SpendSense

An ML/AI-Powered Financial Planning Application

Table of Contents

Introduction

Project Overview and Purpose

SpendSense is an intelligent financial companion designed to go beyond simple expense tracking. While traditional apps tell users where their money *went*, SpendSense uses **Time-Series Forecasting (Facebook Prophet)** to predict where their money *is going* and **Generative AI (Google Gemini)** to act as a personalized financial coach.

The purpose is to empower users with **proactive financial foresight** rather than reactive regret.

Goals

- **Secure Aggregation**: Seamlessly aggregate accounts (Checking, Savings, Credit) via Plaid
- **Smart Visualization**: Provide clear, immediate visuals of spending habits (Pie Charts) and balance trends (Composed Line/Scatter Charts)

- **Predictive Analysis**: Use Facebook Prophet to forecast future net worth based on reconstructed historical data, accounting for seasonality
- **Actionable AI Coaching**: Deliver personalized, casual, and human-like advice using Google Gemini to help users stay on track
- **Gray Charge Detection**: Use AI to analyze transaction history for recurring subscriptions and "vampire costs" while preserving privacy

---

Target Audience

- **Young professionals and students** who want to save for specific goals (e.g., "Save $3,000 for a trip")
- **Users who find spreadsheets too manual** and traditional budgeting apps too passive

---

Requirements

Functional Requirements

Requirement	Description
Authentication	Users must be able to sign up and log in securely (JWT/Bcrypt)
Bank Linking	Users must be able to link bank accounts via Plaid Link (Sandbox mode)
Dashboard	System must display current net worth, recent transactions (scrollable/searchable), and spending breakdown
Goal Forecasting	Users can input a target amount and date; the system must calculate if they are "On Track" using ML forecasting
AI Insights	The system must generate unique, text-based financial insights for every forecast request
Spending Analysis	Users can trigger an AI scan to find subscriptions and gray charges in their last 60 days of history

Non-Functional Requirements

Requirement	Description
Security	Plaid Access Tokens are stored encrypted (in prod); sensitive bank credentials never touch the server
Usability	UI must be clean and responsive

Architecture

High-Level System Architecture

Frontend (React) ↔ Backend (FastAPI) ↔ PostgreSQL Database

Backend integrates with:

- **Plaid API** (Banking data)
- **Prophet ML** (Forecasting)
- **Gemini API** (AI insights)

Technology Stack

Frontend

- **Framework**: React (Vite)
- **Styling**: Tailwind CSS
- **Visualization**: Recharts (Composed Charts for Trends, Pie Charts for Breakdown)
- **Icons**: Lucide React
- **Routing**: React Router DOM

Backend

- **Framework**: FastAPI (Python)
- **ORM**: SQLAlchemy
- **ML Library**: Facebook Prophet (Time-Series) & Pandas
- **AI**: Google Gemini 2.5 Flash (Generative Insights)

Database

- **DB**: PostgreSQL

External APIs

- **Plaid API**: For fetching transactions and real-time balance data
- **Google Gemini API**: For generating natural language financial insights

---

User Experience Design

User Personas and User Flows

Primary User Flow

1. **Onboarding**: Landing Page → Sign Up → Login
2. **Setup**: "Connect Bank" (Plaid Link) → Success
3. **Monitoring**: Dashboard loads → User sees Balance Cards, Spending Breakdown (Pie Chart), and Scrollable Transaction List
4. **Analysis**: User clicks "Analyze My Spending" → AI scans for gray charges and recurring costs
5. **Forecasting**: User enters Goal ($ Amount & Date) → System runs Prophet model → Charts Trend Line + AI Advice appears

Design Principles

- **Trust**: Professional color palette (Slate, Blue, Emerald)
- **Clarity**: Use charts to explain complex data (Trends over time)
- **Feedback**: Loading states (spinners) and clear error messages are mandatory

---

Database Design

Data Model

Users Table

Column	Type	Constraints	Description
`id`	Integer	PRIMARY KEY	Unique user identifier
`first_name`	String		User's first name
`last_name`	String		User's last name
`email`	String	UNIQUE	User's email address
`hashed_password`	String		Bcrypt-hashed password
`plaid_access_token`	String	NULLABLE	Plaid access token (Bank Key)
`bank_connected`	Boolean	DEFAULT FALSE	Connection status

Data Storage Considerations

- Transactions are not stored permanently in the database to ensure data freshness and privacy

- They are fetched live from Plaid and processed in-memory

Technical Design

Module Breakdown

1. Auth Module (`routes/auth.py`)

Handles JWT token generation, password hashing (bcrypt), and user registration/login.

2. Plaid Service (`routes/plaid_api.py`)

- Exchanges public tokens for access tokens
- Fetches recent transactions (up to 180 days)
- Calculates real-time Net Worth (Assets - Liabilities)

3. ML Engine (`ml_utils.py`)

- **Data Reconstruction**: Reconstructs historical daily balances by "undoing" transactions from the current balance
- **Forecasting**: Uses **Facebook Prophet** to fit a time-series model to historical data, accounting for seasonality
- **Prediction**: Extrapolates the trend line to the target date to predict future net worth

4. Goal & AI Service (`routes/goals.py`)

- **Forecast Endpoint**: Orchestrates the flow: Fetch Data → Run Prophet → Feed Stats to Gemini → Return JSON
- **Analyze Endpoint**: Fetches last 60 days of data → Anonymizes descriptions → Asks Gemini to find "Gray Charges"

Component Interaction: The "Forecast" Flow

```
1. Frontend sends POST /api/goals/forecast
   {target_amount: 3000, target_date: "2025-06-01"}

2. Backend queries Database for plaid_access_token

3. Backend calls Plaid API
   → Gets current balance and 180 days of transactions
```

```
4. Backend calls forecast_balance() (ML Utility)
   a. Reconstructs historical daily balances
   b. Prepares data for Prophet (ds, y)
   c. Trains Prophet model
   d. Predicts future balance

5. Backend calls Gemini API
   Prompt: "User has $X, predicted to have $Y.
           Goal is $Z. Give casual advice."

6. Backend returns combined JSON
   {history, prediction, insight, is_on_track}
   → Frontend
```

Component Interaction: The "AI Insights" Flow

```
1. Frontend requests POST /api/goals/analyze_spending
   Payload: {user_id: 1}

2. Backend queries Database for plaid_access_token

3. Backend calls Plaid API
   → Gets 60 days of transaction data

4. Backend extracts transaction metadata
   → Only: {description, amount, category}
   → Anonymization: Excludes account numbers and IDs

5. Backend calls Gemini API (Flash 2.5) for analysis
   Prompt: "Analyze these transactions for:
   - Forgotten subscriptions (Gray Charges)
   - Spending habits/patterns
   - One actionable saving tip"

6. Backend returns AI Analysis
   {
     "analysis": "1. [Subscription 📺] Netflix
                 ($15.99)...\n2. [Habit ☕]..."
   }
   → Frontend displays the formatted text card
```

API Endpoints

Base URL: `http://localhost:8000`

Authentication Endpoints

- `POST /api/auth/signup` - Register new user
- `POST /api/auth/login` - Authenticate user and return JWT token

Plaid Integration Endpoints

- `POST /api/plaid/create_link_token` - Generate Plaid Link token
- `POST /api/plaid/exchange_public_token` - Exchange public token for access token
- `POST /api/plaid/transactions` - Fetch balances and transaction history

Goal Forecasting Endpoints

- `POST /api/goals/forecast` - Run Prophet forecasting and generate AI advice

AI Insights Endpoints

- `POST /api/goals/analyze_spending` - Detect gray charges and provide spending recommendations

Security Considerations

Current Implementation

- **Password Security**: Bcrypt hashing with salt rounds for all passwords
- **JWT Authentication**: Tokens expire after configurable duration, stored securely in httpOnly cookies
- **Privacy-First AI**: Only transaction descriptions and amounts sent to Gemini API—no account numbers, routing numbers, SSNs, or personal identifiers
- **Input Validation**: Server-side validation on all endpoints to prevent injection attacks
- **CORS Configuration**: Restricted to specific allowed origins in production

Security Best Practices Applied

- **No Raw Credentials**: Plaid access tokens stored encrypted; banking credentials never touch our servers
- **Principle of Least Privilege**: API only requests necessary Plaid scopes (transactions, auth)
- **SQL Injection Prevention**: SQLAlchemy ORM with parameterized queries

Future Enhancements

Architecture & Performance

Asynchronous Task Queue:

- **Issue**: The analyze_spending endpoint blocks the main thread while waiting for Google Gemini (Generative AI), risking HTTP timeouts
- **Solution**: Offload AI and ML processing to a background worker queue (e.g., Celery with Redis). The API would return a "Job ID" immediately, and the frontend would poll (or use WebSockets) for the result

Model Caching Strategy:

- **Solution**: Since the Prophet forecasting model is computationally expensive, implement a Write-Through Cache in Redis. Store the forecast JSON with a 24-hour TTL, invalidating the cache only when new transaction webhooks are processed

Security & Data Integrity

Token Encryption:

- **Solution**: Instead of storing plaid_access_token as plain text, implement application-level encryption using AES-256-GCM before writing to the database. This ensures that even a database dump would not compromise user banking credentials

Advanced Authentication:

- Two-Factor Authentication (2FA) using TOTP or SMS
- OAuth2 social login (Google, Apple)

Feature Expansion

- Custom budget categories and limits
- Bill payment predictions and reminders

Development Setup

Backend Setup

```
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
uvicorn main:app --reload
```

Frontend Setup

```
cd frontend
npm install
npm run dev
```

Environment Variables

```
# .env
DATABASE_URL=postgresql://user:pass@localhost/spendsense
PLAID_CLIENT_ID=your_client_id
PLAID_SECRET=your_secret
PLAID_ENV=sandbox
GEMINI_API_KEY=your_gemini_key
JWT_SECRET_KEY=your_jwt_secret
```