Product Requirements Document

Overview

This document outlines a **web-based AI-driven financial tracking tool** designed to help users track expenses, set budgets, and receive personalized guidance on either saving more effectively or paying off debt faster. It clarifies team responsibilities, resolves conflicting requirements, and keeps the scope focused on delivering clear, maintainable features.

Goals and Objectives

Goal

Empower users to make better financial decisions by combining **real-time account tracking** (or manual data entry) with **AI-driven insights**, enabling them to either **boost savings** or **reduce debt** in a systematic, personalized way.

Objectives

- **Consolidate Financial Data**: Allow users to either manually input their income and expenses or optionally link their bank/credit accounts (if desired in future versions), so they have a **centralized view** of their finances.

- **AI-Driven Recommendations**: Leverage historical spending, budgeting patterns, and user goals to **suggest next steps**, such as saving allocations or debt payments.

- **Actionable Alerts & Notifications**: Provide **timely, relevant nudges** to keep users on track with their budgeting and financial goals.

- **Flexible Goal Management**: Support **both saving-oriented and debt-oriented** objectives, adjusting forecasts and recommended actions accordingly.

Responsibility and Collaboration

- **Design Team**: Create the UX/UI flows for user onboarding, goal setup (saving or debt), and real-time dashboards displaying recommendations.

- **Frontend Development Team**: Implement React (or chosen framework) components for dashboards, forms (goal setup, budget tracking), and real-time charts.

- **Backend/ML Development Team**: Build secure APIs for any potential account integrations (optional), develop or integrate **AI/ML models** (e.g., expense categorization, goal forecasting), and manage user data.

- **QA Team**: Test all user stories with a focus on **functionality** (recommendations make sense, data is accurate) and **user experience** (system is intuitive, alerts work as expected).

- **Product Management**: Define success metrics, scope, and acceptance criteria for AI recommendations. Oversee timeline and feature prioritization.

Scope

In Scope

1. **Manual Income and Expense Entry**, with the option (in future phases) for secure account integration if users prefer.

2. **Automated Expense Categorization** using machine learning models to label transactions (e.g., groceries, bills, dining out).

3. **Budget Setup & Goal Tracking** for both saving and debt repayment.

4. **AI-Driven Forecasting** that predicts future cash flow, suggests debt payment strategies, and recommends saving/investment adjustments.

5. **Alerts & Notifications** that adapt to user behavior (e.g., overspending, missed debt targets).

6. **Basic Investment Guidance** (e.g., high-yield savings accounts, simple ETF suggestions) with disclaimers.

Out of Scope

1. **Advanced Multi-Currency Support** (focusing on a single currency for MVP).

2. **Complex Investment Portfolio Management** (beyond basic suggestions).

3. **Full Credit Score Monitoring/Reporting** (only basic debt prioritization).
4. **Comprehensive Tax Calculation or Filing Features**.

Key Features

1. **Manual Income & Expense Entry** (or optional secure account integration in future).
2. **Automated Expense Tracking & Categorization**.
3. **Budgeting & Goal Management** (Saving or Debt).
4. **AI-Driven Forecasting & Recommendations**.
5. **Alerts & Notifications**.
6. **Basic Investment Insights** (For Savings Mode).
7. **Dynamic UI & Visual Insights** (Real-time dashboards and “what-if” scenarios).

Architecture and Tech Stack

1. **Frontend**
    - React (or a comparable modern framework) for UI/UX, state management, and chart visualizations (Chart.js, Recharts, etc.).
2. **Backend**
    - Node.js (Express) or Python (Flask/FastAPI) for handling REST APIs, user auth, and optional communication with financial institution APIs.
    - **AI/ML Components** implemented in Python or Node-based ML libraries for transaction categorization and forecasting.
3. **Database**
    - Relational (PostgreSQL, MySQL) or NoSQL (MongoDB) for user data, transaction storage, and model insights.

4. **Data Aggregation & Security**
    - (Future/optional) Use an aggregator (e.g., Plaid) or direct institution APIs for secure, real-time data if the user opts to connect.
    - Must use HTTPS and store all credentials/tokens securely (encrypted at rest).
5. **Hosting & Deployment**
    - **Replit** for collaborative development and quick iteration.
    - Potential migration to AWS, GCP, or Azure for production scaling.

Development Approach

1. **Project Setup**
    - Create a single-page React application (if using React) with a well-defined folder structure.
    - Initialize the backend environment (Node.js or Python) in Replit.
    - Configure environment variables for aggregator API keys and DB credentials (if implementing optional institution integration).
2. **Core Features Implementation**
    - **Manual Income/Expense Entry** (User Story 1, newly updated).
    - **Expense Categorization Model**: Implement or integrate an ML model; maintain a feedback loop for user recategorization.
    - **Budget & Goal UI**: Let users create monthly budgets and set saving/debt goals.
    - **Forecasting Logic**: Build or integrate a service that analyzes spending patterns to generate dynamic recommendations.
3. **Real-time Visualization**
    - Implement charts to show spending breakdown, progress toward goals, and future projections.

    - Ensure data refreshes automatically upon new entries or user input changes.

4. **Alerts & Notifications**
    - Provide user-configurable thresholds (e.g., spend limit in a specific category).
    - Send email or in-app notifications triggered by system conditions (approaching overspending, debt repayment progress, etc.).

5. **Testing & Validation**
    - **Unit Tests**: For categorization logic, budget calculations, forecasting modules.
    - **Integration Tests**: Verify correct communication between front-end, back-end, and (optional) financial data aggregator.
    - **User Acceptance Testing (UAT)**: Involve real users in evaluating the clarity of AI recommendations and overall user experience.

6. **Phase 2 Enhancements** (Post-MVP)
    - Expand personalization (more nuanced ML-driven nudges, advanced forecasting).
    - Potential integration with additional third-party financial apps (e.g., credit monitoring, automated portfolio rebalancing).

User Stories

US-001: Manual Income/Expense Input

User Story: As a user, I want to manually input my monthly income and expenses (rather than linking my financial institution), so I can track my financial data in one place without providing bank credentials.

Acceptance Criteria:

- User can add or edit entries for “Income” and each “Expense” category.
- The system updates the overall dashboard, budgets, and forecasts based on these manual inputs.

- (Optional) If the user chooses to connect an account in the future, the system can override or merge manual entries with new data sources.

US-002: Automated Expense Categorization (with Specific Category Titles)

User Story: As a user, I want my manually entered expenses automatically categorized into **Housing, Transportation, Food, Healthcare, Insurance & Personal Protection, Debt & Loans, Personal & Family, Entertainment & Leisure, Gifts & Charity, Education & Professional Development,** and **Miscellaneous & Other**, so I don't have to sort each entry individually.

Acceptance Criteria:

- ML categorization runs on each new expense entry.
- By default, each expense is assigned to one of the specified categories:
    1. **Housing**
    2. **Transportation**
    3. **Food**
    4. **Healthcare**
    5. **Insurance & Personal Protection**
    6. **Debt & Loans**
    7. **Personal & Family**
    8. **Entertainment & Leisure**
    9. **Gifts & Charity**
    10. **Education & Professional Development**
    11. **Miscellaneous & Other**
- Users can override any category, which trains or updates the system's future predictions.

US-003: Budget Creation & Monitoring

User Story: As a user, I want to set monthly budget categories (food, bills, entertainment, etc.) and monitor spending so I stay within my limits.

Acceptance Criteria:

- Users can create or edit category budgets.
- Dashboard shows real-time budget consumption (e.g., X% of the monthly dining or entertainment budget used).
- The system alerts the user when spending approaches or exceeds a given category's budget limit.

US-004: Goal Setting (Saving or Debt)

User Story: As a user, I want to specify a financial goal, like building a $5,000 emergency fund or paying off $2,000 credit card debt by a certain date, so I can measure my progress.

Acceptance Criteria:

- Users can add a saving or debt goal with a target date.
- The system shows percentage completion, estimated time to completion, and recommended changes (e.g., "Increase monthly contributions by $50 to meet your deadline").

US-005: AI-Driven Recommendations

User Story: As a user, I want AI-generated suggestions (e.g., reduce weekly dining out, add $50 to monthly debt payments) so I can see faster progress toward my goals.

Acceptance Criteria:

- AI engine analyzes historical transactions and user goals.
- System provides weekly or monthly recommendation summaries (e.g., "Cut spending by $X to reach your savings goal one month sooner").

US-006: Debt Prioritization

User Story: As a user with multiple debts, I want the system to show me which debt to pay first (highest interest vs. snowball method), so I can minimize interest or gain quick wins.

Acceptance Criteria:

- System calculates interest rates, balances, and user preferences (avalanche vs. snowball) to determine the recommended order.

- Alerts indicate progress or encourage additional payments if beneficial.

US-007: Alerts & Notifications

User Story: As a user, I want to receive notifications when I'm about to overspend in a category or if my debt payment is due, so I can avoid penalties and stay on budget.

Acceptance Criteria:

- Users can enable/disable or customize alerts.

- Notifications are triggered when certain thresholds are met (e.g., 80% of budget used, loan payment due in 3 days).

US-008: Real-time Visualization & "What-if" Scenarios

User Story: As a user, I want to run "what-if" scenarios (e.g., reduce monthly dining by 20%), so I can see how it impacts my savings or debt payoff timeline.

Acceptance Criteria:

- User inputs hypothetical changes (like adjusting budget categories).

- The system updates a forecast chart, showing the impact on payoff date or saving timeline.

Changes from Earlier Versions

1. **User Story 1 Updated**: Users now manually input income and expenses rather than linking to a financial institution (though optional linking can be added in the future).

2. **User Story 2 Enhanced**: Specific category titles (Housing, Transportation, Food, Healthcare, Insurance & Personal Protection, Debt & Loans, Personal & Family, Entertainment & Leisure, Gifts & Charity, Education & Professional Development, and Miscellaneous & Other) have been added for more detailed automatic classification.

Final Notes

- **Success Metrics**: Product management will define daily/weekly active user benchmarks, average monthly savings, or average debt reduction to measure impact.

- **Timeline & Milestones**: Collaboration between design, development, and QA will be tracked via a shared board (e.g., Trello, Jira) for sprint planning.

- **Future Expansion**: Potential enhancements include multi-currency support, deeper investment analytics, or advanced credit score tracking. Each new feature will be reviewed for feasibility and maintainability before implementation.