SOFTWARE PROJECT PLAN

1.0 Introduction

This document provides an overview and detailed plan for the development of the Personal Finance and Budgeting Application. The application aims to help users securely track their daily expenses, set spending limits, and gain insights into their financial habits.

1.1 Project Scope

The Personal Finance and Budgeting Application is a web-based system that enables registered users to:

- **Create Accounts and Log In**: Secure user registration and authentication.
- **Manage Expenses**: Users can add, categorize, and track expenses over time.
- **Set Budgets**: Users define spending limits for specific categories (e.g., Groceries, Entertainment).
- **View Reports and Summaries**: Simple data visualizations (tables, charts) to show spending trends and highlight when a budget threshold is approached.

Major inputs include **expense details** (amount, date, category, and optional notes) and **budget parameters** (category and limit). The system processes these inputs to store data, calculate spending vs. budget ratios, and generate relevant summaries. Outputs include **spending summaries**, **budget status**, and **alerts** (if integrated) about nearing or exceeding budgets.

1.2 Major Software Functions

- **User Management**: Registration, login, and session handling using secure tokens (JWT).
- **Expense Management**: Create, retrieve, update, and delete (CRUD) expense records.
- **Budget Management**: Define and maintain budgets per category, monitor actual spending vs. budgeted amounts.
- **Analytics & Visualization** (basic level): Provide charts/tables to display spending distribution and identify trends over time.
- **Optional Extended Functions**: CSV import for expenses, external bank API integration (e.g., Plaid), or notifications for budget limits.

1.3 Performance/Behavior Issues

- **Response Time**: The system should fetch and display expense data within a few seconds, even if the user has a large number of transactions.
- **Scalability**: The back end should be able to handle multiple concurrent requests if the user base grows.
- **Security and Privacy**: Passwords must be encrypted. Only authenticated users can access or modify their own data.

1.4 Management and Technical Constraints

- **Limited Team Size**: The project is developed by a small team, so feature scope must be realistic.
- **Time Constraints**: The application must be delivered before the end of the semester, with key milestones set by the course timeline.
- **Technical Stack**: The back end is built in Node.js/TypeScript (though Python or other languages could be used), and the front end can be React or a similar framework.
- **Budget**: As a student project, free tiers of hosting and databases (e.g., MongoDB Atlas free tier) will be used to avoid costs.

2.0 Risk Management

This section identifies potential risks and outlines how to mitigate, monitor, and manage them.

2.1 Project Risks

- **R1**: Team members lack experience with Node.js/TypeScript, causing delays.
- **R2**: Security vulnerabilities (weak JWT handling or password hashing) could expose user data.
- **R3**: Data model or schema misalignment leading to incorrect budget/expense calculations.
- **R4**: Time constraints may lead to incomplete or untested features.
- **R5**: Scalability issues if the app is used by many concurrent users (e.g., performance bottlenecks).

2.2 Risk Table

Risk	Probability	Impact	Reference
R1: Node.js/TS inexperience	Medium	Medium	RM3 R1
R2: Security vulnerabilities	Medium	High	RM3 R2
R3: Data model misalignment	Low	Medium	RM3 R3
R4: Time constraints	High	High	RM3 R4
R5: Scalability issues	Low	Low	RM3 R5

2.3 Overview of Risk Mitigation, Monitoring, Management (RM3)

1. **Mitigation**:
    - **R1**: Provide tutorials and short training sessions on Node.js, TypeScript, and REST best practices.
    - **R2**: Implement secure coding practices, code reviews, and library-based authentication/authorization.
    - **R4**: Break down tasks into smaller pieces with early prototypes to avoid last-minute rush.
2. **Monitoring**:
    - Regular check-ins to see if tasks are on schedule, including code reviews and testing for security vulnerabilities.
    - Performance testing for concurrency and data model validation using test sets.
3. **Management**:
    - Document all identified risks.
    - Assign responsible persons for each risk, ensuring clear accountability.
    - Update the risk table whenever a new risk arises.

3.0 Project Schedule

3.1 Project Task Set

The project uses a **modified agile** approach with short sprints. High-level tasks include:

1. **Requirements & Planning**: Define feature scope, user flows, and data model.
2. **Environment Setup**: Configure Node.js/TypeScript, database, and repository structure.
3. **Core Features**: Implement user authentication, expense tracking, and budgets.
4. **Extended Features**: Data visualization, alerts/notifications, integration with external APIs (if time permits).
5. **Testing & QA**: Unit tests, integration tests, security checks, and user acceptance testing.
6. **Deployment & Documentation**: Deploy to a free tier hosting service, prepare user guide, final demo.

3.2 Functional Decomposition

- **Auth Module**
  - Register, Login, JWT creation, session management.
- **Expense Module**
  - Add expense, update expense, list expenses, retrieve expense by category.
- **Budget Module**
  - Set or update budget limit, retrieve budgets, check usage vs. limit.
- **Reporting/Visualization**
  - Summaries of expenses per category, monthly totals, charts.

3.3 Task Network

1. **Requirement Analysis** → 2. **Database Schema & Models** → 3. **API for Auth** → 4. **API for Expense** → 5. **API for Budget** → 6. **Front-End Integration** → 7. **Testing & Bug Fixes** → 8. **Deployment**.

3.4 Timeline Chart

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Requirements & Planning	X					
Env. Setup & DB Schema		X				
Auth & User Management		X	X			
Expense & Budget Endpoints			X	X		
Front-End Integration				X	X	
Testing & QA					X	X
Deployment & Final Demo						X

3.5 Schedule Compliance

- **Monitoring**: Weekly team meetings to track progress and any deviations from schedule.
- **Reporting**: Team lead updates a shared document or board (e.g., Trello, Jira) with completed tasks and upcoming targets.
- **Controlling**: If certain features lag, scope may be reduced or simplified to meet final deadlines.

4.0 Appendix

Additional materials such as:

- **Wireframes** or UI mockups of the expense/budget pages.
- **Database ER Diagrams** showing how user, expense, and budget data connect.
- **References to External APIs** if bank integration is pursued.