Financial Goal Forecasting Advisor

Abstract

This project implements a Goal Forecasting Advisor that helps a user determine whether they are on track to achieve a savings goal by a deadline based on their historical transaction activity. To keep the prototype simple and privacy-preserving for a hackathon environment, the system ingests transactions via CSV upload instead of integrating a live bank API. The backend performs deterministic forecasting and feasibility checks, and generates concrete, non-judgmental spending adjustment suggestions when the goal is not achievable under current spending patterns. An optional LLM layer is used only at the final step to improve the *wording and personalization* of suggestions, while all numerical calculations and feasibility decisions remain deterministic and explainable.

1. Introduction

Many budgeting tools show a category breakdown but don't answer the user's real question:

"Given my spending habits, will I reach my goal by my deadline, and what should I change if not?"

Our Goal Forecasting Advisor addresses this by:

1. Converting historical transactions into a monthly savings capacity estimate.
2. Computing a goal feasibility decision ("On track" vs "Off track").
3. If off track, producing a short list of actionable, realistic adjustments, with estimated monthly impact.

2. System Design and Implementation

2.1 Why CSV Upload Instead of Bank APIs

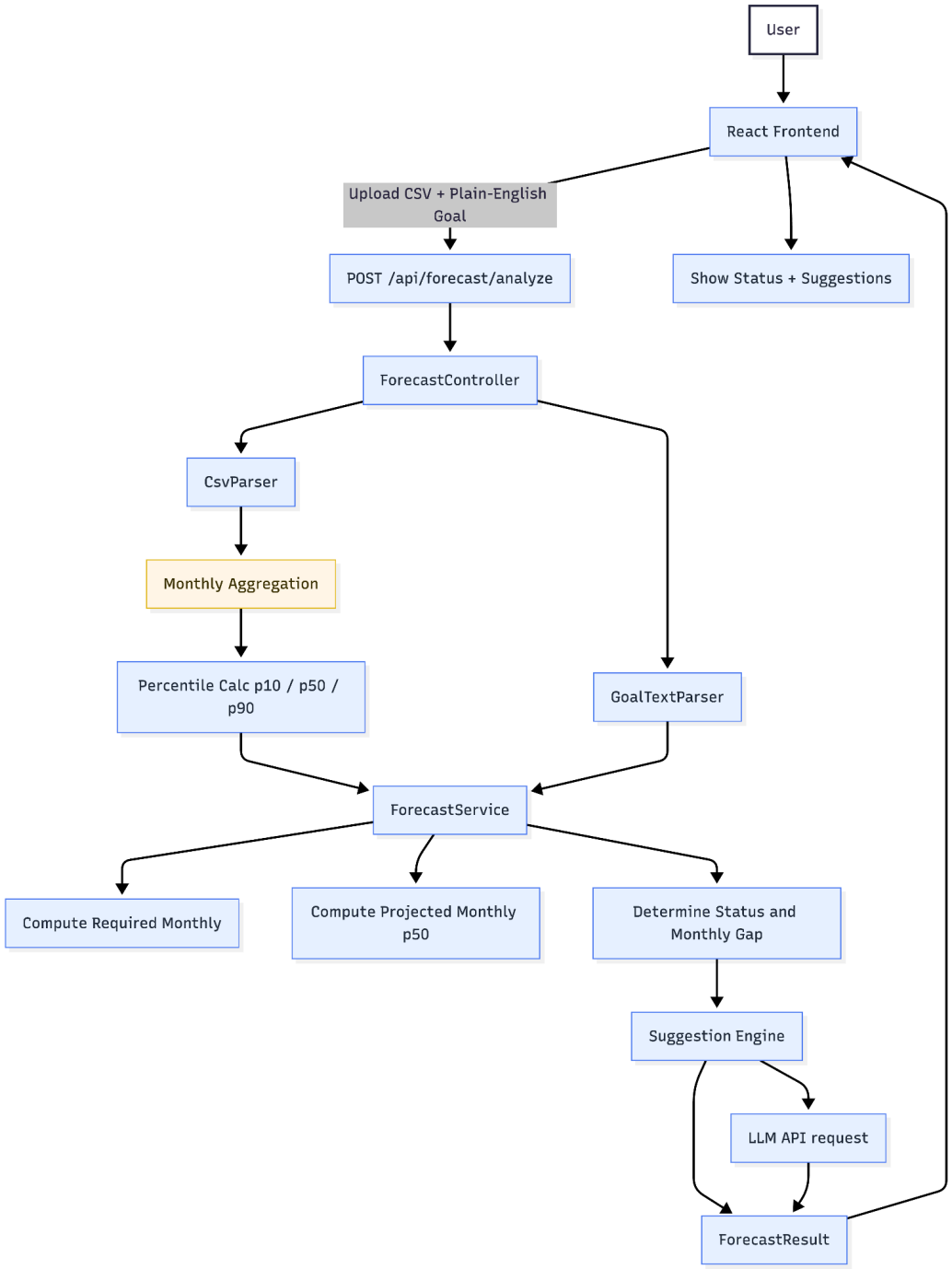
For the hackathon prototype, I chose CSV ingestion over bank APIs for three reasons:

- Security + privacy: No handling of bank credentials, OAuth tokens, or third-party aggregator access scopes.
- Reduced attack surface: Fewer integrations means fewer endpoints and fewer secrets to store.
- Hackathon realism: CSV still represents real transaction logs and supports a complete end-to-end flow (ingest → forecast → suggestions) without external dependencies.

How it works:

- The user uploads a transaction's CSV file from the UI.
- The backend parses it, normalizes records, groups them monthly, and calculates savings capacity.

2.2 End-to-End Flow



1. The user uploads a transaction's CSV.
2. Users type a goal in plain English (example: “Save $5000 in 6 months for a trip”).

3. Frontend submits both inputs to the backend.
4. Backend:
    - Parses the goal text into structured fields (amount + timeframe).
    - Computes monthly savings capacity from transaction history.
    - Determines “On track” or “Off track”.
    - If off track, generate a ranked list of suggestions.
5. Frontend displays:
    - A single status line
    - If off track → 3 suggestions + button to load more

3. Feasibility Logic: How I Decide If the Goal is Achievable

This is the core logic, and it is fully deterministic.

3.1 Monthly Savings Capacity

I estimate how much the user can realistically save per month based on history.
For each month:

- Income = sum of positive transaction amounts
- Spending = sum of absolute values of negative transaction amounts
- Savings capacity (per month) = income - spending

3.2 Capacity Distribution

To avoid overfitting to a single month, I compute a small distribution using the most recent months (example: last 3 months):

- p10 (conservative)
- p50 (typical/median)
- p90 (optimistic)

3.3 Required Monthly Savings

From the goal parser:

- targetAmount extracted from goal text
- monthsToDeadline extracted from goal text

Then:

- requiredMonthly = targetAmount / monthsToDeadline

3.4 Status Decision Rule

I use the median capacity (p50) for the main decision:

- On track if p50 ≥ requiredMonthly
- Off track otherwise

I also compute:

- monthlyGap = max(requiredMonthly − p50, 0)
  This powers the “You’re off-track by ~$X/month” message and drives suggestions.

This logic is simple, explainable, and easy to defend during judging.

4. Suggestions Engine: Deterministic First, LLM Last

4.1 Deterministic Suggestion Generation

When monthlyGap > 0, the backend generates specific behavioral suggestions based on spending patterns.
Instead of saying “Cap Dining at $200”, it suggests actions like:

- “Swap one takeout meal per week for home-cooked”
- “Pause one low-use subscription for a month”
- “Bundle errands to reduce one rideshare trip/week”

Each suggestion includes:

- Title
- Action
- Estimated monthly impact (rough but grounded in historical spend)
- Why it fits (based on categories/merchants seen)

4.2 Why LLM Is Only Used at the End

I intentionally restrict the LLM to language generation only, not calculations:

- Forecast math and feasibility must be correct + deterministic
- LLM output can drift or hallucinate numbers
- So I pass the LLM only structured suggestion objects and ask it to:
  - rewrite them in a supportive tone
  - keep the computed $ impact exactly as provided
  - avoid judgment, avoid cutting essentials, avoid generic advice

Guardrail principle:

> “LLM can improve presentation, but cannot change the numbers.”

4.3 Security Benefit of This Design

Using LLM only at the final step reduces risk because:

- I can send aggregated summaries instead of raw transactions
- I avoid sending sensitive merchant-level data unless required
- If LLM fails, I still return deterministic suggestions (safe fallback)

5. Security and Privacy Design

5.1 Key Security Choices

- CSV-based ingestion: avoids bank credential and token storage for the demo.
- Minimal data handling: analysis can be performed in-memory for the request; no persistence required for the prototype.
- Environment-based secrets: LLM endpoint/key (if used) is configured via environment variables, not hardcoded.
- Least privilege: frontend only calls one backend endpoint; backend exposes minimal API surface.

5.2 Responsible AI Considerations

- Suggestions are framed as options, not commands.
- No investment/financial advising; only spending adjustments inferred from data.
- Avoid recommending cuts to essentials unless explicitly allowed by the user.
- LLM: Send only aggregated summaries and not raw transaction rows

6. Future Enhancements

- Use the Banking API to fetch transactions instead of requesting a CSV file with transactions, format the data received from the API response in required form.
- Multi-goal planning: prioritize goals (emergency fund vs travel, etc.).
- Drift detection: if spending changes significantly, recalibrate capacity.

7. Conclusion

This system demonstrates an end-to-end approach to goal forecasting using transaction history while keeping the prototype secure. It uses CSV ingestion instead of bank APIs, the solution avoids sensitive credential handling and keeps data exposure minimal. Goal achievability is determined using deterministic capacity and required monthly savings logic, ensuring correctness. LLM usage is intentionally constrained to the final “wording” step to improve suggestion quality without compromising numerical integrity or system trust.