GoEZ

Health Intelligence App

Documentation

By: Divya Patel

1. Executive Summary

GoEZ is a unified health analytics platform that transforms raw Apple Health data into actionable insights through two complementary approaches:

- Visual Analytics – Interactive charts and trend visualization powered by DuckDB for fast analytical queries.
- Deep Analysis – AI-powered conversational health insights using LangGraph multi-agent workflows and MongoDB.

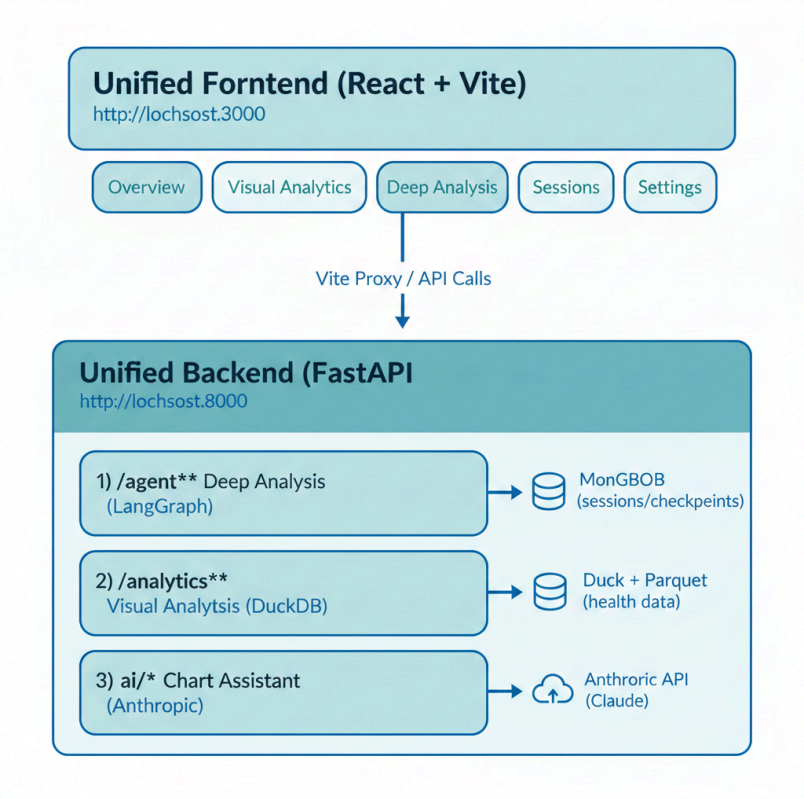
The platform is designed with a modern, scalable architecture that separates concerns between data processing, analytics, and AI reasoning while providing a unified user experience through a single-page React application.

2. System Overview

GoEZ follows a client-server architecture with a React-based frontend communicating with a unified FastAPI backend. The backend consolidates three distinct services:

Service	Route Prefix	Purpose
Deep Analysis	/agent/*	LangGraph-based conversational AI for in-depth health analysis
Visual Analytics	/analytics/*	DuckDB-powered metrics, trends, and chart data
Chart Assistant	/ai/*	Context-aware AI chat scoped to specific visualizations

Architecture Diagram



3. Technology Stack

3.1 Frontend Technologies

Technology	Version	Purpose
React	19.2	UI framework with latest concurrent features
TypeScript	5.9	Static type checking and developer experience
Vite	7.2	Fast build tool and development server
TailwindCSS	4.1	Utility-first CSS framework
React Router	7.1	Client-side routing and navigation
Zustand	5.0	Lightweight state management
Recharts	3.7	Declarative charting library
Framer Motion	11.18	Animation library
Lucide React	0.468	Icon library

3.2 Backend Technologies

Technology	Version	Purpose
Python	3.11+	Runtime environment
FastAPI	0.109+	Modern async REST API framework
Uvicorn	0.27+	ASGI server for production
LangGraph	0.2+	Agentic workflow orchestration
LangChain	0.3+	LLM abstraction and tooling
Anthropic SDK	0.18+	Claude AI model integration
DuckDB	0.10+	Embedded analytical database
PyMongo	4.6+	MongoDB driver
Pydantic	2.5+	Data validation and serialization
SSE-Starlette	1.8+	Server-sent events for streaming

3.3 Data & AI Technologies

Component	Technology	Purpose

Large Language Model	Claude (Anthropic)	Natural language understanding & generation
Agent Framework	LangGraph	Multi-step reasoning with state management
Analytics Database	DuckDB + Parquet	Fast columnar analytics over health data
Session Storage	MongoDB	Conversation history & LangGraph checkpoints

4. System Architecture

4.1 High-Level Architecture

The system employs a layered architecture that cleanly separates presentation, business logic, and data access concerns:

- Presentation Layer: React SPA with feature-based module organization
- API Gateway: FastAPI with CORS middleware and route prefixes
- Business Logic: LangGraph agents, analytics processors, AI orchestrators
- Data Access: DuckDB queries, MongoDB operations, Anthropic API calls
- Storage: Parquet files, MongoDB collections, in-memory caches

4.2 Frontend Module Architecture

```
frontend/src/
├── app/layout/          # AppLayout, Sidebar, Header components
├── modules/
│   ├── overview/        # Dashboard with quick metrics and actions
│   ├── visual_analytics/ # Interactive charts (Recharts-based)
│   ├── deep_analysis/   # AI conversation interface
│   ├── sessions/        # Analysis history browser
│   └── settings/        # Configuration and preferences
├── context/             # React context providers
└── shared/              # Reusable UI components
```

Key Design Patterns:

- Route-based code splitting – Lazy loading of modules via React Router
- Feature-based organization – Each module is self-contained with its own components
- Centralized state – Zustand stores for cross-cutting state (sessions, user prefs)

4.3 Backend Service Architecture

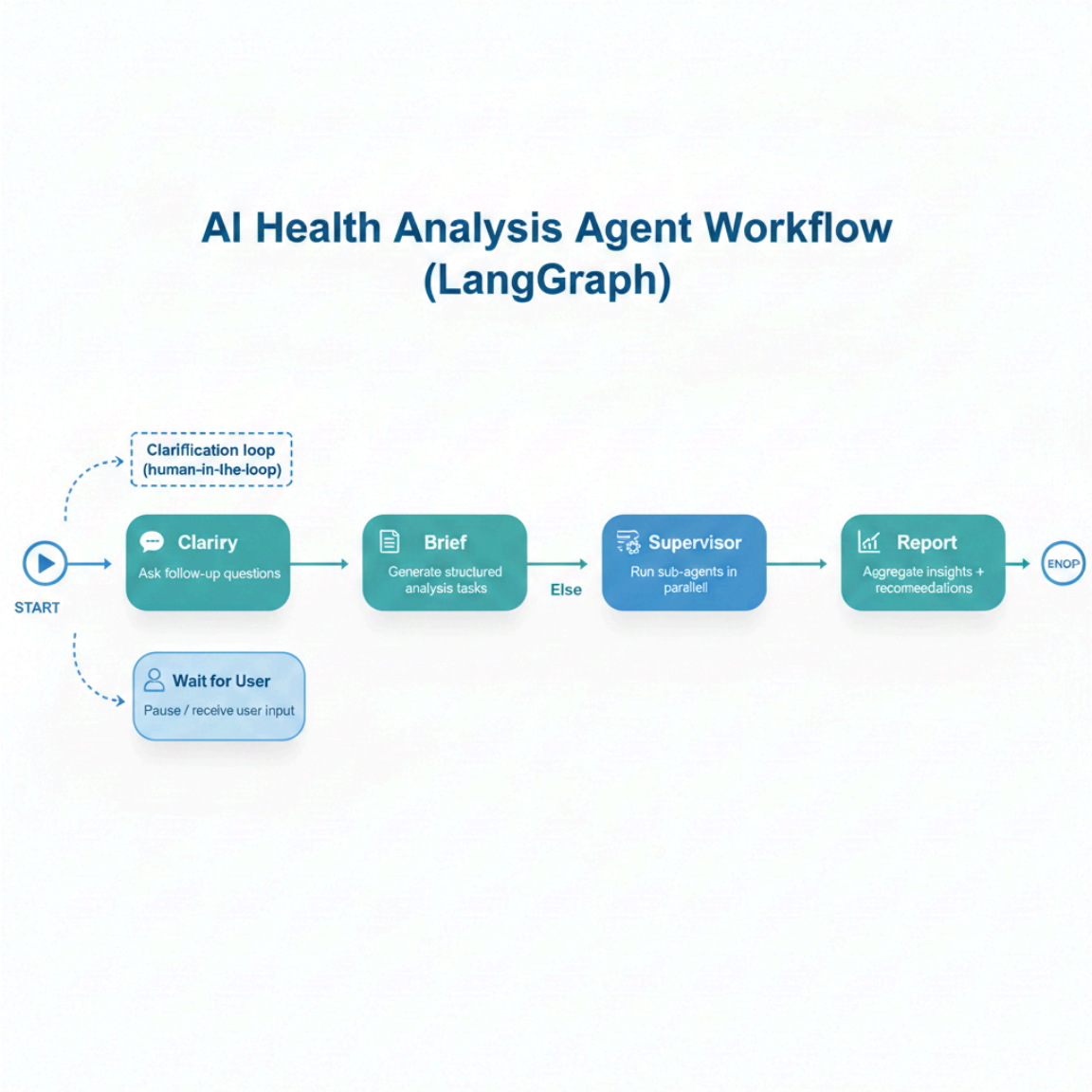
```
backend/
├── main.py              # Unified FastAPI entrypoint
├── agent/               # Deep Analysis Service
│   ├── graph.py         # LangGraph workflow definition
│   ├── state.py         # Workflow state schema
│   ├── nodes/           # Clarify, Brief, Supervisor, Report nodes
│   ├── subagents/       # Dynamic sub-agent factory
│   ├── prompts/         # System prompts and templates
│   └── memory.py        # MongoDB checkpointer
```

```
└── healthdata/          # Visual Analytics Service
    ├── ai/              # Chart-scoped AI assistant
    ├── analytics/       # DuckDB query builders
    ├── api/             # FastAPI routers
    └── ingest/          # Apple Health XML parser
```

5. AI Agent Design

5.1 LangGraph Workflow

The Deep Analysis feature implements a multi-phase agentic workflow using LangGraph. This approach enables complex reasoning with human-in-the-loop capabilities:



Phase	Node	Responsibility
1	Clarify	Understands user intent; asks follow-up questions if request is ambiguous

2	Brief	Generates structured analysis tasks with objectives and relevant fields
3	Supervisor	Spawns parallel sub-agents via ReAct pattern; aggregates results from data stored in MongoDB leveraging natural language to MQL
4	Report	Synthesizes findings into a coherent narrative with recommendations

5.2 Sub-Agent Factory Pattern

The Supervisor node dynamically creates specialized analysis agents based on task definitions. Each sub-agent is a ReAct agent with:

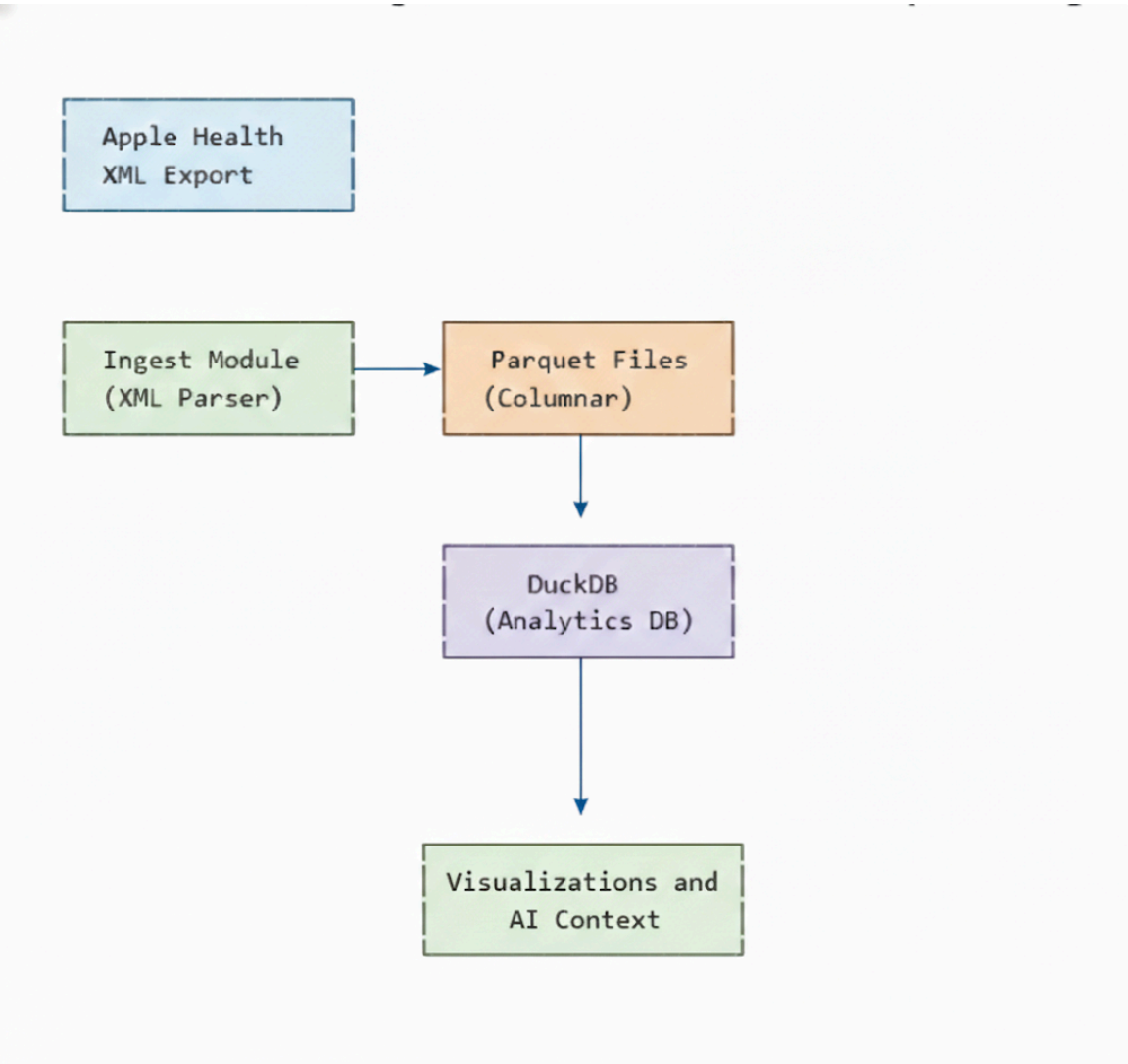
- MongoDB query tools from langchain_mongodb toolkit
- Task-specific system prompts with schema context
- Structured output parsing for metrics, trends, and anomalies
- Error handling with graceful degradation

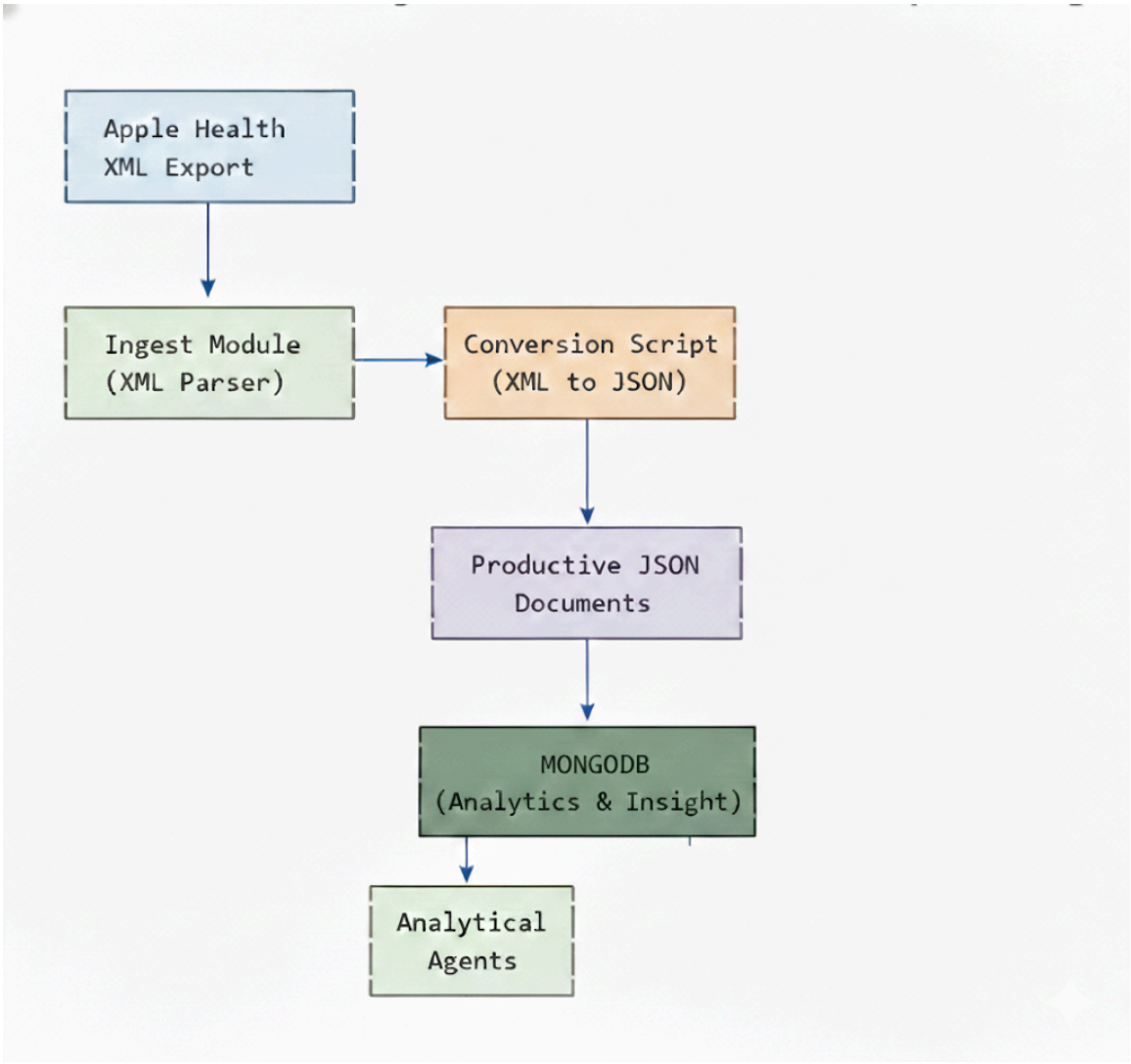
Sub-agents execute in parallel using Python's asyncio, with results aggregated by an LLM-powered aggregation step that identifies cross-metric patterns and correlations.

6. Data Flow

The system processes health data through the following pipeline:

1. Ingestion: Apple Health XML export is parsed and converted to Parquet files
2. Storage: Parquet files stored in data/parquet/; DuckDB database built for queries
3. Analytics: DuckDB executes fast columnar queries for charts and aggregations
4. AI Context: Chart-scoped context builder fetches relevant data slices for LLM
5. Persistence: MongoDB stores conversation history and LangGraph checkpoints




Apple Health
XML Export
Ingest Module
(XML Parser)
Conversion Script
(XML to JSON)
Productive JSON
Documents
MONGODB
(Analytics & Insight)
Analytical
Agents

7. API Design

The unified backend exposes RESTful endpoints organized by service domain. All endpoints follow consistent patterns for error handling and response formats.

Agent Endpoints (/agent/*)

Method	Endpoint	Description
POST	/agent/chat/stream	Streaming deep analysis chat with SSE
GET	/agent/sessions/:userId	List user's analysis sessions
GET	/agent/sessions/:userId/:sessionId/history	Get session message history
DELETE	/agent/sessions/:userId/:sessionId	Delete a session

Analytics Endpoints (/analytics/*)

Method	Endpoint	Description
GET	/analytics/metrics	List available health metrics
GET	/analytics/metric/daily	Get daily aggregated metric data
GET	/analytics/overview	Get dashboard overview tiles
GET	/analytics/scores	Get derived recovery/strain scores

AI Chat Endpoints (/ai/*)

Method	Endpoint	Description
POST	/ai/chat	Send message to chart-scoped AI assistant
GET	/ai/charts	Get list of available charts for AI chat
GET	/ai/chat/thread	Get conversation thread for a chart
DELETE	/ai/chat/thread	Clear conversation thread

8. Key Design Decisions

The following architectural decisions were made to optimize for performance, maintainability, and user experience:

Decision	Rationale
LangGraph over simple LLM chains	Enables interruptible workflows, human-in-the-loop clarification, parallel sub-agent execution, and state persistence across sessions
DuckDB for analytics	Embedded columnar database provides sub-second aggregations over millions of health records without infrastructure overhead
MongoDB for sessions and day to day data aggregation	Native LangGraph checkpoint support; flexible schema accommodates evolving conversation state
Unified FastAPI backend	Single deployment artifact reduces operational complexity; shared CORS and auth middleware
Chart-scoped AI context	Limiting context to relevant data reduces token usage by 60-80% and improves answer relevance
Parquet for raw storage	Columnar format enables efficient compression and fast analytical reads; schema evolution friendly

9. Future Enhancements

Goal Tracking: Set personal health goals with AI-powered progress monitoring

Wearable Sync: Direct integration with Garmin, Fitbit, and WHOOP

Predictive Insights: ML-based forecasting for recovery and performance trends

Mobile App: React Native companion for on-the-go access