SOFTWARE REQUIREMENTS SPECIFICATION

Group 4 – Personal Finance

Anthony Pongallo, Nicholas Molchak, Jacob Gerbracht

1.0 Introduction

This document describes all **data, functional, and behavioral requirements** for the Personal Finance and Budgeting Application. The application enables users to track their daily expenses, set budgets, and gain insights into their financial habits. It details user roles, use-cases, data models, and the interfaces required for operation.

1.1 Goals and Objectives

- **Goal**: Provide a **secure, user-friendly** platform for individuals to record expenses, set budget limits, and view financial summaries.
- **Objectives**:
    1. Enable **user authentication** and **secure data** storage.
    2. Allow users to **create, read, update,** and **delete** (CRUD) expense records.
    3. Offer a **budgeting feature** to track spending limits per category.
    4. Generate **summaries** (e.g., monthly totals, category breakdown) to help users identify spending patterns.

1.2 Statement of Scope

The software is a **web application** that focuses on **expense logging** and **budget management** for individual users. Major inputs include **expense details** (amount, date, category, notes) and **budget constraints** (category and limit). Processing involves **storing**, **validating**, and **aggregating** these records. Outputs include **charts, lists, and alerts** that show users their spending patterns and how close they are to their budget limits.

1.3 Software Context

- The system is part of the **personal finance** domain, targeting individuals or students who want to **improve financial awareness**.
- The application can be deployed as a **standalone web service** with a front-end (React, Vue, or Angular) communicating with a back-end (Node.js/TypeScript or Python) and a NoSQL or SQL database (e.g., MongoDB, PostgreSQL).
- **Integration Points**: Optionally, the system could integrate with external banking APIs or import CSV statements to retrieve transaction data automatically.

1.4 Major Constraints

- **Time Constraints**: Must be completed by the end of the semester.
- **Team Size**: Limited developers, so features must be prioritized carefully.
- **Security and Compliance**: Must handle user authentication and sensitive financial data responsibly (password hashing, secure token usage).
- **Budget**: Reliant on free-tier hosting and database services, limiting large-scale performance testing.

2.0 Usage Scenario

2.1 User Profiles

1. **Registered User**
   - Typical individual managing personal finances.
   - Can add/edit/delete expenses, view budgets, and set spending limits.
   - No administrative privileges over other users' data.
2. **Administrator (Optional)**
   - Monitors system usage, performance, or suspicious activities.
   - May help reset user passwords or manage platform-wide settings (e.g., category defaults).

2.2 Use-Cases

1. **UC-01: User Registration**
   - **Goal**: Create a new account with a secure password.
   - **Primary Actor**: Unregistered user
   - **Precondition**: User has not registered an account.
   - **Main Success Scenario**: User provides name, email, password → system saves encrypted password → system returns success.
2. **UC-02: User Login**
   - **Goal**: Authenticate and access personal finance data.
   - **Primary Actor**: Registered user
   - **Precondition**: Valid user account exists.
   - **Main Success Scenario**: User provides correct credentials → system returns JWT or session → user proceeds to dashboard
   -
3. **UC-03: Record Expense**
   - **Goal**: Add a new expense with amount, category, date, and optional note.
   - **Primary Actor**: Registered user
   - **Main Success Scenario**: User inputs expense data → system validates and saves record → updated list of expenses is displayed.
4. **UC-04: Create/Update Budget**
   - **Goal**: Define or modify budget limit per category.
   - **Primary Actor**: Registered user
   - **Main Success Scenario**: User specifies category and limit → system records or updates the budget → user sees updated budget data.
5. **UC-05: View Financial Summaries**
   - **Goal**: See detailed expense breakdown and budget usage.
   - **Primary Actor**: Registered user
   - **Main Success Scenario**: User navigates to summary page → system retrieves aggregated data → user sees charts/lists of spending trends.

2.3 Special Usage Considerations

- **Offline Access**: If needed, a progressive web app approach may allow limited offline functionality (though out of scope for the basic version).
- **Integration with Bank APIs**: Future enhancement; not part of core release but may require advanced security and data mapping.

3.0 Data Model and Description

3.1 Data Description

3.1.1 Data Objects

- **User**: Represents an individual with fields such as `name`, `email`, `password`, and `userId`.
- **Expense**: Stores transactional data with `amount`, `category`, `date`, `note`, and references `userId`.
- **Budget**: Holds the spending limit with `category`, `limit`, and references `userId`.

3.1.2 Relationships

- One **User** → Many **Expenses**.
- One **User** → Many **Budgets**.
- Each **Budget** is linked to a single **category**. A **User** can have multiple budgets for different categories.

3.1.3 Complete Data Model

A simplified Entity-Relationship Diagram (ERD) might look like:

```
+--------+              +----------+             +----------+
|  User  |  ---------*  | Expense  |             |  Budget  |
+--------+              +----------+             +----------+
|userId  |--------------|userId    |             |userId    |
|name    |       \--    |amount    |             |category  |
|email   |            \ |category  |             |limit     |
|password|              |date      |             |budgetId  |
+--------+              |note      |             +----------+
                        +----------+
```

3.1.4 Data Dictionary

- **userId**: Unique identifier (e.g., string/UUID).
- **name**: String representing the user’s full name.
- **email**: String storing user’s email; must be unique.
- **password**: Hashed password string.
- **expenseId**: Unique identifier for an expense.
- **amount**: Numeric value of the expense.
- **category**: String describing expense or budget category (e.g., “Groceries,” “Rent,” “Entertainment”).
- **date**: Date object to track when the expense occurred.
- **note**: Optional string for additional details.
- **limit**: Numeric budget threshold for a given category.
- **budgetId**: Unique identifier for each budget record.

4.0 Functional Model and Description

4.1 Description of Major Functions

4.1.1 Requirement 1 – Registration & Login

- **Description**: The system must allow new users to create accounts, and existing users to authenticate.
- **Inputs**: User credentials (name, email, password).
- **Process**: Password hashing, user validation, token generation.
- **Output**: Confirmation of success, JWT for authenticated sessions.

4.1.2 Requirement 2 – Expense Management

- **Description**: Users can add, view, update, and delete expenses.
- **Inputs**: Expense attributes (amount, category, date, note).
- **Process**: Validate data, link to user, save to database.
- **Output**: Updated list or detail of expenses.

4.1.3 Requirement 3 – Budget Management

- **Description**: Users define or modify spending limits for each category.
- **Inputs**: Category, limit amount.
- **Process**: Check if budget exists; if yes, update limit; if no, create new record.
- **Output**: New or updated budget record.

4.1.4 Requirement 4 – Financial Summaries

- **Description**: Provide aggregated views of spending data (e.g., monthly totals, category breakdown).
- **Inputs**: Date range or user ID to filter data.
- **Process**: Query expenses, group by category/date, compare with budget.
- **Output**: Summaries, charts, or alerts if nearing budget.

4.2 Software Interface Description

4.2.1 External Machine Interfaces

- **Database**: The application interfaces with MongoDB (or a similar database) to store and retrieve user data.
- **Optional External APIs**: Potential integration with banking services for automated transaction imports.

4.2.2 External System Interfaces

- **Deployment Environment**: The system can be hosted on a cloud platform (e.g., AWS, Heroku, or Render) with possible load balancing or containerization (Docker).

4.2.3 Human Interface

- **Front-End UI**: A web-based dashboard (HTML/CSS/JavaScript frameworks) that provides forms for expenses, budget pages, and summary views (charts/tables).
- **Mobile Access** (Optional): A responsive design or dedicated mobile app could be provided.

5.0 Restrictions, Limitations, and Constraints

- **Time & Scope**: Project must be delivered by semester’s end; advanced features (bank integration, ML-driven insights) may be postponed.
- **Security**: Must use secure password hashing (e.g., bcrypt) and token-based access; requires HTTPS in production to protect data in transit.
- **Deployment**: Likely constrained to free or low-cost hosting services, which may limit performance under high load.
- **Team Expertise**: Minimal prior experience with some technologies requires careful planning to manage the learning curve.