Case Challenge: Smart Financial Coach

FinAI

By Anahita Dinesh

1. Project Overview

FinAI is a lang-chain based full-stack application designed to bridge the gap between raw financial data and actionable behavioral change. The application allows users to ask questions in plain English and receive automated, data-driven insights.

Github: https://github.com/anahita20/personal-finance-insights-langchain
Demo:https://vimeo.com/1160883209?share=copy&fl=sv&fe=ci

2. Technical Architecture

The system follows a decoupled architecture, separating data persistence, the AI orchestration layer, and the visualization.

2.1 Tech Stack

- Frontend: React.js with Tailwind CSS for a responsive UI and Recharts for dynamic data visualization.
- Backend: Flask (Python) serving as the API layer.
- AI Orchestration: LangChain for managing LLM prompts, entity extraction, and SQL generation.
- LLM: Google Gemini (via API) for natural language processing.
- Database: SQLite for local, lightweight relational data storage.

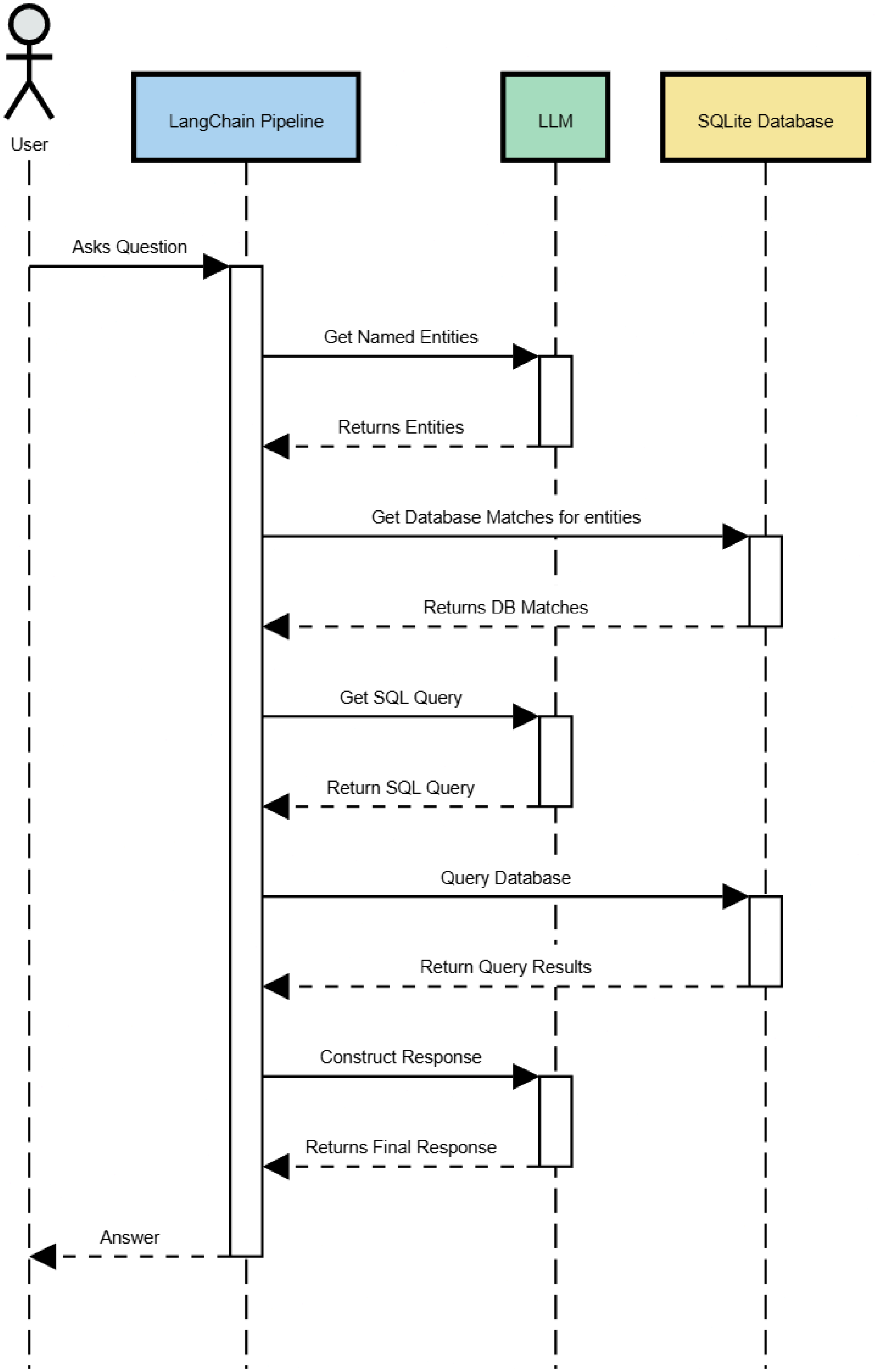
2.2 Functionality

The application handles two primary workflows: the AI Chat and the Automated Insights Dashboard.

AI Chat Pipeline

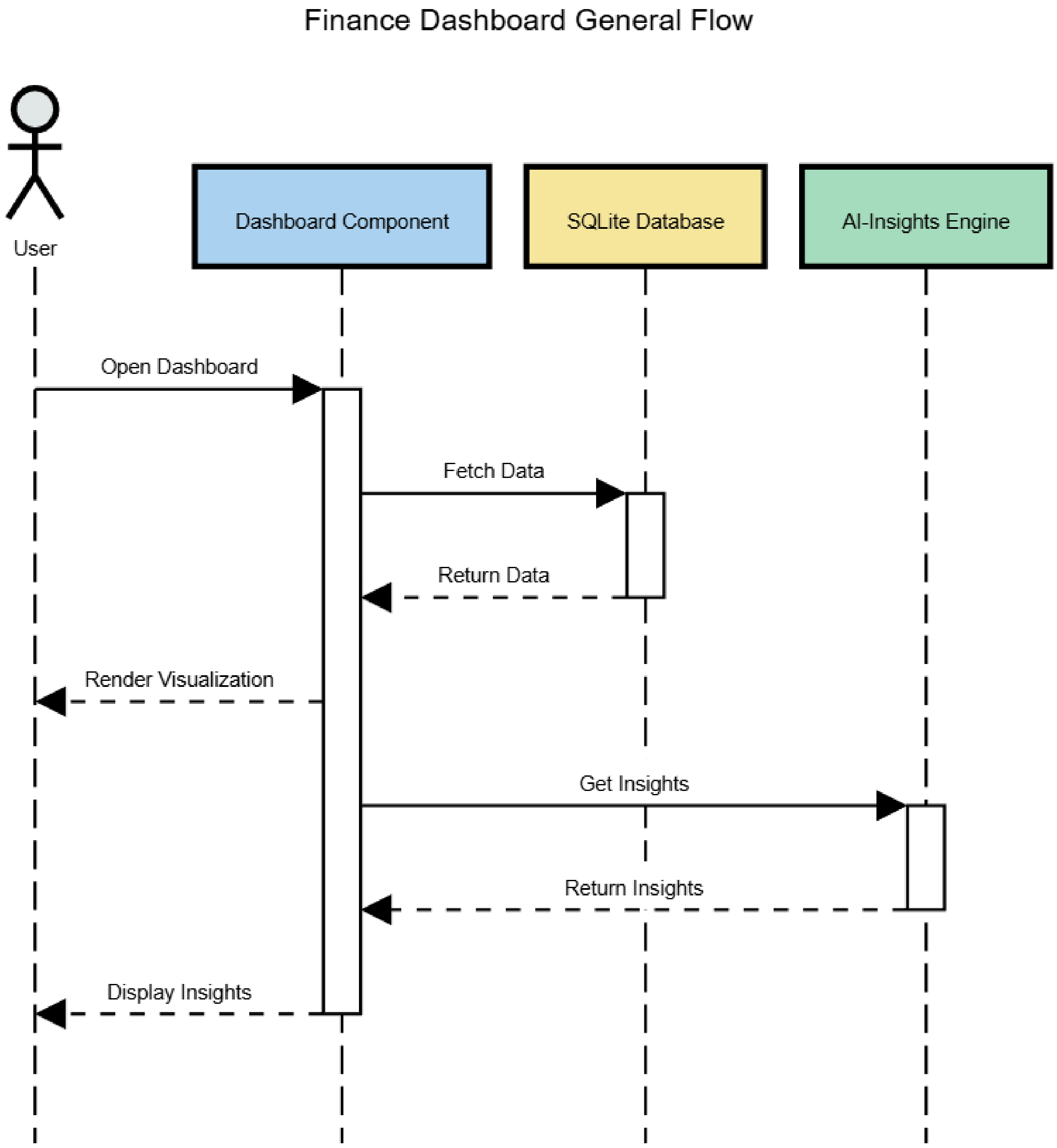
1. Entity Extraction: The LLM parses the user's natural language question to identify key entities (ex, coffee, last month, Starbucks).
2. Schema Mapping: The system matches extracted entities against the database schema and unique column values.
3. SQL Translation: LangChain prompts the LLM to generate a syntactically correct SQLite query based on the user's intent.
4. Data Execution: The query is executed against the SQLite database.
5. Natural Language Synthesis: The raw data results are passed back to the LLM to be formatted into a conversational, "coaching" style response.

Finance SQL Pipeline



Dashboard & Insights Flow

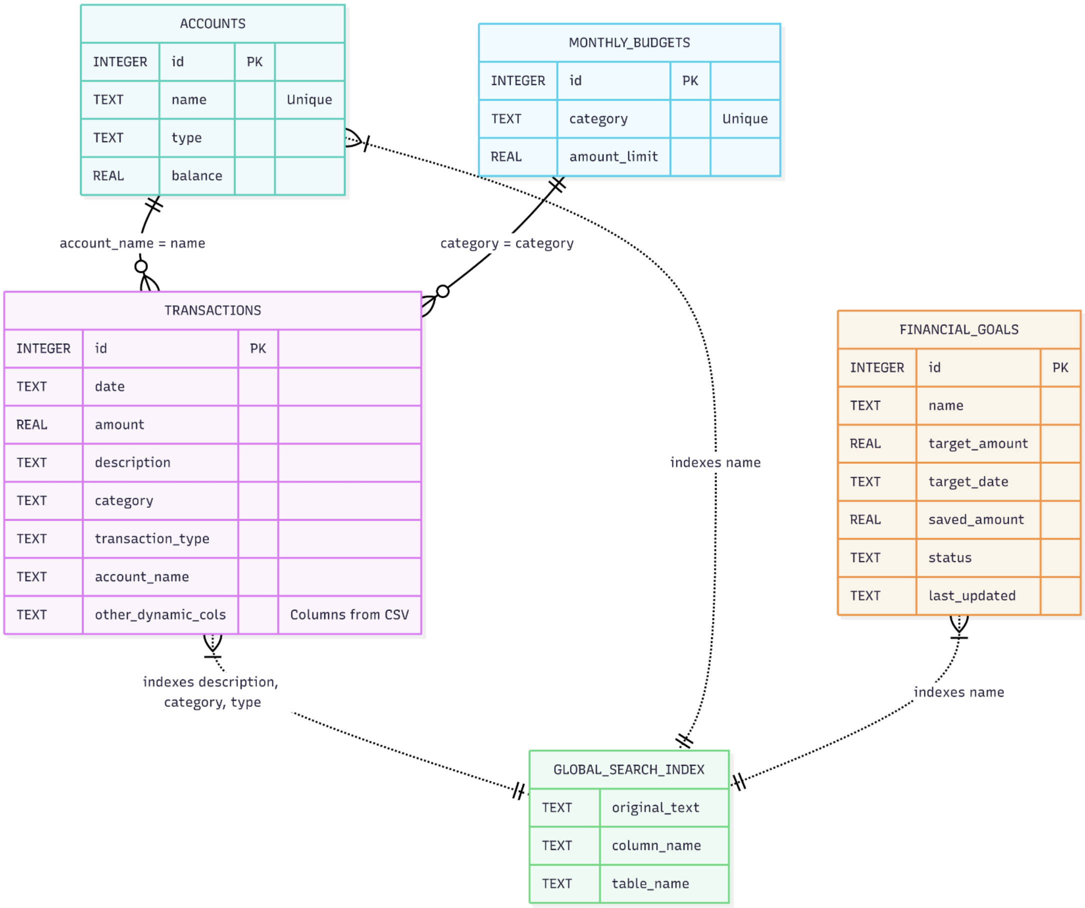
1. Data Retrieval: The dashboard fetches aggregated transaction data.
2. Visualization: React-Recharts renders spending trends and category breakdowns.
3. Anomaly Detection: The AI-Insights engine scans the returned dataset for patterns, such as "gray charges" (forgotten subscriptions) or spending spikes, returning them as proactive alerts.



3. Data Modeling

The system utilizes a relational schema to ensure strict financial integrity and high-performance aggregation.

1. ACCOUNTS: Stores account metadata and balances.
2. TRANSACTIONS: The core of the application revolves around the TRANSACTIONS table, which acts as a central ledger. It captures granular spend data. Includes other_dynamic_cols to support varied CSV imports from different banking institutions.
3. MONTHLY_BUDGETS: Defines spending limits per category for variance analysis.
4. GLOBAL_SEARCH_INDEX: A specialized index table used by the AI pipeline to map natural language terms to specific database records.



ACCOUNTS			
INTEGER	id	PK	
TEXT	name		Unique
TEXT	type		
REAL	balance		

MONTHLY_BUDGETS			
INTEGER	id	PK	
TEXT	category		Unique
REAL	amount_limit		

TRANSACTIONS			
INTEGER	id	PK	
TEXT	date		
REAL	amount		
TEXT	description		
TEXT	category		
TEXT	transaction_type		
TEXT	account_name		
TEXT	other_dynamic_cols		Columns from CSV

FINANCIAL_GOALS		
INTEGER	id	PK
TEXT	name	
REAL	target_amount	
TEXT	target_date	
REAL	saved_amount	
TEXT	status	
TEXT	last_updated	

GLOBAL_SEARCH_INDEX	
TEXT	original_text
TEXT	column_name
TEXT	table_name

Design Decision: SQL vs. Graph

While Graph databases excel at relationship mapping, a Relational (SQL) approach was chosen for:

- ACID Compliance: Ensuring that balance updates and transaction logs are atomic and consistent.
- Aggregation Speed: SQL is natively optimized for calculating sums, averages, and month-over-month variances across thousands of rows.

4. Key Features & Implementation

1. Natural Language Querying (NLQ): Users can ask, "How much did I spend on dining out in January?" without needing to navigate complex filters.
2. Subscription Detection: The system identifies recurring transaction descriptions with consistent amounts to flag potential "gray charges."
3. Goal Forecasting: By analyzing historical burn rates against financial goals, the AI predicts the likelihood of reaching a savings target by a specific date.

5. Security & Trust

- Local Processing: The use of SQLite allows for local data persistence, minimizing the footprint of sensitive financial records.
- API Safety: The system utilizes environment variables (.env) for API key management, ensuring credentials are never hardcoded or exposed.

6. Future Enhancements

- Vector Embeddings: Implementing a vector store for semantic search of transaction descriptions to improve entity matching.
- Multi-User Support: Migrating from SQLite to PostgreSQL for multi-tenant capabilities.