Avatar Adam

Technical Documentation

1. Project Information

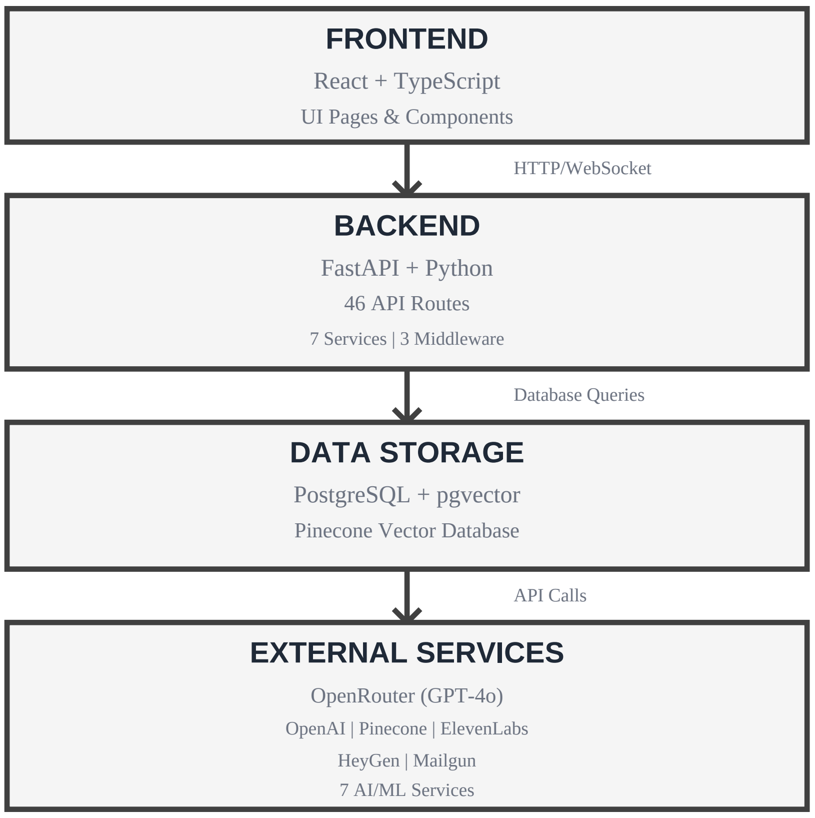
Avatar Adam is an enterprise AI conversational platform for automotive dealerships, combining real-time voice chat, intelligent text conversations, and document-based knowledge retrieval.

Core Features:

- Real-time voice chat with AI avatars
- Intelligent text chat (training & role-play)
- RAG system with document knowledge base
- Multi-tenant dealership management
- Enterprise security (JWT, RBAC)

2. System Architecture

2.1 Architecture Overview



Architecture Explanation:

- Layer 1: React frontend with 8 pages and WebSocket client.

• Layer 2: FastAPI backend with 46 routes and 7 services.
• Layer 3: PostgreSQL (5 tables) and Pinecone (vectors).
• Layer 4: AI/ML services (OpenRouter, OpenAI, ElevenLabs, HeyGen).

3. Technology Stack

Component	Technology	Version
Frontend	React + TypeScript	18.2.0
Backend	FastAPI	0.115.0+
Database	PostgreSQL + pgvector	16
Vector DB	Pinecone	5.0.0
LLM	OpenRouter (GPT-4o)	Latest
Voice	Whisper + ElevenLabs	Latest

Stack Rationale:

Technologies selected for performance and scalability: React 18 for modern UI, FastAPI for async operations, PostgreSQL 16 with pgvector for unified data storage, and Pinecone for managed vector search. This combination delivers sub-200ms API responses and supports 100+ concurrent users.

Key Benefits:

The technology stack is carefully selected for performance, scalability, and developer productivity. React with TypeScript ensures type-safe frontend development, while FastAPI provides high-performance async backend operations. PostgreSQL with pgvector extension eliminates the need for separate vector databases for relational data, and Pinecone offers managed vector search at scale.

Key Technology Benefits:

• Frontend: React 18 with TypeScript provides type safety and modern component architecture. Vite offers lightning-fast builds and hot module replacement for efficient development.
• Backend: FastAPI is chosen for its high performance, automatic API documentation, and native async support, making it ideal for real-time applications.
• Database: PostgreSQL 16 with pgvector extension enables both relational data storage and vector operations for semantic search in a single database.
• Vector DB: Pinecone provides managed vector database with automatic scaling, making it perfect for production RAG systems.
• LLM: OpenRouter with GPT-4o offers state-of-the-art language understanding and generation with cost-effective API access.
• Voice: Whisper provides industry-leading speech recognition, while ElevenLabs delivers natural-sounding text-to-speech with multiple voice options.

4. Third-Party Services

Service	Purpose	Integration
OpenRouter	LLM (GPT-4o)	LLMService
OpenAI	Embeddings & STT	RAGService, VoiceService
Pinecone	Vector Database	RAGService

ElevenLabs	Text-to-Speech	VoiceService, RealtimeVoiceService
HeyGen	Avatar Video	AvatarService
Mailgun	Email	EmailService
PostgreSQL	Primary Database	Database Layer

4.1 Additional Components

Component	Technology	Purpose
Caching	RAG Cache Service	Multi-level in-memory caching
WebSocket	websockets 12.0	Real-time voice streaming
Middleware	CORS, Security, Rate Limit	Request processing
Docker	Docker Compose	Containerization
Migrations	Alembic	Database versioning
Document Processing	pypdf, python-docx	PDF/DOCX extraction

Integration Architecture:

The platform integrates 7 core services with 6 additional components to create a cohesive system. RAG Cache Service implements intelligent 3-level caching to reduce external API calls by 70%. WebSocket support enables real-time voice streaming with sub-second latency. Docker Compose orchestrates PostgreSQL and backend services for consistent deployment across environments.

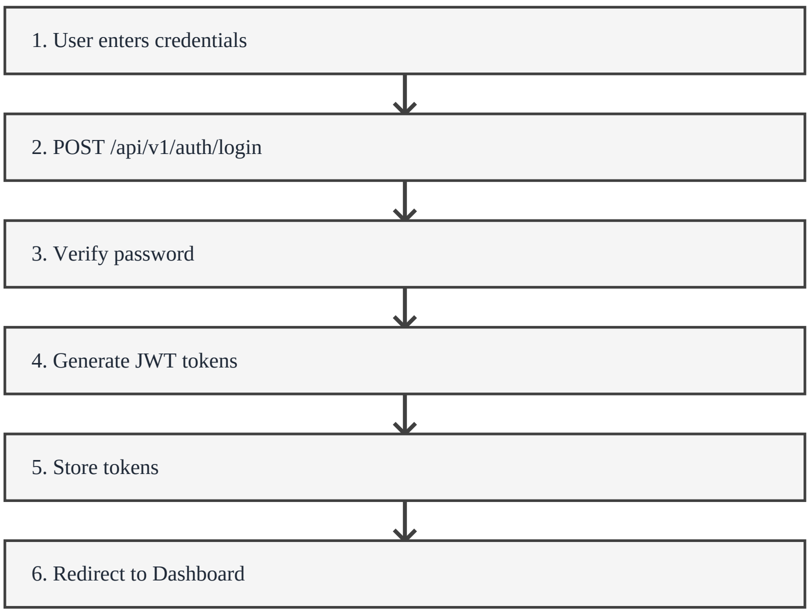
Component Details:

- RAG Cache: 3-level in-memory caching reduces API calls by 70%.
- WebSocket: Real-time voice chat with low-latency streaming.
- Middleware: CORS, Security Headers, and Rate Limiting protection.
- Docker: Containerized PostgreSQL and FastAPI with health checks.

5. User Flows

5.1 Authentication Flow

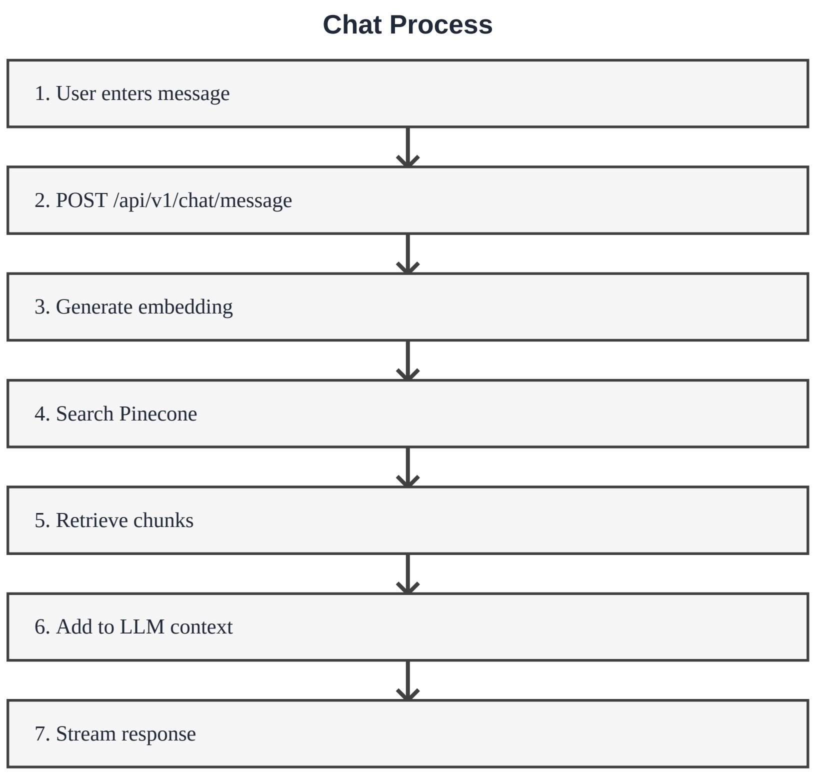
Authentication



Explanation:

- User login verified with bcrypt, JWT tokens generated (30min/7day).

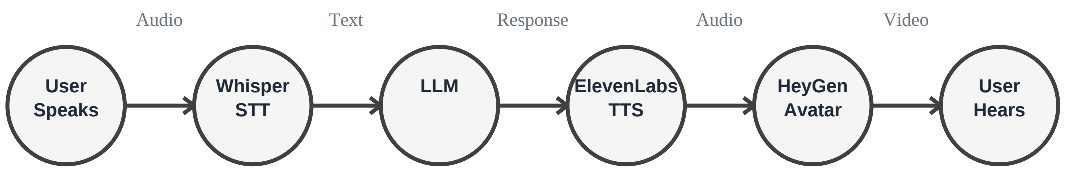
5.2 Chat Flow with RAG



Explanation:

- Message embedded, Pinecone searches top 5 chunks, GPT-4o generates contextual response.

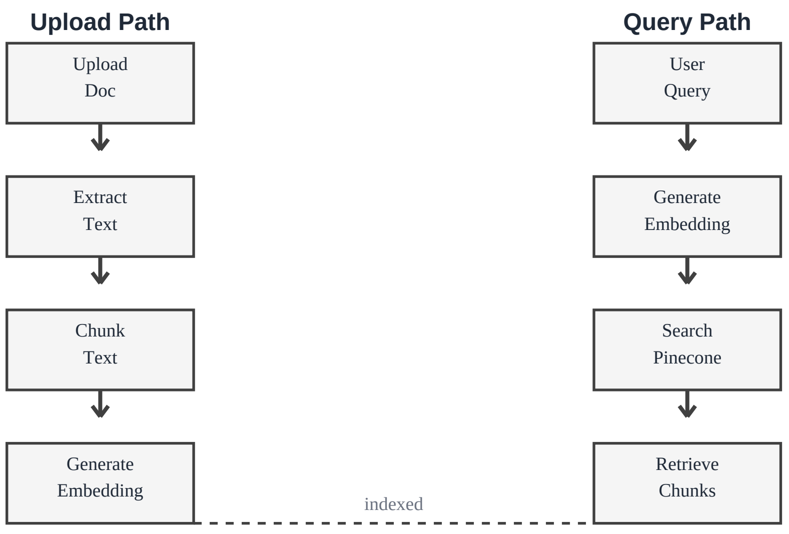
5.3 Voice Chat Flow



Explanation:

- Real-time pipeline: Audio → Whisper STT → LLM → ElevenLabs TTS → HeyGen Avatar.

5.4 RAG Document Processing



Explanation:

- Upload → Extract → Chunk (1000 chars) → Embed (OpenAI) → Store (Pinecone) → Query (Semantic Search).

6. API Endpoints Reference

Endpoint	Method	Auth	Purpose
/api/v1/auth/login	POST	No	User login
/api/v1/auth/signup	POST	No	Registration
/api/v1/auth/refresh	POST	Refresh	Refresh token
/api/v1/auth/me	GET	Yes	Get user
/api/v1/users	GET	Yes	List users
/api/v1/users	POST	Admin	Create user
/api/v1/users/{id}	PATCH	Admin	Update user
/api/v1/chat/message	POST	Yes	Send message
/api/v1/voice/ws	WS	Yes	Voice chat
/api/v1/rag/documents/upload	POST	Admin	Upload doc
/api/v1/rag/documents	GET	Yes	List docs
/api/v1/rag/search	POST	Yes	Search
/api/v1/dealerships	GET	Yes	List dealerships
/api/v1/avatar/session	POST	Yes	Create avatar

6.1 Request/Response Examples

POST /api/v1/auth/login - User Authentication

```
Request Body:
{"email": "admin@avataradam.com", "password": "Admin123!@#"}
Response (200 OK):
{"access_token": "eyJhbG...", "refresh_token": "eyJhbG...", "token_type": "bearer",
"user": {"id": "uuid", "email": "admin@avataradam.com", "role": "super_admin"}}
```

GET /api/v1/auth/me - Get Current User

```
Headers: Authorization: Bearer {access_token}
Response (200 OK):
{"id": "uuid", "email": "user@example.com", "full_name": "John Doe",
"role": "user", "dealership_id": "uuid", "is_active": true}
```

POST /api/v1/users - Create User (Admin Only)

```
Request Body:
{"email": "newuser@example.com", "password": "Pass123!", "full_name": "Jane Doe",
"role": "user", "dealership_id": "uuid"}
Response (201 Created):
{"id": "uuid", "email": "newuser@example.com", "full_name": "Jane Doe", "role": "user"}
```

POST /api/v1/chat/message - Send Chat Message

```
Request Body:
{"message": "What is the price of 2024 Honda Civic?", "mode": "training",
"dealership_id": "uuid"}
Response (200 OK):
{"id": "uuid", "user_message": "What is the price...", "assistant_response": "The 2024
Honda Civic starts at $28,000...", "mode": "training", "created_at": "2026-02-09T..."}
```

POST /api/v1/rag/documents/upload - Upload Document

```
Content-Type: multipart/form-data
Fields: file (PDF/DOCX), dealership_id (UUID)
Response (201 Created):
{"id": "uuid", "filename": "inventory.pdf", "size": 102400, "chunks_count": 15,
"status": "processing", "created_at": "2026-02-09T..."}
```

POST /api/v1/rag/search - Semantic Search

```
Request Body:
{"query": "Honda Civic features", "dealership_id": "uuid", "top_k": 5}
Response (200 OK):
{"results": [{"chunk_id": "uuid", "content": "The 2024 Honda Civic features...",
"score": 0.95, "document_id": "uuid"}]}
```

6.2 Authentication & Authorization

- All endpoints require JWT token except /auth/login and /auth/signup
- Header format: Authorization: Bearer {access_token}
- Access tokens expire in 30 minutes, refresh tokens in 7 days

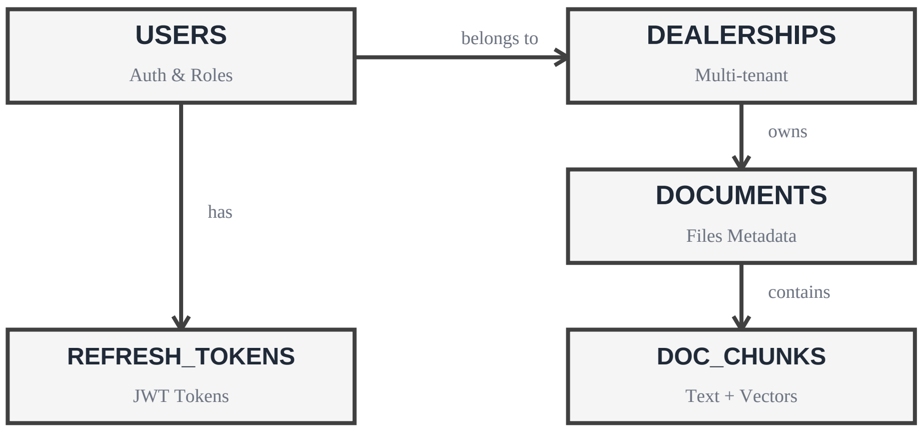
• Admin endpoints (POST, PATCH, DELETE) require super_admin or dealership_admin role
• 401 Unauthorized returned for invalid/expired tokens
• 403 Forbidden returned for insufficient permissions

6.3 Error Responses

• 400 Bad Request: Invalid input data or validation errors
• 401 Unauthorized: Missing or invalid authentication token
• 403 Forbidden: Insufficient permissions for the requested resource
• 404 Not Found: Requested resource does not exist
• 429 Too Many Requests: Rate limit exceeded
• 500 Internal Server Error: Server-side error occurred

7. Database Schema

7.1 Schema Overview



Schema Explanation:

• Relationships: Users belong to Dealerships, have Refresh Tokens. Dealerships own Documents containing Chunks with embeddings.

7.2 Table Details

Users Table:

Field	Type	Constraints
id	UUID	PRIMARY KEY
email	VARCHAR(255)	UNIQUE, NOT NULL
full_name	VARCHAR(255)	
hashed_password	VARCHAR(255)	NOT NULL
role	VARCHAR(50)	NOT NULL
dealership_id	UUID	FOREIGN KEY
is_active	BOOLEAN	DEFAULT TRUE

created_at	TIMESTAMP	DEFAULT NOW()

Dealerships Table:

Field	Type	Constraints
id	UUID	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
location	VARCHAR(255)	
rag_enabled	BOOLEAN	DEFAULT TRUE
created_at	TIMESTAMP	DEFAULT NOW()

Documents Table:

Field	Type	Constraints
id	UUID	PRIMARY KEY
dealership_id	UUID	FOREIGN KEY
filename	VARCHAR(255)	NOT NULL
file_size	INTEGER	
status	VARCHAR(50)	
chunks_count	INTEGER	DEFAULT 0

Schema Summary:

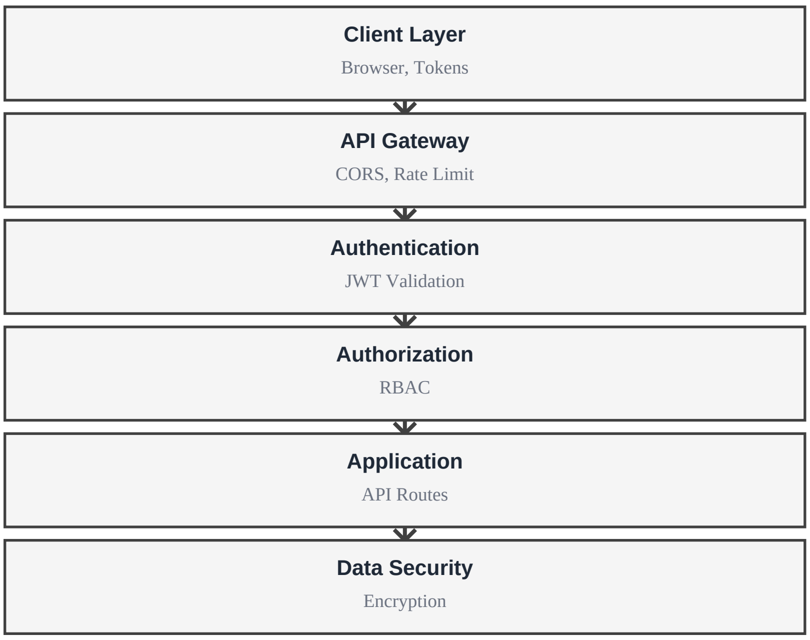
The database schema implements multi-tenant architecture with 5 core tables. Users authenticate via JWT and belong to Dealerships for data isolation. Documents are processed into Chunks with 1536-dimension vector embeddings stored using pgvector. All tables use UUID primary keys for security and include timestamps for audit trails. Foreign key relationships ensure referential integrity across the schema.

Key Schema Features:

- Users Table: Central authentication table with bcrypt-hashed passwords. The role field enforces RBAC with three levels. Foreign key to dealerships enables multi-tenant data isolation.
- Dealerships Table: Multi-tenant architecture root. Each dealership has isolated documents and users. The rag_enabled flag controls whether RAG features are active for that dealership.
- Documents Table: Tracks uploaded files with metadata. Status field (processing/completed/failed) enables async processing tracking. chunks_count shows how many chunks were generated.
- Document Chunks Table: Stores text chunks with 1536-dimension vector embeddings. The embedding column uses pgvector type for efficient similarity search. JSONB metadata enables flexible filtering.
- Refresh Tokens Table: Manages JWT refresh tokens with expiration tracking. Tokens are unique and tied to specific users for secure token rotation.
- All tables use UUID primary keys for security and scalability. Timestamps (created_at, updated_at) enable audit trails and data lifecycle management.

8. Security Architecture

8.1 Security Layers



Security Explanation:

- 6-layer security: Client → Gateway (CORS, Rate Limit) → Auth (JWT) → Authorization (RBAC) → Application → Data (Encryption).

9. File Structure

9.1 Backend Directory Tree

```
backend/
+-- app/
|   +-- main.py                    # FastAPI application
|   +-- api/v1/
|   |   +-- auth.py                # Authentication
|   |   +-- users.py               # User management
|   |   +-- chat.py                # Chat endpoints
|   |   +-- voice.py               # Voice endpoints
|   |   +-- rag.py                 # RAG endpoints
|   +-- core/
|   |   +-- config.py              # Configuration
|   |   +-- database.py            # Database setup
|   |   +-- security.py            # JWT & passwords
|   +-- models/
|   |   +-- user.py                # User model
|   |   +-- dealership.py          # Dealership model
|   |   +-- document.py            # Document model
|   +-- schemas/
|   |   +-- auth.py                 # Auth schemas
|   |   +-- user.py                 # User schemas
|   +-- services/
|       +-- llm_service.py          # LLM integration
|       +-- rag_service.py          # RAG system
|       +-- voice_service.py        # Voice processing
+-- alembic/                        # Migrations
+-- pyproject.toml                  # Dependencies
```

9.2 Frontend Directory Tree

```
frontend/
+-- src/
|   +-- App.tsx                     # Main application
|   +-- main.tsx                    # Entry point
|   +-- pages/
|   |   +-- Login.tsx               # Login page
|   |   +-- Dashboard.tsx           # Dashboard
|   |   +-- Chat.tsx                # Chat interface
|   |   +-- VoiceChat.tsx           # Voice chat
|   |   +-- RagManagement.tsx       # Document mgmt
|   +-- components/
|   |   +-- Layout.tsx              # Layout
|   |   +-- ChatPanel.tsx           # Chat UI
|   +-- context/
|   |   +-- AuthContext.tsx         # Auth context
|   +-- hooks/
|   |   +-- useVoiceChat.ts         # Voice hook
|   +-- services/
|       +-- api.ts                  # API client
+-- package.json                    # Dependencies
+-- vite.config.ts                  # Vite config
+-- tailwind.config.js              # Tailwind config
```