Business Requirements Document

1. Project Overview & Vision

The Personal Finance Tracker (PFT) is a locally-run web application designed to empower individuals to take control of their finances. By providing clear, simple tools for tracking income, expenses, and budgets, and by leveraging machine learning for intelligent insights, the application helps users understand their spending habits, make informed financial decisions, and work towards their financial goals. All user data remains private and stored exclusively on their machine.

2. Business Objectives

- Clarity: Provide a clear and accurate view of the user's financial standing across all their accounts.
- Control: Enable users to create and manage a monthly budget to proactively control their spending.
- Insight: Offer simple, visual reports and AI-driven insights that reveal spending patterns and financial progress over time.
- Privacy & Simplicity: Ensure all user financial data remains private and the application is straightforward to set up and use.

3. Target Audience

The primary user is a tech-savvy individual who wants to manage their personal finances without relying on third-party cloud services. They are comfortable with basic command-line operations to start a local server and prioritize data privacy and control.

4. Scope & Functional Requirements (V1.0)

The scope for Version 1.0 is focused on establishing the core functionalities of manual financial tracking and budgeting with intelligent assistance.

In-Scope Features (MVP)

Feature ID	Feature Name	User Story & Acceptance Criteria	Priority
FN-01	Application Setup & Bootstrapping	User Story: As a new user, I want to easily start the application and have a pre-configured environment with default categories so I can begin using it immediately. Acceptance Criteria: 1) The application is started by running	Must-Have

		two simple commands in a terminal (one for backend, one for frontend). 2) On first run, the database is automatically created and populated with a standard set of expense and income categories. 3) The user can immediately access the dashboard at a local URL (e.g., http://localhost:3000).	
FN-02	Dashboard	User Story: As a user, I want to see a summary of my key financial information on one screen so I can get a quick overview of my financial health. Acceptance Criteria: The dashboard must display: 1) Total balance across all accounts, calculated as the sum of all account balances. 2) A list of accounts with their current balances. 3) A quick view of recent transactions. 4) A visual indicator for categories where spending is close to or over budget. 5) Visual flags for any transactions flagged as	Must-Have

		anomalies.	
FN-03	Account Management	User Story: As a user, I want to add, edit, and delete my financial accounts (e.g., Checking, Savings, Credit Card) so I can track their balances. Acceptance Criteria: 1) User can create an account with a name, type, and initial balance. 2) User can edit the account name. 3) User can only delete an account if it has zero transactions linked to it. A confirmation dialog must be shown before deletion.	Must-Have
FN-04	Transaction Management	User Story: As a user, I want to manually add, edit, and delete transactions (income, expenses, transfers) so I can maintain an accurate record of my cash flow. Acceptance Criteria: 1) The "Add Transaction" form must include Date, Description, Amount, Account, and Category fields. 2) The form must validate inputs (valid date, positive numerical amount). 3) User can edit any field of an existing transaction.	Must-Have

		4) Deleting a transaction requires confirmation and recalculates the account balance. 5) Transfers: The user can record a transfer between two accounts, which creates two linked transactions: an expense from the source account and an income to the destination account.	
FN-05	Category Management	User Story: As a user, I want to create, edit, and delete spending and income categories so I can organize my transactions meaningfully. Acceptance Criteria: 1) The system starts with a default set of categories. 2) Each category is defined as an "Income" or "Expense" type. 3) User cannot delete a category that is currently assigned to any transaction.	Must-Have
FN-06	Monthly Budgeting	User Story: As a user, I want to set a monthly spending limit for each expense category so I can track my spending against my	Must-Have

		budget. Acceptance Criteria: 1) User can set a budget amount for any expense category for a specific month and year. 2) The system displays the budgeted amount, actual spending, and the remaining amount. 3) The dashboard highlights categories where spending exceeds 90% of the budget.	
FN-07	Transaction View & Filtering	User Story: As a user, I want a dedicated page to view and filter all my transactions so I can easily find specific entries. Acceptance Criteria: User can filter transactions by date range, account, and category. The list displays all transaction details and visually highlights any anomalies.	High
FN-08	Basic Reporting	User Story: As a user, I want to see a simple report of my spending by category for a selected month so I can understand where my money is going.	High

		Acceptance Criteria: The report page must display a pie chart of expenses by category for a user-selected month.	
FN-09	Smart Categorization	User Story: As a user, when I enter a transaction, I want the system to suggest a category and I want to be able to correct it, so the system learns and data entry becomes faster over time. Acceptance Criteria: 1) As the user types in the description field, the system suggests a category in a dropdown. 2) The user can accept the suggestion or choose a different one. 3) The user's final selection is stored and given highest priority for future model retraining.	High
FN-10	Anomaly Detection & Feedback	User Story: As a user, I want to be alerted to unusually large transactions and provide feedback on them, so I can catch errors and improve the system's accuracy. Acceptance Criteria: 1) The system flags a transaction as an	Medium

		anomaly if its amount is >3 standard deviations from the category mean. 2) Flagged transactions are visually highlighted in the transaction list and dashboard. 3) The user can "Mark as Normal" on a flagged transaction, which feeds this information back to the model.	
FN-11	Cash Flow Forecast	User Story: As a user, I want to see a projection of my account balance for the next 30 days so I can plan my finances better. Acceptance Criteria: 1) A graph on the dashboard shows the projected daily balance. 2) If there is less than 45 days of historical data, the graph is hidden and a message "Insufficient data for forecasting" is shown.	Medium
FN-12	Budget Recommendation	User Story: As a user, when creating a new monthly budget, I want the system to suggest budget amounts based on my past spending habits. Acceptance Criteria: 1) On the	High

		budget setup page, the system suggests the average spending for each category over the last 3 months. 2) If there is less than 1 month of historical data for a category, the suggestion is left blank.	
FN-13	ML Model Retraining	User Story: As a user, I want to manually trigger the retraining of the ML models with my latest data so that the smart features become more personalized and accurate. Acceptance Criteria: 1) A "Retrain AI Models" button is available in a Settings page. 2) Clicking it shows a loading indicator and a confirmation message upon completion.	Medium

Out-of-Scope Features (Explicitly for Future Releases)

- Automatic bank syncing or data aggregation.
- Multi-user support or user profiles.
- Investment tracking (stocks, mutual funds).
- Loan amortization schedules.
- Multi-currency support.
- Bill payment reminders and alerts.
- Data import/export (e.g., from CSV).
- Automated, scheduled model retraining.

5. Assumptions & Constraints

- Platform: The application will be run locally on the user's computer. The user must manually start the backend (Python/FastAPI) and frontend (React) servers via command line.
- Data Persistence: All data is stored in a single SQLite file (finance.db) in the backend directory. The user is responsible for backing up this file.
- Security: The backend API will only be accessible from localhost to prevent external network access.
- Currency: The application will only support a single currency (e.g., INR). No currency conversion is provided.
- Initial ML Models: The application will be shipped with pre-trained, general-purpose ML models for categorization and anomaly detection.

6. Non-Functional Requirements
   - Usability: The application must be intuitive. Key actions (adding a transaction, checking a budget) should be achievable in 3 clicks or less from the dashboard.
   - Performance: The UI should feel responsive. API responses for standard requests (e.g., loading transactions) should be under 500ms.
   - Error Handling: The application must not crash on user input errors. Friendly, actionable error messages must be displayed for invalid inputs (e.g., "Please enter a valid date") and backend communication failures (e.g., "Cannot connect to server. Please ensure the backend is running.").
   - Data Integrity: The system must prevent actions that would corrupt data, such as deleting a category in use, with clear error messages explaining the constraint.