Product Requirements Document (PRD)

Project: Personal Finance Tracker API (MVP)

Key Stakeholder	Contact	Date	Version	Status
Backend Developer(Michael Adeniran)	07030834157	Oct 2025	1.0	MVP Complete

1. Goal

To develop a secure, high-performance API that allows individual users to track personal income and expenses in near real-time. The API must demonstrate proficiency in modern backend development practices, including database interaction, secure authentication, and high-speed caching for analytics.

2. Success Metrics

- **100% API Uptime** during testing.
- **Authentication Success Rate:** 100% of valid login/registration attempts succeed.
- **Analytics Latency:** User balance calculation (via the /analytics/ endpoint) must return in under 50ms (achieved via Redis caching).
- **Code Quality:** Use of Python/Django best practices, clear separation of concerns (Views, Serializers, Models).

3. Scope & Features

3.1. Core Features (Must Have)

Feature ID	Feature Name	Description
FEAT-01	User Registration & Login	Allow new users to register (username, email, password) and existing users to log in, receiving a JWT pair (access/refresh).
FEAT-02	Transaction Creation (POST)	Authenticated users can log new transactions, specifying amount, type (income or expense), description, and category.
FEAT-03	Transaction History (GET)	Authenticated users can retrieve a list of all their

		historical transactions.
FEAT-04	Real-Time Analytics (Balance)	Calculate and return the authenticated user's current total balance (Income - Expenses).
FEAT-05	Go Concurrency Demo	Provide a separate, runnable Go program demonstrating high-volume, concurrent transaction processing, showcasing performance and scalability awareness.

3.2. Future Enhancements (Nice to Have)

- **FEAT-06:** Transaction Editing/Deletion (PUT/DELETE).
- **FEAT-07:** Date-Range Filtering on Transaction History.
- **FEAT-08:** Detailed Analytics (e.g., breakdown of spending by category, monthly trends).
- **FEAT-09:** Password Reset functionality.

4. Technical Requirements

4.1. Technology Stack

- **Backend Framework:** Python 3.11+, Django 5.x, Django REST Framework (DRF).
- **Database:** SQLite (Development), PostgreSQL (Production/Heroku).
- **Authentication:** djangorestframework-simplejwt.
- **Caching/Message Broker:** Redis.
- **Concurrency Demo:** Go Language.

4.2. Architecture and Data Model

- **User Model:** Custom Django model linked to the transactions app (AUTH_USER_MODEL = 'transactions.User'). Includes username, email, and standard fields.
- **Transaction Model:** Linked via ForeignKey to the User. Fields include user, amount (Decimal), type (Choice: 'income', 'expense'), description, category.
- **Caching Strategy (FEAT-04):** The user's running balance must be calculated and stored in **Redis** with a short Time-To-Live (TTL) (e.g., 5 minutes) to avoid repeated database lookups for analytics. The cache must be invalidated upon successful creation of a new transaction.