Smart Financial Coach

Design Documentation
PANW Hackathon

Shivprasad Ravikumar Rao Allur

---

1. Project Overview

Smart Financial Coach is an AI-driven personal finance platform designed to address the lack of financial visibility among young adults and freelancers. It transforms opaque transaction data into actionable, personalized insights by combining Generative AI with statistical modeling.

The platform delivers three core capabilities: zero-configuration visualization of bank CSV exports, persona-driven financial coaching powered by an LLM, and automated detection of unwanted subscriptions and spending forecasts — all without requiring manual user input.

2. Design Choices

2.1 Architecture

The application uses a Monolithic Layered Architecture, split into three logical tiers: a React-based frontend, a Node.js/Express backend, and a local SQLite database. This was chosen over microservices to keep the project deployable and maintainable within a hackathon timeline, while still enforcing clear separation of concerns internally. The backend follows a Controller-Service-Repository pattern — controllers handle HTTP routing, services contain business logic, and the repository layer manages SQL queries.

2.2 LLM Integration

Google Gemini is used as an analytical engine rather than a simple chatbot. The key design decision was to pre-compute all hard financial statistics deterministically on the backend before passing them to the model — this prevents the LLM from hallucinating financial figures. All relevant context is sent in a single prompt (one-shot), and the model returns a structured JSON response containing both an array of specific insights and a coaching narrative in a defined persona. Responses are cached in memory for 10 minutes, keyed by a hash of the user's transaction data, to reduce API calls and latency.

2.3 ML & Detection Algorithms

A hybrid intelligence approach combines classical statistical methods with heuristic rules:

- **Spending Forecasts:** OLS linear regression on 6 months of historical monthly totals produces a next-month prediction, a trend direction (Δ), and an R^2 confidence score.
- **Anomaly Detection:** Z-score analysis on category-level monthly spend flags any transaction exceeding $\mu + 2\sigma$, surfacing unusual charges automatically.
- **Subscription Detection:** Inter-transaction arrival times are analyzed per payee. Payments with a Coefficient of Variation (CV) below 0.25 are identified as recurring subscriptions — these are the "gray charges" the platform is designed to catch.
- **Burn Rate Tracking:** Daily average spend is projected across the full calendar month (daily_spend × days_in_month) to give a real-time end-of-month forecast.

2.4 Resilience & Error Handling

A Circuit Breaker pattern wraps the Gemini API integration. If the external service returns errors or hits rate limits, the system automatically falls back to a local mock strategy so the rest of the dashboard stays functional. On the backend, a global exception-handling middleware catches uncaught errors, logs them for debugging, and returns sanitized JSON responses to the client. The frontend is designed to degrade gracefully — statistical data renders immediately while AI insights load asynchronously in the background.

3. Technical Stack

Domain	Technology	Purpose
Languages	TypeScript	End-to-end type safety with shared type definitions across frontend and backend.
	SQL	Standardized data manipulation for complex reporting queries.
Frontend	React 19 + Vite	Component-based UI with a fast build and dev server.
	Tailwind CSS + shadcn/ui	Utility-first styling with pre-built accessible components.
	Recharts	Declarative library for rendering spending and trend visualizations.
Backend	Node.js + Express	Lightweight server for API routes, business logic, and AI orchestration.
	SQLite (Better-SQLite3)	Local, file-based relational database — no external server required.
AI Model	Google Gemini (flash)	Generative AI for personalized coaching and financial insight generation.
Validation	Zod	Runtime schema validation on all API request bodies.
Security	bcrypt + JWT	Password hashing (10 salt rounds) and stateless token-based authentication.

3.1 Security Measures

- Helmet middleware enforces secure HTTP headers including HSTS, X-Content-Type-Options, and XSS protection.
- Rate limiting via express-rate-limit protects all API endpoints from abuse and DDoS.
- All credentials and API keys are stored in .env files and never committed to version control.
- Zod schemas validate every incoming request at the API boundary, blocking injection attacks before they reach application logic.

3.2 Data Handling

CSV imports from bank exports go through a dedicated parsing service that includes relative-date inference logic. This handles inconsistent or ambiguous date formats and non-chronological uploads, normalizing everything before it enters the database.

4. Future Enhancements

Phase	Description
Phase 1	Advanced Analytics — Extract ML logic into a standalone Python/FastAPI microservice. Replace Ordinary Least Squares regression with LSTM/RNN models via PyTorch for significantly better time-series forecasting.
Phase 2	Natural Language Queries — Deploy a Retrieval-Augmented Generation (RAG) pipeline with LangChain to support Text-to-SQL. Users will be able to ask questions like "Show me my Uber spending last month" instead of navigating dashboards manually.
Phase 3	PDF Statement Ingestion — Integrate Gemini Vision for OCR on raw PDF bank statements. This eliminates the need for manual CSV exports entirely, enabling fully automated ETL from uploaded documents.
