Software Requirements Specification

Financial Management: Full Stack Web Application

HashSlingingSlashers

Jeffery Kimbrow, Madeline Brothers, Yasmin Hassan, and Samantha Saunsaucie

Table of Contents

**1 Introduction3**

1.1 Overview4

1.2 Goals and Objectives5

1.3 Scope5

1.4 Definitions6

**2 Design Constraints and Assumptions7**

2.1 Constraints7

2.2 Assumptions8

**Non Functional Requirements8**

Type chapter title (level 2)9

Type chapter title9

**Functional Requirements8**

Type chapter title (level 2)9

Type chapter title9

[Revision History 2](#_Toc785484496)

[1 Introduction 3](#_Toc1492128250)

[1.1 Overview 3](#_Toc202871115)

[1.2 Goals and Objectives 4](#_Toc423816327)

[1.3 Scope 4](#_Toc2070547139)

[2 Design Constraints and Assumptions 5](#_Toc2130720646)

[2.1 Design Constraints 5](#_Toc1719037807)

[2.2 Assumptions 5](#_Toc2068307817)

[3 Non-Functional Requirements 6](#_Toc761572177)

[Functional Requirements 6](#_Toc528787259)

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Name | Description |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 0.4 | 09/25/25 | Madeline | Added nonfunctional requirements |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 1 Introduction

### 1.1 Overview

The Financial Management: Full Stack Web Application is designed as a cross-platform solution, accessible from both mobile devices and web browsers, provided users have stable internet connection. Its primary purpose is to empower users to manage their finances with ease and clarity. Through the application, users can track budgets, categorize their spending into meaningful categories (e.g., transportations, groceries, shopping), and review account statements. Additionally, the system allows users to transfer funds between accounts—for example, moving money from a checking account to a savings account with just a few clicks. A key feature of the application is an integrated AI chatbot to provide users with quick, personalized feedback. Users can ask a range of finance related questions, and the chatbot will aim to deliver clear, accurate answers to support decision-making.

This document outlines the detailed requirements for the application. The introduction defines the project goals, objectives, and scope. Section two describes the design constraints and assumptions guiding development. Section three covers non-functional requirements, which specify measurable system characteristics such as usability, reliability, and performance. Finally, section four focuses on functional requirements, presented through use cases that illustrate how the system responds to user actions and supports them in completing their tasks.

### 1.2 Goals and Objectives

One key goal of the project is to ensure broad accessibility. This includes making the application attainable across multiple operating systems and devices, ensuring that the text and interface elements are consistently visible, readable, and user-friendly. Accessibility also involves accommodating multilingual users by providing a multi-language interface so that users from different regions can interact with the system comfortably and confidently.

A second goal is security. The application must protect user data through a well-designed security framework. This include performing regular security audits, enforcing secure password policies with strong encryption, hashing stored passwords, and ensuring that all session transmissions are protected against interception (e.g. session hijacking).

A third major goal is to leverage a cloud-based deployment environment. The objective is to ensure that user data is backed up regularly and that the application remains available to users anywhere, provide they have internet access. This idea supports scalability, in the sense that the system can handle an increasing number of users and data overtime.

### 1.3 Scope

The Financial Management: Full Stack Web Application is a cross-platform system designed to help users manage their personal finances securely and conveniently. The types of users expected to interact with the program are administrators, internal administrators, customers, and multilingual users. External users are able to create accounts, update profiles, review transaction history, track budgets, and transfer funds between accounts through a dashboard or mobile interface. Internal administrators can assess system stability, system security, data integrity, fix bugs, update system configurations, and implement new features.

The system being deployed in a cloud environment ensures availability, scalability, and regular data backups. Security is a core focus to guarantee strong encryption, secure authentication, and hashed passwords for the protection of user data.

The technology stack will include Python, C, and Java for backend services, PostgreSQL for data management, and JavaScript, HTML, and CSS for the frontend interface.

## 2 Design Constraints and Assumptions

### 2.1 Design Constraints

**Scheduling:**

Each team member is a full-time student with additional extracurricular and personal responsibilities. Coordinating a consistent meeting schedule can be challenging, and individual availability may vary week to week. While the team strives for flexibility, some scheduling conflicts will be unavoidable.

**Timeframe:**

The project must be completed within a four-month development timeframe. Building an application from the ground up, while balancing diverse team schedules, limits the time available for meetings, development, and testing. This constraint requires careful planning and prioritization of deliverables.

**Security and Compliance:**

Because the application handles personal user data, security governance and risk management are critical. The team must ensure that sensitive data is stored, transmitted, and processed responsibly, with proper encryption, authentication, and compliance with relevant privacy standards to prevent potential harm.

**User Accessibility and System Reliability:**

Ensuring that the system functions correctly across all supported operating systems and devices is a challenge. Additionally, cloud service reliability may introduce risks, such as downtime or latency, which affect users' experiences. The team must account for potential errors and implement measures to maintain consistent availability.

### 2.2 Assumptions

The following assumptions are made about how the system will be accessed and used:

* Users have an active bank account with Commerce and valid credentials to access their account information.
* Users have a device capable of connecting to the internet and initiating the application, whether through a web browser or mobile platform.
* Users have basic familiarity with online financial environments or similar systems, allowing them to navigate the system easily and without much guidance.

## 3 Non-Functional Requirements

* 1. Useability: The system must be intuitive and easy to navigate for users with varying levels of technical proficiency. Interfaces should follow consistent design and provide help prompts with choice of AI assistance. A new user should be able to perform basic tasks, such as checking balances, creating a budget, or locking a card within the first five minutes of use.
  2. Efficiency: The application should respond to user requests within two seconds under normal operating conditions. Resource utilization (CPU, memory, network) must remain within acceptable limits to support concurrent users.
  3. Security: All data must be encrypted in transit and at rest. The system must enforce strong password policies, session timeouts, and multi-factor authentication. Regular testing and code scanning will be conducted.
  4. Availability: the application should have a 99.9% uptime, excluding scheduled maintenance. Cloud-based redundancy and failover strategies must ensure continuity of service.
  5. Reliability: Backups must be performed daily and retained according to policy. The system should recover from failure without data loss beyond the most recent backup. Error handling must gracefully notify users without exposing sensitive information.

## Functional Requirements

* 1. Use Case 1

Title: Registration   
Actors: Mobile User

Precondition: User is at the login page

Postconditions: User is informed if account is activated or not

Basic Flow:

1. User presses “create account” button
2. The system initiates a new page for user to enter in personal and banking information
3. User enters personal and checking/saving accounts information
4. System processes both personal and banking information
5. The system verifies the user, and the new account is now active

Alternative Flow:

1. The system is unable to verify the user’s information due to invalid data

2. System displays an error message corresponding to invalid information and prompts user to try again

3. User can make another attempt to create account

2.2 Use Case 2

Title: Login

Actors: Mobile and Web Users

Precondition: User must have an active account to log in

Postcondition: User successfully logs in

Basic Flow:

1. User enters their information and clicks the “log in” button

2. The system process and verifies the user’s information

3. The system directs the user to the dashboard

Alternate Flow:

1. User forgets their username and selects “forgot username”
2. The system directs the user to the “forgot username” page
3. User enters valid full name and email address
4. The system verifies the user’s information and emails the user their correct username
5. User forgets their password and selects “forgot password”
6. The system directs the user to the “forgot password” page
7. User enters information
8. System verifies user’s information and directs user to a new page to reset password

2.3 Use Case 3

Title: User Profile Settings

Actors: End User

Preconditions: User has a valid account in system and logged into the system.

Postconditions: User is able to modify account profile settings.

Basic Flow:

1. User selects their profile icon.

2. User is displayed a “Edit Profile” page

3. User is able to update profile data. (e.g., change phone number, email, or name spelling.

4. User clicks save/ complete after modification and data is saved.

Alternate Flows:

1. User has valid account.

2. User updates data that could be parsed as malicious versus true information.

3. System guardrails stop the user from executing.

4. Users are not allowed to make improper input.

2.3 Use Case 4

Title: Dashboard

Actors: Web User

Preconditions: User has a valid account and logged into the system

Postconditions: User is able to navigate the web application in under 10 minutes

Basic Flow:

1. The system presents the homepage/dashboard page.
2. User can choose from the various drop-down options: budget tracking, transaction history, payment, etc., or they can search for an option
3. The system directs the user to the selected web page.
4. The system allows for user to be directed to homepage at any time.

Alternate Flow:

1. User inputs an invalid data type or an invalid option in the search bar
   1. The system will print an error message, informing user of mistake
   2. The user will receive a second attempt to input data

2.4 Use Case 5

1. Title: Initiate Backup
2. Actors: Admin/DevOps
3. Preconditions: Admin authenticated with elevated roles.
4. Postconditions: Backup artifact(s) created, encrypted, indexed with retention tag

Basic Flow:

1. Admin triggers/ schedules backup.
2. System snapshots DB/files
3. Encrypts backup
4. Uploads to cloud; logs results

Alternate Flow:

1. Storage quota exceeded. Abort, alert raised.  
 2. Partial shared failure. Retry failed parts, mark as degraded if unresolved.

2.5 Use Case 6  
 Title: Restore from Backups  
 Actors: DevOps  
 Preconditions: Valid backup present; runs integrity  
 Postconditions: Target environment restored to selected point in time; integrity is verified.

Basic Flow:

1. Admin selects backup and timestamp
2. System restores to staging environment
3. Integrity Checks and smoke tests run (QA).
4. Admin promotes to production

Alternate Flows:

1. Integrity check fails: Attempt prior backup; open incident (WebHook)
2. Schema Drift: System runs migration shim before promoting.

2.6 Use Case 7  
 Title: Promote Build via Docker to Cloud  
 Actors: DevOps, CI/CD System  
 Preconditions: Valid build artifact exists in the repository. Target cloud environment (AWS/Azure VM/ Docker host) is reachable. Environment secrets are available and configured.  
 Postconditions: New version running in target environment with health checks green. Rollback plan remains intact.

Basic Flow:

1. CI Builds and signs Docker image, then pushes it to the registry.
2. System restores to staging environment with environment secrets.
3. Canary deployment runs and passes health and SLP checks.
4. System promotes build to Production
5. Release notes are recorded

Alternate Flows:

1. Health check fails: System performs auto-rollback to prior image.
2. Secret injection failure: Deployment halted; pipeline blocks util corrected.

2.7 Use Case 8  
Title: Run OWASP ZAP in CI  
Actors: CI/CD System, Security Engineer  
Preconditions: Staging environment is running and accessible. Authentication context (e.g., login script) is configured. OWASP ZAP installed and integrated with CI/CD pipeline.   
Postconditions: Dynamic Application Security Testing (DAST) report archived. Authentication context is seeded via login script. Scan completes and generates JSON/HTML report. Pipeline evaluates severity, sets pass/fail status, and files findings.

Basic Flow:

1. CI Builds and signs Docker image, then pushes it to the registry.
2. System restores to staging environment with environment secrets.
3. Canary deployment runs and passes health and SLO checks.
4. System promotes build to Production
5. Release notes are recorded

Alternate Flows:

1. Authentication fails (cannot reach authenticated pages): Pipeline marks scan invalid and requires rerun with corrected credentials.
2. Scanner timeouts: CI retries with tuned spider limits; if failure persists, job fails safely.