats, Ratings, Response Time
Dashboard Displays Charts · Reports · User History


AI customer support Egyptian Service
Send Text Messages
Send voice messages
STT & TTS
<<include>>
0..*
Emotion aware response
Multi_intent handling
AI model processing
Register & login
Add business "owner , staff"
Customize Bot name & tone,
Generste API key for integration
See chat history
Request analytics reports
Answer manual
Upload knowledge base
user
business admin / owner

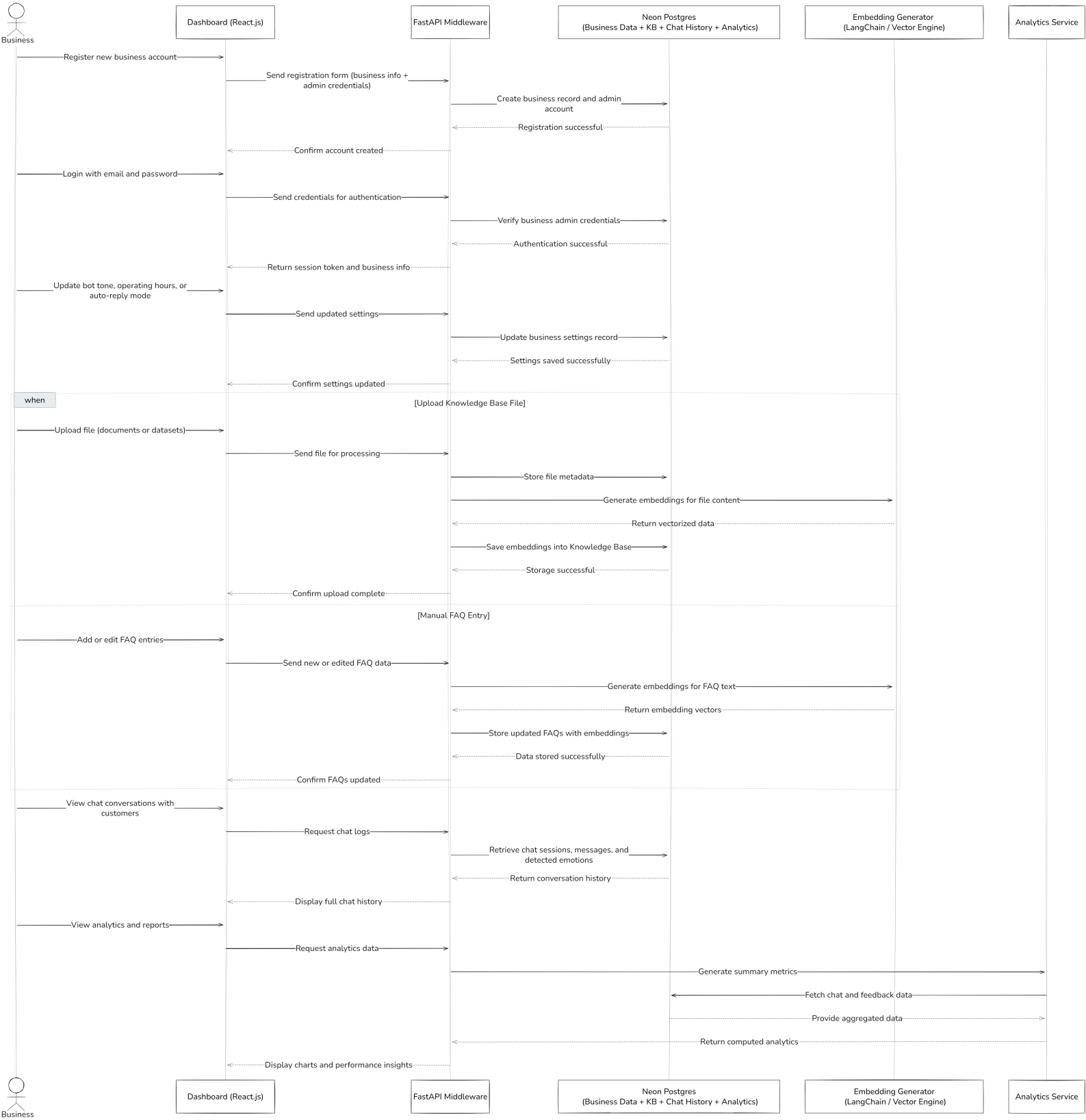


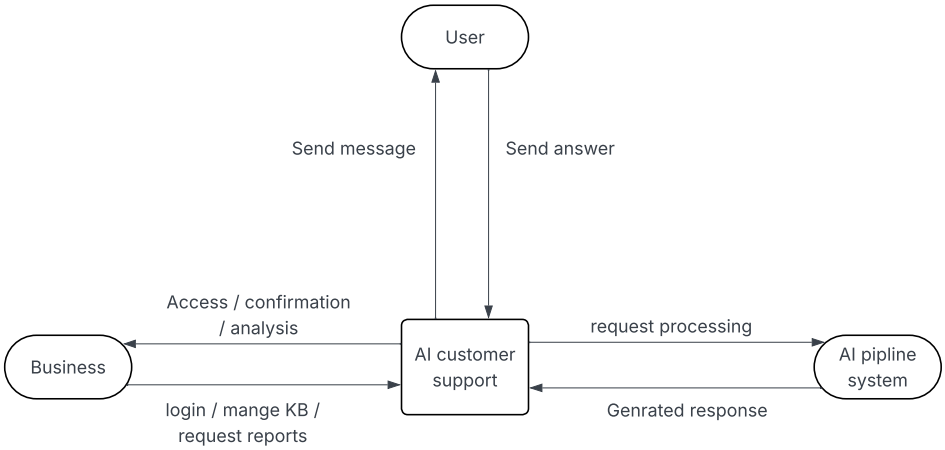
Sequence Diagram user flow

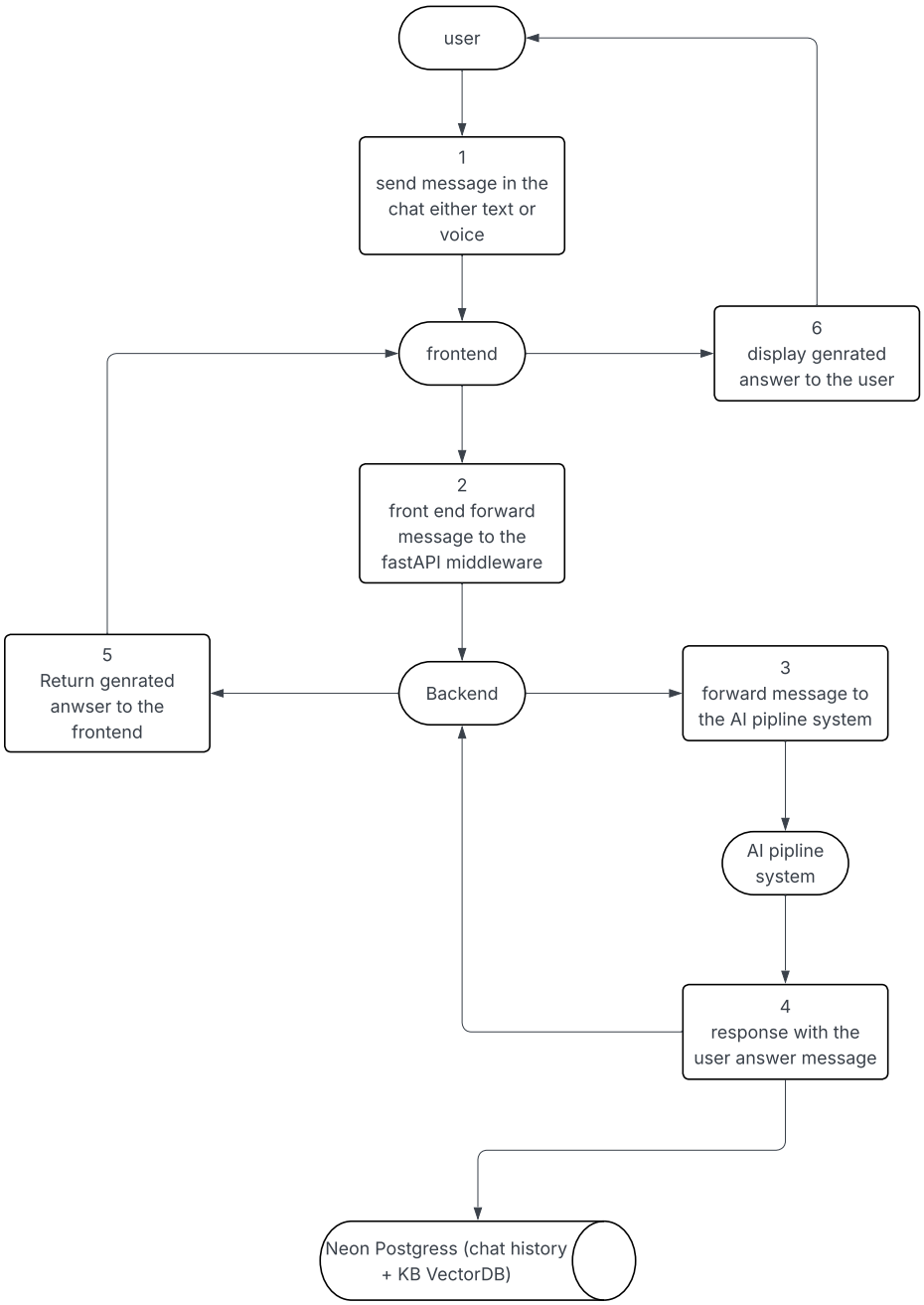


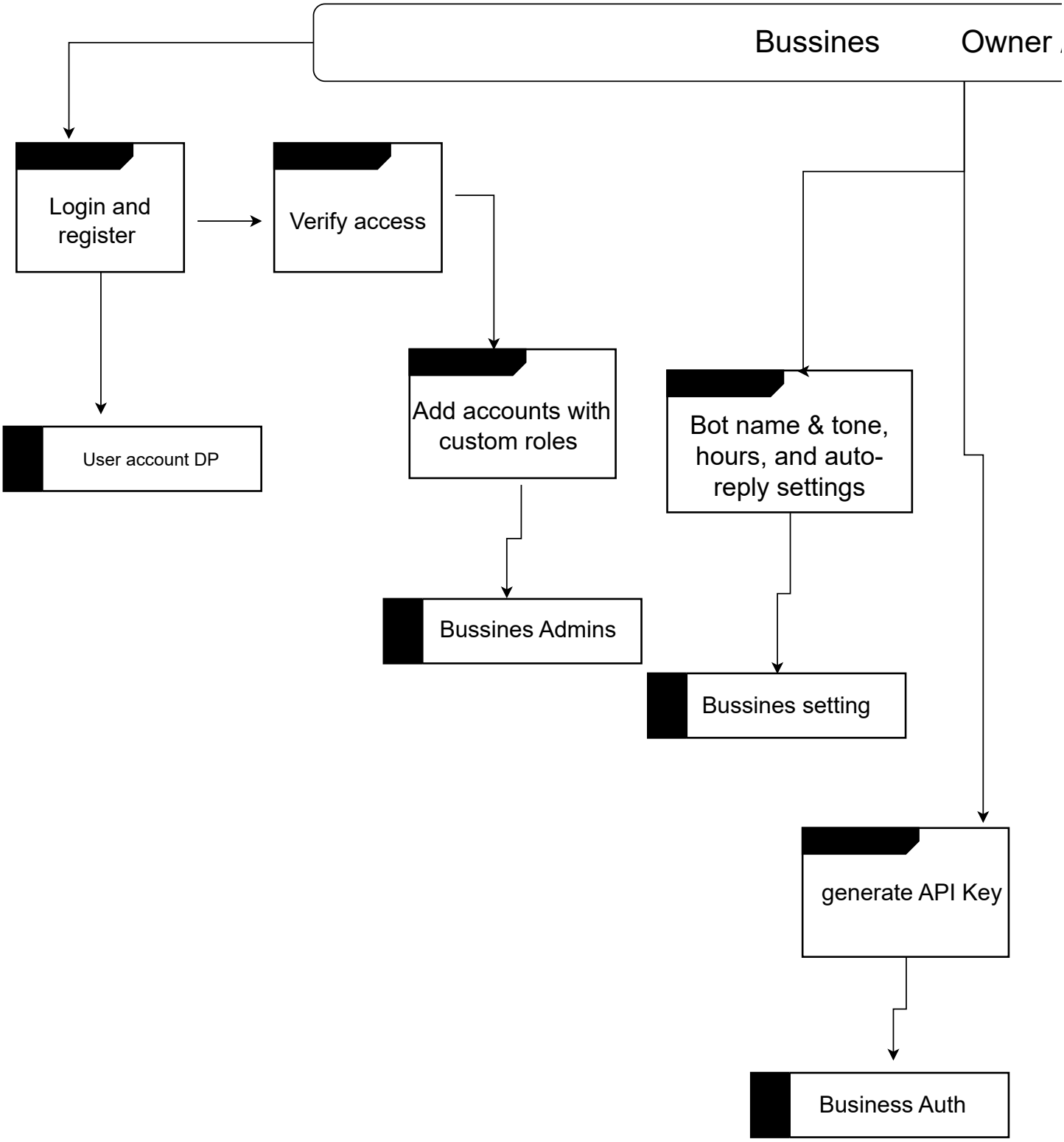

Controller Layer
Base Model
"NileChat or Gemini/Qwen APIs"
Fallback Model
"Nilechat or Gemini / Qwen APIs"
Formatting Layer
(Reformatter)
Neon Postgres
(Chat History + KB VectorDB)
Check similarity score & threshold
if
[High Confidence]
Send prompt + context
Raw answer
else
[Low Confidence]
Send prompt + context
Raw answer
Send raw answer + formatting style rules
Reformatted Egyptian Arabic response
Save query + response + emotion tag
Controller Layer
Base Model
"NileChat or Gemini/Qwen APIs"
Fallback Model
"Gemini / Qwen APIs"
Formatting Layer
(Reformatter)
Neon Postgres
(Chat History + KB VectorDB)

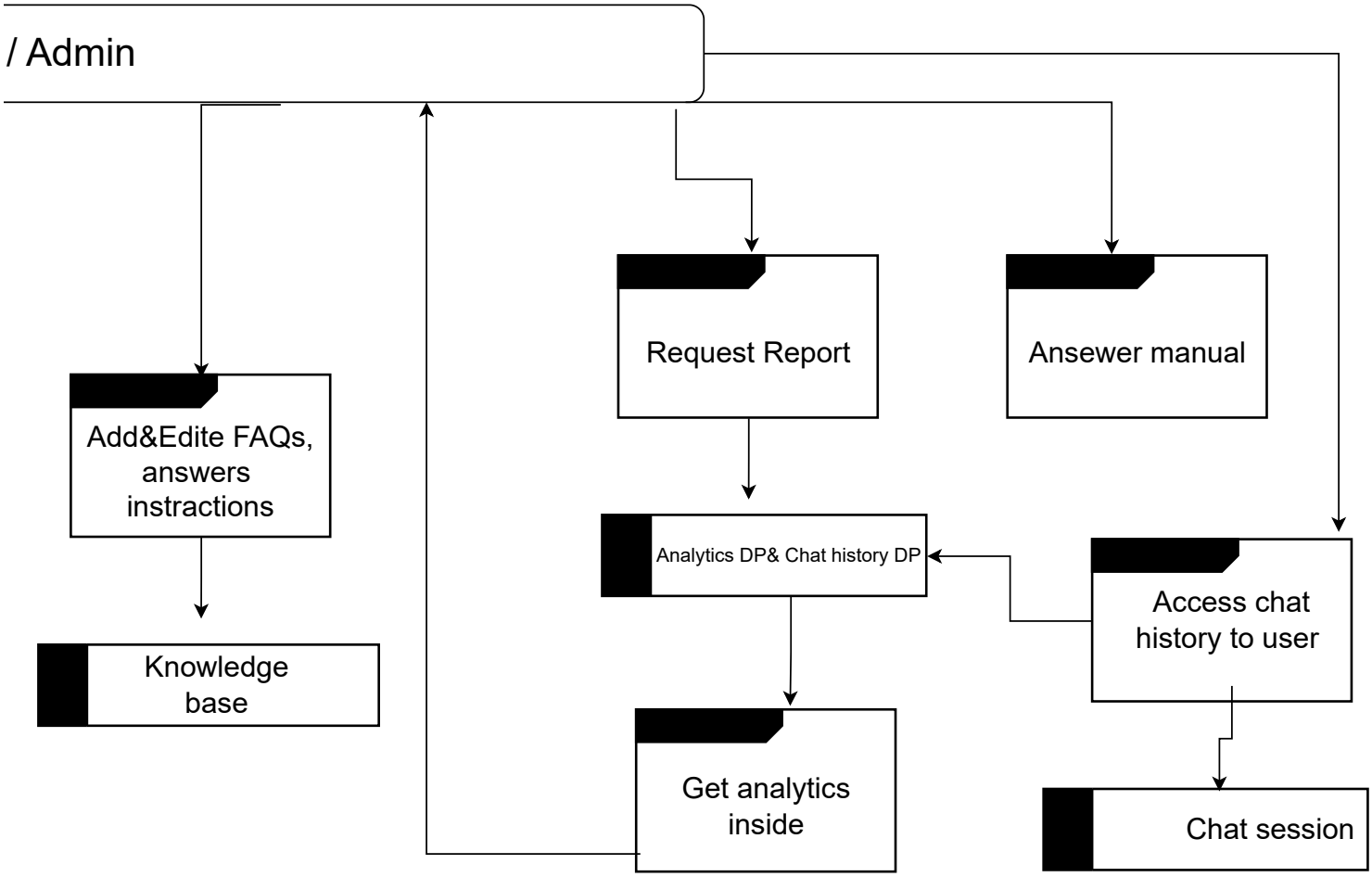
Sequence Diagram Business flow

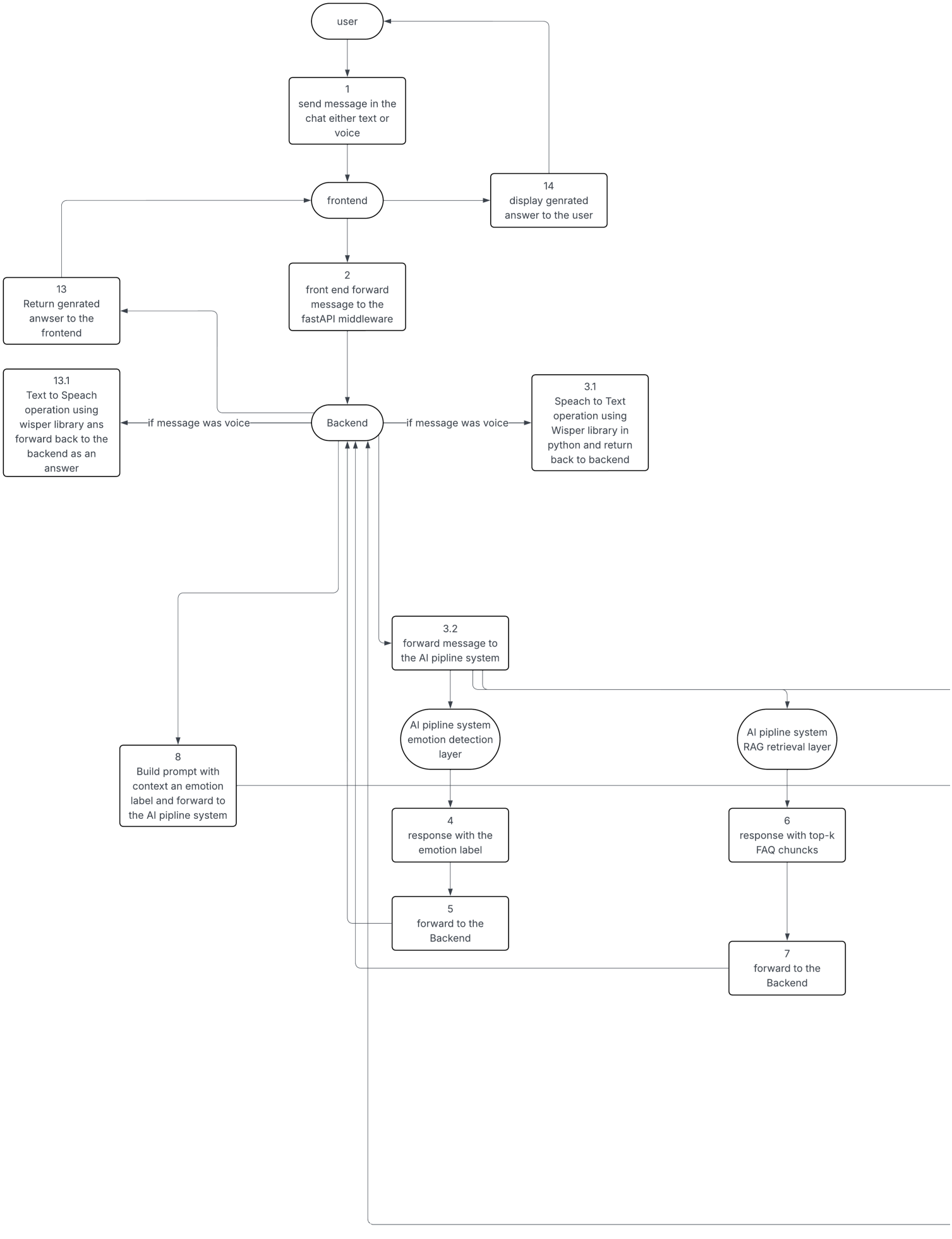


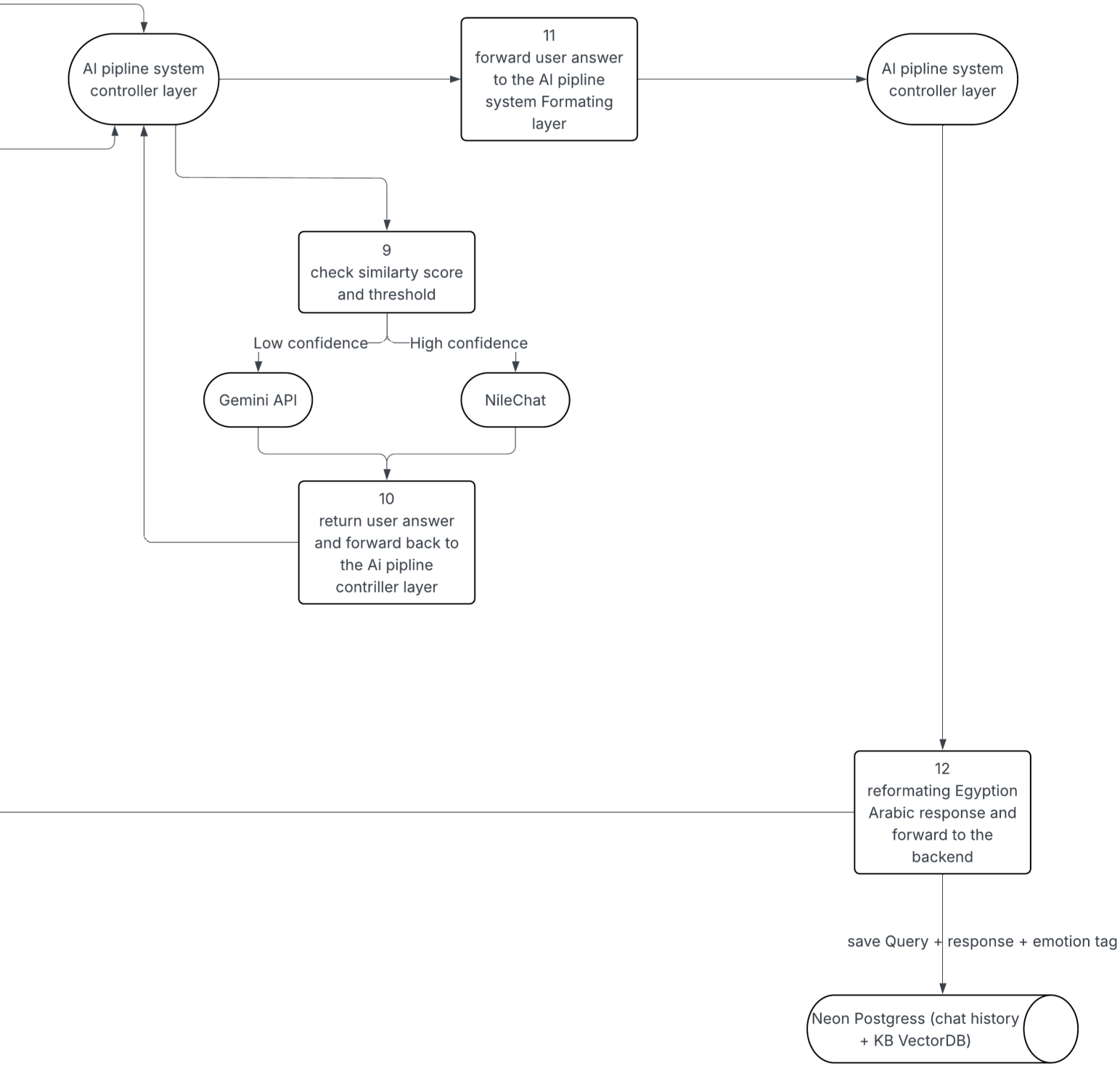

User
Send message
Send answer
Access / confirmation
/ analysis
Business
AI customer
support
request processing
AI pipline
system
login / mange KB /
request reports
Genrated response


user
1
send message in the chat either text or voice
frontend
6
display genrated answer to the user
2
front end forward message to the fastAPI middleware
5
Return genrated anwser to the frontend
Backend
3
forward message to the AI pipline system
AI pipline system
4
response with the user answer message
Neon Postgress (chat history + KB VectorDB)


Bussines Owner
Login and register
Verify access
User account DP
Add accounts with custom roles
Bussines Admins
Bot name & tone, hours, and auto-reply settings
Bussines setting
generate API Key
Business Auth


/ Admin
Request Report
Ansewer manual
Add&Edite FAQs, answers instractions
Analytics DP& Chat history DP
Access chat history to user
Knowledge base
Get analytics inside
Chat session


user
1
send message in the chat either text or voice
frontend
14
display genrated answer to the user
2
front end forward message to the fastAPI middleware
13
Return genrated anwser to the frontend
13.1
Text to Speach operation using wisper library ans forward back to the backend as an answer
Backend
if message was voice
if message was voice
3.1
Speach to Text operation using Wisper library in python and return back to backend
3.2
forward message to the AI pipline system
AI pipline system emotion detection layer
AI pipline system RAG retrieval layer
8
Build prompt with context an emotion label and forward to the AI pipline system
4
response with the emotion label
6
response with top-k FAQ chuncks
5
forward to the Backend
7
forward to the Backend


AI pipline system controller layer
11
forward user answer to the AI pipline system Formating layer
AI pipline system controller layer
9
check similarty score and threshold
Low confidence
High confidence
Gemini API
NileChat
10
return user answer and forward back to the Ai pipline contriller layer
12
reformating Egyption Arabic response and forward to the backend
save Query + response + emotion tag
Neon Postgress (chat history + KB VectorDB)

6. Functional Requirements Specification (FRS)

6.1 Core Actors

Actor	Description
End User (Customer)	A visitor or client interacting with the business chatbot via widget or Meta integration.
Business Admin	Registered business staff managing chatbot configurations, viewing analytics, and editing the knowledge base.
AI System	Orchestrator responsible for intent detection, emotion analysis, RAG retrieval, and response generation.
Backend API (FastAPI)	Middleware managing requests between user, AI models, and the database (PostgreSQL).
Database (Neon PostgreSQL + pgvector)	Stores all persistent entities: users, sessions, messages, knowledge base, and analytics reports.

6.2 Functional Requirements

6.2.1 User Interaction (Text and Voice)

- **FR1:** The system shall allow users to send **text messages** through the chat widget or Meta channels.
- **FR2:** The system shall support **voice input** (speech-to-text) and **voice output** (text-to-speech) via Whisper and TTS APIs (future).
- **FR3:** The system shall store each message or voice record in `ChatMessage` or `VoiceInteraction` with metadata such as AI model used, detected emotion, timestamp, content or transcription text, voice URL, and sender type.
- **FR4:** The system shall detect user **emotion** from the text or voice tone using an emotion detection layer.
- **FR5:** The chatbot shall **adapt its tone or response style** based on the detected emotion and business tone configuration.

6.2.2 Widget & Meta Integration

- **FR6:** The system shall provide an **embeddable chat widget** using a unique `api_key` from `BusinessAuth`.
- **FR7:** The widget shall support **real-time messaging**, **typing indicators**, and **session persistence**.
- **FR8:** The system shall verify the `api_key` before initializing any chat session.
- **FR9:** The system shall integrate with **WhatsApp**, **Messenger**, and **Instagram APIs** (planned feature) for businesses to connect their customer channels.
- **FR10:** The widget and integrations shall automatically switch to **offline mode** outside business hours (`operating_hour_start` / `operating_hour_end`).

6.2.3 Knowledge Base and AI Response

- **FR11:** The business admin shall be able to upload or input **FAQs, text, or files** into the `KnowledgeBase` (text or file-based).
- **FR12:** The AI pipeline shall generate **embedding vectors** for each entry using AI frameworks (e.g., LangChain) and store them in the `embedding_vector` field.
- **FR13:** When a user message is received, the system shall perform **contextual retrieval (RAG)** to fetch top-matching knowledge chunks.
- **FR14:** The AI controller layer shall choose the most confident model output between **NileChat**, **Gemini**, or **Qwen** based on similarity threshold.
- **FR15:** The system shall format AI responses based on business rules from `BusinessSetting`, consistently using a **response formatting layer**.

6.2.4 Business Registration and Authentication

- **FR16:** Businesses can register an account providing `name`, `email`, and `domain_url`.
- **FR17:** A default `BusinessSetting` and active `BusinessAuth` record shall be automatically generated upon registration.
- **FR18:** Each business can manage multiple `BusinessAdmin` accounts with roles (`owner`, `staff`).
- **FR19:** Authentication shall be done through secure login using email and password.
- **FR20:** The system shall allow owners to **generate, rotate, or deactivate API keys** without deleting the business record.

6.2.5 Dashboard Management

- **FR21:** The business admin shall access an **admin dashboard** via web (React.js).
- **FR22:** The dashboard shall allow management of:
  - Knowledge Base entries (CRUD)
  - Bot tone style and response mode (auto/manual)
  - Operating hours
  - Viewing chat history and feedback
- **FR23:** The dashboard shall show analytics reports such as total chats, average response time, and user satisfaction (from `AnalyticsReport`).
- **FR24:** The dashboard shall include a **manual reply mode**, allowing admins to respond directly to ongoing chats if AI auto-reply is disabled.

6.2.6 Chat Session Management

- **FR25:** When a user starts chatting, the system shall create a new `ChatSession` linked to the `EndUser` and `Business`.
- **FR26:** All messages within the same chat shall link to that session ID for history tracking.
- **FR27:** When a chat ends, the system shall mark the session as `closed` or `expired`.
- **FR28:** The user shall be prompted to submit **feedback** (rating and comment), stored in the `Feedback` table.

6.2.7 Analytics and Reporting

- **FR29:** The system shall automatically aggregate chat statistics and feedback to generate `AnalyticsReport` entries.
- **FR30:** Reports shall include total chats, average response time, and average user satisfaction.
- **FR31:** Reports shall be available for download or viewing in the dashboard.
- **FR32:** Analytics shall update periodically (e.g., daily or weekly background job).

6.2.8 Voice Interaction (Future Expansion)

- **FR33:** The system shall store voice recordings and transcription text in the `VoiceInteraction` table.
- **FR34:** The system shall detect emotion from voice tone and integrate results with AI response generation.
- **FR35:** The chatbot shall be able to **reply with customized voice** based on user preference or platform type (e.g., WhatsApp voice message).

6.2.9 AI Pipeline Orchestration

- **FR36:** The AI Orchestrator shall manage the workflow between input message → emotion detection → RAG → model selection → formatting → output.
- **FR37:** The orchestrator shall fallback between models if the primary fails.
- **FR38:** Each step of the pipeline shall log its operation (model used, latency, confidence score).

6.2.10 System Administration

- **FR39:** The system shall maintain logs of authentication attempts, API usage, and integration events.
- **FR40:** The system shall support environment-based configurations for development, staging, and production.
- **FR41:** The system shall notify admins of integration errors (e.g., API key invalid, channel disconnected).

7. Non-Functional Requirements

7.1 Performance

- **NFR-1:** Average chatbot response (text) shall be less than or equal to **5 seconds** for 90% of requests.
- **NFR-2:** Emotion detection latency shall not exceed **600 ms for text , 1000ms for voice**, and RAG retrieval shall not exceed **1.5 s for text , 3 s for voice**.
- **NFR-3:** Voice query processing (audio → text → reply) shall take less than 1 min

- **NFR-4:** Average database query latency shall not exceed **200 ms** under normal load.

7.2 Scalability

- **NFR-5:** The architecture shall support **horizontal scaling** using Docker and (Railway or Coolify) .
- **NFR-6:** The queue system (Redis "caching") shall buffer up to **2k concurrent jobs** without failure.

7.3 Reliability & Availability

- **NFR-7:** The system shall maintain an uptime of at least **95% per month**.
- **NFR-8:** Critical services shall automatically restart within **30 seconds** after failure.
- **NFR-9:** Database backups shall be taken **daily** and retained for **14 days**.
- **NFR-10:** Model fallback (NileChat $\rightarrow$ Gemini $\rightarrow$ Qwen) success rate shall be at least **95%**.

7.4 Security

- **NFR-11:** All traffic shall use **HTTPS/TLS 1.3**.
- **NFR-12:** Passwords shall be hashed using **bcrypt (cost $\geq$ 12)**, and API keys hashed using **SHA-256**.
- **NFR-13:** JWT access tokens shall expire in **24 hours**, and refresh tokens shall be valid for **7 days**.
- **NFR-14:** Accounts shall be locked after **5 failed login attempts within 30 minutes**.
- **NFR-15:** Sensitive data shall be encrypted at rest using **AES-256**.

7.5 Usability

- **NFR-16:** Chat widget shall load in less than or equal to **30 seconds**
- **NFR-17:** Dashboard shall be responsive for screens wider than **768 px**.
- **NFR-18:** System shall support both **Arabic (Egyptian dialect)** and **English**, with automatic RTL layout.
- **NFR-19:** At least **80% of users** shall be able to complete a chat without assistance

7.6 Maintainability & Testability

- **NFR-20:** Codebase shall be organized into modular components: `api`, `ai_pipeline`, `db`, `dashboard`, and `integrations`.
- **NFR-21:** Unit-test coverage shall be at least **70%** for backend components.
- **NFR-22:** The CI/CD pipeline (GitHub Actions) shall complete build and deployment in less than **8 minutes**.
- **NFR-23:** All APIs shall include auto-generated documentation using **Swagger** or **ReDoc** and manual postman doc for testing.

7.7 Observability

- **NFR-24:** JSON logs shall be generated for every API call and AI model latency.
- **NFR-25:** Critical system errors shall trigger alerts (via slack or email) within **5 minutes** of detection.

7.8 Privacy

- **NFR-26:** The system shall comply with **Egyptian Data Protection Law**.
- **NFR-27:** Minimal personal data shall be stored, and identifiers shall be hashed or pseudonymized.
- **NFR-28:** Businesses shall be able to export or delete their stored data.

8. Risk Analysis Plan

Category	Risk Description	Probability	Impact	Mitigation / Prevention
Technical	AI model integration (Gemini / GPT / NileChat) may fail or produce inconsistent outputs.	Medium	High	Create a fallback layer with multiple models and strong prompt formatting.
Data Quality	Poor or insufficient Egyptian Arabic datasets for training emotion detection or intent models.	High	High	Use open datasets, fine-tuning, and real testing with SMEs.
Performance	Chatbot may experience slow response time or overload under concurrent users.	Medium	Medium	Optimize backend performance using async, caching, and monitoring tools.
Security	Unauthorized access to business data or chat logs.	Medium	High	Implement JWT, encryption, and role-based access controls.
User Acceptance	Businesses may find customization complex or prefer competitors.	Medium	High	Simplify dashboard UI and emphasize SME-friendly pricing.
Financial	Budget or sponsorship delays.	Low	Medium	Use minimal-cost deployment options and open-source resources.
Ethical / Legal	Issues with data privacy and user consent.	Low	High	Ensure consent-based data usage and anonymize logs.

Table 1: Risk Analysis Plan for the AI Customer Support Platform

9. Security Implementation Plan

Security is integrated across all project phases and layers to ensure the chatbot platform remains safe, compliant, and resilient against real-world threats. The security design aligns with OWASP Top 10 and modern AI system protection standards.

9.1 Security Objectives

- Protect user and business data during storage, processing, and communication.
- Prevent unauthorized access or abuse through robust authentication and rate limiting.
- Ensure AI and API components cannot be exploited or injected with malicious inputs.
- Maintain system visibility and control through secure logging and monitoring.

9.2 Implementation Details

9.2.1 1. Authentication & Authorization

- Implement OAuth2 and JWT-based authentication for all users and businesses.
- Use short-lived tokens and refresh mechanisms to prevent token reuse.
- Enforce Role-Based Access Control (RBAC) for business admins and platform users.

9.2.2 2. Data Protection

- Enforce HTTPS/TLS for all communication (frontend, backend, APIs, integrations).
- Encrypt sensitive data (chat logs, credentials, business info) in PostgreSQL (pgcrypto).
- Use environment variables and secret management in CI/CD to protect API keys.

9.2.3 3. Input Validation & AI Security

- Validate and sanitize all user inputs with Pydantic schemas.
- Prevent prompt injection and ensure AI models only access intended data.
- Filter and review AI responses before displaying them to users.

9.2.4 4. API & Network Security

- Apply rate limiting and throttling on key endpoints to prevent abuse.
- Enforce CORS with domain whitelisting for integrations (WhatsApp, Facebook, Web).
- Use Docker container isolation and vulnerability scans (Trivy, Bandit).

9.2.5 5. Monitoring & Testing

- Enable structured security logging (Loguru) for authentication and API events.
- Mask sensitive data in logs.
- Automate OWASP ZAP and Bandit scans in GitHub CI/CD pipeline.
- Conduct periodic penetration testing and performance checks before release.

9.3 OWASP Top 10 Security Measures

9.3.1 1. Authentication & Session Security (A07: Identification & Authentication Failures)

Already covered: OAuth2 + JWT.
Add:

- Token rotation & short TTL (short-lived JWTs, refresh tokens stored securely).
- Device/session revocation — allow logout from all sessions.
- Secure password policy + account lockout on repeated failed logins.
- Use FastAPI's OAuth2PasswordBearer securely (no token in URL).

9.3.2 2. Access Control & Authorization (A01: Broken Access Control)

- Enforce RBAC (Role-Based Access Control) between businesses, admins, and normal users.
- Prevent access to chat histories or admin routes unless authorized.
- Use path- and object-level access control (e.g., users can only view their own chat logs).
- Validate all incoming API requests against permissions.

9.3.3 3. Data Protection & Privacy (A02: Cryptographic Failures)

- Encrypt sensitive data (e.g., chat logs, business info) in PostgreSQL using pgcrypto.
- Never store plain tokens, passwords, or API keys.
- Use `.env` for credentials (never hardcode).
- Force HTTPS for all communications, including AI API calls.

9.3.4 4. Injection & Input Validation (A03: Injection, A05: Security Misconfiguration)

- Sanitize all incoming user messages before processing (no injection into AI prompts).
- Use parameterized queries in PostgreSQL.
- Add input schema validation in FastAPI with Pydantic.
- Escape all user-provided content on the frontend (prevent XSS).
- Filter file uploads (if added later) by MIME type.

9.3.5 5. Prompt Injection & AI Layer Security (Emerging Category, OWASP LLM Top 10)

Since you're using LLMs (Gemini/Qwen):

- Implement input/output sanitization before passing data to AI.
- Restrict model access to only what's needed (no sensitive data in prompts).
- Add a safety layer that checks responses before returning them to users.
- Limit what the AI can access through LangChain (no unrestricted file or system calls).

9.3.6 6. API Security (A08: Software and Data Integrity Failures)

- Require API keys or OAuth2 for external integrations (WhatsApp, Facebook, etc.).
- Add Rate Limiting + Throttling per endpoint.
- Use CORS with strict domain whitelisting.
- Validate all payloads with schemas (avoid mass assignment vulnerabilities).

9.3.7 7. Dependency & Infrastructure Security (A06: Vulnerable & Outdated Components)

- Use Dependabot or Safety CLI to scan dependencies weekly.
- Regularly update FastAPI, React, and Hugging Face libraries.
- Keep Docker images minimal and use non-root users.
- Scan Docker images for vulnerabilities (e.g., Trivy).

9.3.8 8. Logging & Monitoring (A09: Security Logging & Monitoring Failures)

- Implement structured logging (e.g., Loguru in FastAPI).
- Log all login attempts, API errors, and suspicious activity.
- Mask sensitive data in logs (tokens, passwords).
- Set up alerting on failed login bursts or rate-limit triggers.

9.3.9 9. Server & Config Security (A05: Security Misconfiguration)

- Enforce Content Security Policy (CSP) in React.
- Disable directory listing and debug mode.
- Use secure headers (HSTS, X-Frame-Options, X-Content-Type-Options).
- Manage secrets securely (GitHub Actions Secrets + .env + Docker secrets).

9.3.10 10. Business Logic & Abuse Prevention (A10: SSRF, A04: Insecure Design)

- Validate that bots can't be abused to spam or flood messages.
- Limit AI response length and frequency.
- Validate business IDs and message origins.
- Use rate limiting per user/IP to stop abuse of free tiers or spam attacks.