SpendWise

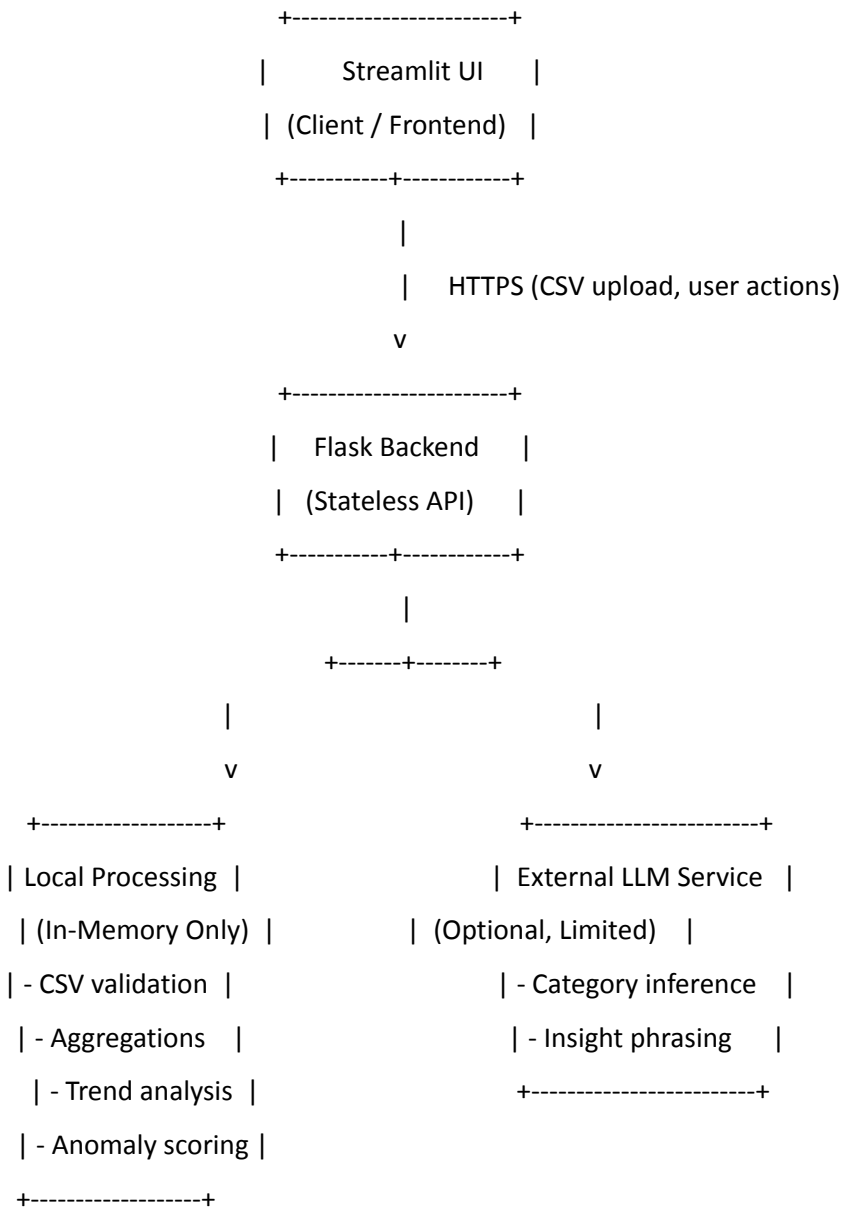
1. Overview

SpendWise is an AI-powered financial analysis tool designed to help users understand their spending patterns, categorize expenses, detect anomalies, and receive goal-based financial recommendations. It offers a streamlined experience that turns raw transaction data into meaningful insights that users can act on.

SpendWise is designed with a privacy-first approach due to the sensitivity of financial data. The system uses stateless processing, analyzing uploaded transactions in memory which minimizes data retention and reduces security risk. When external LLMs services are used, SpendWise sends only anonymized, derived data and never user identifiable details. This design intentionally limits certain advanced features, like long-term habit prediction, which rely on persistent, data-heavy user profiles.

2. System Architecture

2.1 High-Level Architecture



2.2 Architectural Flow

1. Users upload transaction data via the Streamlit UI.
2. The Flask backend validates and processes the data entirely in memory.
3. Core analytics (totals, trends, anomalies, subscriptions) are computed locally.
4. When semantic understanding is required (e.g., category inference or insight phrasing), only derived, anonymized data is sent to an external LLM.
5. Aggregated results are returned to the UI; no raw data is persisted.

2.3 Design Principles

Privacy by Default

- The backend is stateless; uploaded data exists only for the lifetime of the request and is discarded immediately after analysis.
- No raw transactions, merchant names, or user details are stored or logged.
- This minimizes data exposure and reduces potential security issues.

Controlled External Dependency Usage

- External LLMs are used only where deterministic logic is insufficient, such as semantic categorization or natural-language insight generation.
- LLM inputs are restricted to aggregated or abstracted features, never raw financial records.
- The system continues to function meaningfully even if external services are unavailable.

Modularity & Separation of Concerns

- Validation, analytics, anomaly detection, and insight generation are implemented as separate service modules.
- This structure improves testability, limits coupling, and allows individual components to evolve independently.

Secure-by-Design Trade-offs

- Advanced features that require long-term behavioral modeling or persistent user profiling are intentionally excluded from the MVP.
- This trade-off prioritizes user trust, transparency, and data minimization over speculative or high-risk personalization.

3. Tech Stack

- Python 3.11
- Streamlit (frontend)
- Flask (backend API)
- Pandas (data processing)
- Plotly (visualization)
- OpenAI GPT-4o-mini (categorization & financial recommendations)
- PyTest (testing)

4. Features Implemented

- CSV ingestion with validation.
- Editable category manager with prioritization.
- Monthly spending trend analysis.
- Automatic subscription identification.
- LLM categorization with fallback mode.
- Anomaly detection using z-score.
- Goal-based recommendations using LLM.
- Basic backend unit tests.

5. Security & Privacy Considerations

- No data is stored; the system operates entirely in-memory during each request.
- All possible computations are performed locally so sensitive data never leaves the controlled environment.
- Only merchant names and category lists, which are non-sensitive, are sent to LLM for semantic classification and insight generation.
- OpenAI usage is strictly constrained with predefined system prompts, controlled temperature settings, and limited input formats to prevent misuse and ensure consistent, safe behavior.

6. AI/ML Application

- GPT-4o-mini used for deterministic merchant classification.
- Strict system prompt prevents hallucinations.
- Fallback mechanism assigns random categories if OpenAI fails.
- Goal optimization engine suggests cutbacks based on category priorities.
- Anomaly detection uses statistical z-score, fully local.

7. Future Enhancements

7.1 Behavioral Change

-Integrate with secure bank aggregation APIs (e.g., Plaid-style providers) using tokenized, read-only access.

- Support an OCR-based, on-device extraction workflow for PDF bank statements, allowing users to import financial data without granting direct bank access.
- Allow users to configure more aspects of the analysis and insights, such as setting thresholds, adjusting priority levels to more than the current 3 levels.
- Enable users to give simple feedback to the LLM (e.g., “this category is wrong”), improving future categorization.
- Support multiple financial goals, not just savings, such as planning for vacations or reducing specific categories.
- Provide progress tracking that compares user behavior against personal goals over time.

7.2 Financial Visibility

- Introduce more intuitive visualizations such as heatmaps, per-merchant trend charts, and priority-based breakdowns.
- Improve layout and offer a more customizable dashboard where users can choose what insights they want to see.
- Add scenario simulations where users can test how reducing a certain category affects total savings.
- Provide more detailed drill-down views that allow users to explore categories, months, and merchants individually.

7.3 Trust & Security

- Secure API communication between frontend and backend with HTTPS and proper request validation.
- Explore replacing OpenAI with a small local language model for categorization to avoid external data transmission.
- Offer a transparency screen showing what data is used, what data leaves the system, and how insights are generated.
- Add optional encryption for temporary data held in memory during processing.
- Provide stricter guardrails on AI prompts to avoid unintended outputs and ensure consistent behavior.

7.4 AI/ML Application

- Introduce a small local LLM or rule-based classifier to replace external semantic categorization fully.
- Offer alternative anomaly detection models beyond z-score, such as simple local regression or rolling averages.
- Add lightweight forecasting models to estimate next month's spending based on historical data.
- Combine statistical insights with LLM reasoning to create more personalized, context-aware recommendations.

7.5 Technical Consideration

-Use a relational lightweight database such as SQLite to store only abstracted and anonymized aggregates (e.g., monthly totals, anomaly scores, dataset hashes)
- Replace Streamlit with React + Tailwind or React Native for a smoother, more customizable, and mobile-friendly interface.
- Replace Flask with FastAPI to improve performance, enable async operations, and simplify documentation.
- Containerize the application with Docker for consistent deployment across environments.
- Increase test coverage across the front end and back end with all types of automated testing.
- Set up a GitHub CI/CD pipeline to automatically run tests and deploy after each commit.

- Deploy the backend on cloud providers such as GCP or AWS when scalability is needed, while still supporting local, privacy-preserving execution.