Design Document

94-815 Agent Based Modelling and Agentic Technology

Team 8 - Fanxing Bu, Ivan Wiryadi

---

1. System Overview

Sticklet is a multimodal, agent-based financial portal that allows users to upload and analyze receipts, view spending trends, and get financial market insights. Built using LangChain, Mistral and OpenAI, it follows a modular multi-agent architecture.

2. Agent Roles and Responsibilities

Agent Name	Role and Capabilities
Coordinator Agent	Orchestrates the pipeline: routes inputs, queries tools, delegates to other agents.
Receipt Reader Agent	Extracts structured data from receipt images via OCR and parsing tools.
Monthly Report Agent	Analyzes past purchases and generates user-friendly financial summaries.
Market Agent	Retrieves market data and generates contextual insights linked to spending behavior.

3. Workflow Overview

1. **Upload Receipt** → Coordinator routes to Receipt Reader → OCR + Parsing → Stored in Memory.
2. **Monthly Analysis** → Report Agent fetches history → Generates summary & graphs.
3. **Market News** → Market Agent retrieves financial data → Generates narrative summary.
4. **User Query** → Coordinator + Insight tools → Personalized answer based on database & memory.

4. Design Patterns Implemented

4.1 Overview

Pattern	Application in Sticklet
Tool/Agent Registry	Coordinator lazily initializes and fetches agents and tools.
Role-Based Coordination	Specialized agents for OCR, reporting, market, etc.
Human Reflection	Streamlit UI allows users to correct extracted receipt data and pass that to the agent.
Self-Reflection	Receipt Agent evaluates merchant name, category, date logic.
Retrieval Augmented Gen	Agents access SQLite DB for personalized financial insights.

4.2. Tool/Agent Registry Pattern

The Tool/Agent Registry pattern is implemented in the `CoordinatorAgent` class which maintains a centralized registry of specialized agents and tools. The coordinator uses `_initialize_agents()` and `_get_agent()` methods to lazily initialize and retrieve specialized agents only when needed. Each specialized agent also has access to its required tools - for example, `ReceiptReaderAgent` has OCR and parsing tools, while `MonthlyReportAgent` has access to SQL tools. This pattern enables resource conservation, provides a clean API for agent access, and facilitates modular development.

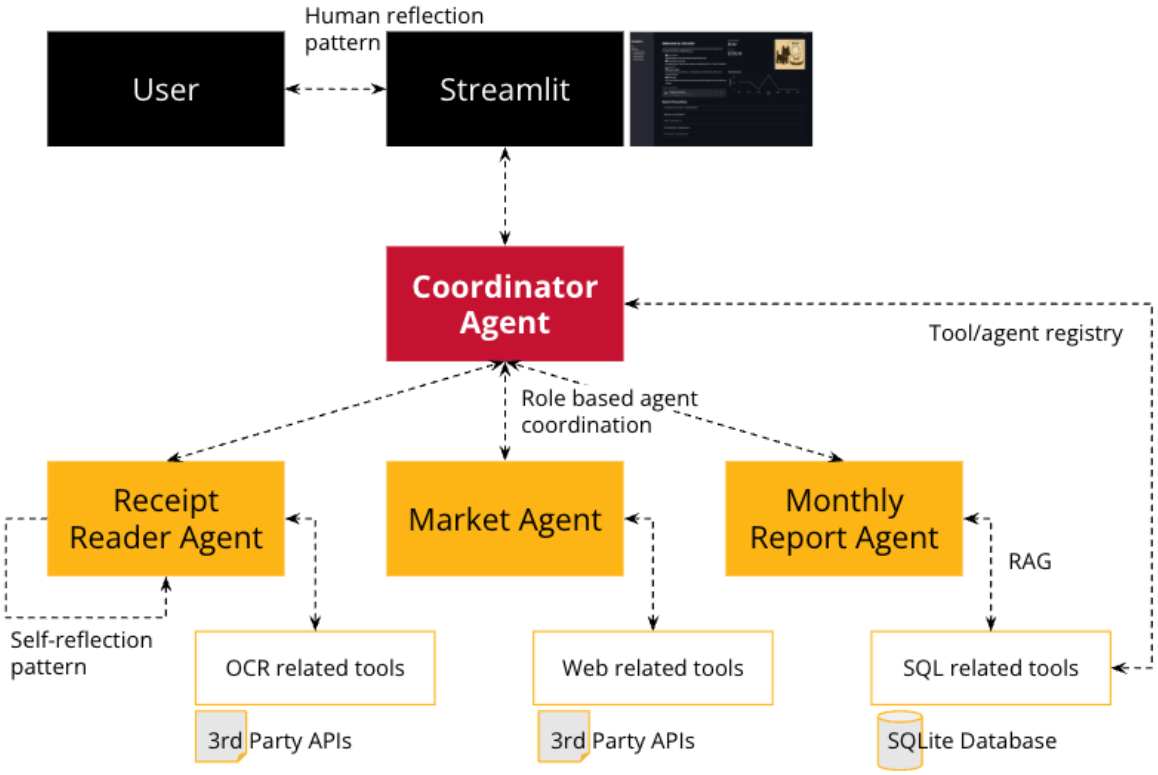

Figure 1: architecture

4.3. Role-Based Agent Coordination Pattern

The Role-Based Agent Coordination pattern distributes specialized tasks to purpose-built agents while the `CoordinatorAgent` orchestrates their activities. Each agent has a distinct role:

- `ReceiptReaderAgent`: Extracts structured data from receipt images
- `MonthlyReportAgent`: Generates financial reports based on spending history
- `MarketAgent`: Retrieves and analyzes market information

The coordinator delegates specific tasks to these specialized agents through methods like `process_receipt()`, `gen_monthly_report()`, and `get_market_indicators()`. This pattern reduces complexity through separation of concerns, allows for specialized optimizations in each agent, and enables parallel development of different agent capabilities.

4.4. Human Reflection Pattern

The Human Reflection pattern in Sticklet enables users to validate and correct AI-processed information through the Streamlit interface. In `app.py`, users can review extracted receipt data (merchant name, date, items, prices), make corrections, and confirm before saving to the database. This pattern increases accuracy by allowing humans to correct mistakes in AI processing, which is particularly important for financial data where errors could lead to incorrect insights.

4.5. Self-Reflection Pattern

The Self-Reflection pattern is implemented in the `ReceiptReaderAgent` through the `_reflect_on_results()` method, which validates and potentially corrects extracted data. When processing receipts, the agent extracts data and then "reflects" on whether the results make sense. For example, it identifies generic merchant names like "receipt" or "store" and attempts to find more specific merchant names from the raw text. It also validates dates and item categories. This pattern improves data quality by identifying and correcting common errors, reducing the need for human intervention, and learning from past mistakes.

4.6. Retrieval Augmented Generation (RAG) Pattern

The RAG pattern enhances the agent's knowledge with external data from the SQLite database. The system uses `PurchaseMemory` to store structured purchase data and provides tools like `SQLQueryTool` to query this database. When users ask questions about their spending patterns, the agent can retrieve relevant transaction data to provide personalized responses rather than relying solely on its pre-trained knowledge. This pattern enables the foundation model to access domain-specific knowledge, provide personalized responses based on user data, and perform temporal analysis of spending patterns without needing to retrain the model.

5. Technology Stack

- **LangChain (Python)**: For agent orchestration, memory, and tool usage.
- **OpenAI GPT-4/Mistral**: For reasoning, summarization, and fallback LLM.
- **Mistral OCR**: For multimodal receipt understanding.
- **SQLite**: For persistent purchase history.
- **Streamlit**: For interactive user interface.

GitHub Repo

https://github.com/LoadingBFX/Sticklet/