Software Requirements Document (SRD)
Expense Manager 2.0

1. Executive Summary

Expense Manager 2.0 is a comprehensive full-stack web application designed to help users track, manage, and analyze their personal finances. The application provides features for transaction management, budget planning, financial reporting, and goal setting with a modern, responsive user interface.

2. Project Overview

2.1 Purpose
The primary purpose of Expense Manager 2.0 is to provide users with a complete financial management solution that enables them to track income and expenses, set and monitor budgets, analyze spending patterns, set and achieve financial goals, and generate comprehensive financial reports.

2.2 Scope
The application covers the complete lifecycle of personal financial management, from basic transaction tracking to advanced budgeting and goal-setting features.

2.3 Target Users
- Individual users seeking personal finance management
- Users who want to track daily expenses and income
- Users who need budget planning and monitoring
- Users who want to set and achieve financial goals

3. System Architecture

The system uses a full-stack architecture with React (frontend), Express.js (backend), and MongoDB (database). Authentication is handled via JWT. The application follows SPA (Single Page Application) and RESTful API patterns.

4. Functional Requirements

Includes modules for User Authentication, Transaction Management, Budget Management, Financial Goals, Reporting & Analytics, Data Export, and User Profile Management.

5. Non-Functional Requirements

Performance, Security, Usability, Reliability, and Compatibility requirements are defined, ensuring system efficiency, safety, and accessibility.

6. User Interface Requirements

Application design follows Tailwind CSS, responsive layouts, and modern UI principles. Includes navigation sidebar, interactive charts, and paginated tables.

7. Data Requirements

Database schema includes User, Transaction, Budget, and Goal models. Data validation ensures proper structure and integrity.

8. Integration Requirements

Includes email service (SMTP), file storage, and MongoDB Atlas. Follows RESTful API design with proper error handling.

9. Deployment Requirements

Frontend hosted on GitHub Pages, backend on cloud, and database on MongoDB Atlas. SSL support is required.

10. Testing Requirements

Includes unit, integration, end-to-end, and performance testing with at least 80% coverage.

11. Maintenance Requirements

Monitoring, logging, updates, and database optimization are required.

12. Constraints and Limitations

Includes browser compatibility, file size limitations, API rate limits, hosting budget, and development constraints.

13. Assumptions and Dependencies

Assumes basic user internet and browser capabilities. Depends on MongoDB Atlas, SMTP, GitHub Pages, and npm registry.

14. Risk Assessment

Technical risks include database failures, email outages, API changes, and vulnerabilities. Mitigation includes retries, redundancy, updates, and audits.

15. Success Criteria

Defines functional success (auth, transactions, budgets) and non-functional success (performance, security, usability, compatibility).

16. Future Enhancements

Planned features: multi-currency support, bank integration, receipt scanning (OCR), advanced analytics, mobile app, social sharing, investment tracking, and tax reporting.

17. Document Approval

Role	Name	Date	Signature
Project Manager	[To be filled]	[To be filled]	[To be filled]
Technical Lead	[To be filled]	[To be filled]	[To be filled]
Product Owner	[To be filled]	[To be filled]	[To be filled]
QA Lead	[To be filled]	[To be filled]	[To be filled]

18. Document Version History

Version	Date	Author	Changes
1.0	[Date]	[Author]	Initial document creation
2.0	[Date]	[Author]	Updated based on codebase analysis