Software Requirements Specification (SRS) Template

Items that are intended to stay in as part of your document are in **bold**; explanatory comments are in *italic* text. Plain text is used where you might insert wording about your project.

The document in this file is an annotated outline for specifying software requirements, adapted from the IEEE Guide to Software Requirements Specifications (Std 830-1993).

Tailor this to your needs, removing explanatory comments as you go along. Where you decide to omit a section, you might keep the header, but insert a comment saying why you omit the data.

Scholar Savings

Software Requirements Specification

Document

Version: (n) **Date: (mm/dd/yyyy)**

Table of Contents

1. Introduction

The following subsections of the Software Requirements Specifications (SRS) document should provide an overview of the entire SRS.

1.1 Purpose

The purpose of this SRS is to describe all functional and non-functional requirements for Scholar Savings, a web-based financial literacy app that helps college students budget, save, and learn about investing through interactive tools and AI-powered recommendations. This document is intended for developers, testers, and stakeholders involved in maintaining or enhancing the product.

1.2 Scope

Scholar Savings is a Blazor/.NET web application deployed on Azure. It provides two primary modules: a Budget Coach for expense tracking and personalized savings tips, and an Investment Coach that suggests stocks and ETFs based on user goals. The software uses OpenAI's API for natural-language interaction and Azure for authentication and data storage. It does not execute real trades or store bank credentials.

1.3 Definitions, Acronyms, and Abbreviations.

AI – Artificial Intelligence

API – Application Programming Interface

UI – User Interface

SRS – Software Requirements Specification

Azure OpenAI – Microsoft cloud service providing access to OpenAI models

1.4 References

Microsoft Azure Documentation

OpenAI API Reference

1.5 Overview

The remainder of this SRS details the overall description of Scholar Savings, its major functions, specific requirements, and software attributes.

2. The Overall Description

Describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in section 3, and makes them easier to understand.

2.1 Product Perspective

Scholar Savings is a standalone web application built using Blazor Server and C#. It integrates external APIs (OpenAI API).

2.1.1 System Interfaces

List each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system.

2.1.2 Interfaces

The interface is a web-based UI built with Blazor Server. Users interact through forms, buttons, and chat boxes. The design prioritizes simplicity and accessibility for students, using menus, minimal text input, and responsive layouts.

2.1.3 Hardware Interfaces

Scholar Savings runs on standard desktop or laptop computers with an internet connection. No special hardware is required. It supports all major browsers on both Windows and macOS systems.

2.1.4 Software Interfaces

OpenAI API (v1) – provides AI-generated text responses; source: OpenAI.

2.1.5 Communications Interfaces

Scholar Savings communicates over HTTPS using RESTful API calls. All communication between the client, server, and external APIs. Azure manages the web server connection and handles SSL certificates automatically.

2.1.6 Memory Constraints

The app requires minimal memory since most data is cloud-hosted.

2.1.7 Operations

The system operates interactively whenever a user is logged in. There are no unattended or batch operations. Azure automatically performs backups of the hosted database, ensuring recovery if necessary.

2.1.8 Site Adaptation Requirements

(1) Scholar Savings is cloud-based and requires no on-site configuration. Site adaptation may involve updating API keys or environment variables for different deployment environments (development, test, or production).

2.2 Product Functions

Scholar Savings provides students with interactive tools for managing their personal finances and learning about investing. The system's main functions include:

1. ***Account Management:*** *Users can register, log in, and manage their profiles securely through Azure authentication.*
2. ***Budget Coach:*** *Users can input income and expenses, view categorized spending summaries, and receive AI-generated budgeting tips.*
3. ***Investment Coach:*** *The app uses the OpenAI API to explain investment concepts and suggest sample stocks or ETFs for educational purposes.*

4. ***Data Visualization:*** *Spending and savings trends are displayed through dynamic charts and graphs.*
5. ***Goal Tracking:*** *Users can set monthly savings goals and track their progress through notifications and visual indicators.*

2.3 User Characteristics

The target users are college students and young adults who may have limited experience with budgeting or investing but are familiar with basic web applications.

- *Experience Level: Beginner to intermediate computer users.*
- *Financial Knowledge: Low to moderate.*
- *Accessibility Needs: Interface is designed with large text, clear navigation, and optional dark mode for readability.*

2.4 Constraints

Time Constraints: *Initial version was developed during a 24-hour hackathon, limiting advanced features such as multi-currency support or real-time financial synchronization.*

2.5 Assumptions and Dependencies

- *The Azure App Service and OpenAI API will remain available and stable.*
- *Users will enter accurate personal finance data for reliable feedback.*
- *Any third-party APIs will maintain consistent endpoints and response formats.*
- *Future iterations may rely on additional APIs (e.g., Plaid or Yahoo Finance) to expand functionality.*

2.6 Apportioning of Requirements.

Identify requirements that may be delayed until future versions of the system.

3. Specific Requirements

This section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output. The following principles apply:

(1) Specific requirements should be stated with all the characteristics of a good SRS
- *correct*
- *unambiguous*
- *complete*
- *consistent*
- *ranked for importance and/or stability*
- *verifiable*
- *modifiable*
- *traceable*

(2) Specific requirements should be cross-referenced to earlier documents that relate
(3) All requirements should be uniquely identifiable
(4) Careful attention should be given to organizing the requirements to maximize readability

Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in the following subclasses.

3.1 External Interfaces

User Interface:
Accessible through a modern web browser using the Blazor Server front end. Users interact with navigation menus, chat boxes, forms, and dashboards. The interface supports responsive layout for desktop and mobile browsers.

OpenAI API Interface:
Used to generate AI-powered budget and investment guidance. JSON-formatted user queries are sent to the API, and responses are parsed into natural-language text displayed in the chat UI.

3.2 Functions

1. ***User Account Management***
    - *The system shall allow users to register, log in, and log out using Azure Authentication.*
    - *The system shall securely store user credentials using salted hashing.*
    - *The system shall allow users to update or delete their accounts.*
2. ***Budget Management***
    - *The system shall allow users to create, edit, and delete budget categories.*
    - *The system shall calculate total expenses and remaining balance automatically.*
    - *The system shall display budget data in tabular and chart form.*
    - *The system shall allow exporting budget data to CSV format.*
3. ***AI Budget Coach***
    - *The system shall allow users to input natural-language financial questions.*
    - *The system shall send queries to the OpenAI API and return AI-generated responses.*
    - *The system shall provide at least one actionable tip per conversation.*
    - *The system shall store chat history for future reference.*
4. ***Investment Coach***
    - *The system shall provide general investment education, not financial advice.*
    - *The system shall display sample stock data with current prices and trends.*
5. ***Goal Tracking and Visualization***
    - *The system shall allow users to set monthly savings goals.*
    - *The system shall visualize progress using progress bars and notifications.*
    - *The system shall provide AI feedback when users exceed or miss goals.*

3.3 Performance Requirements

The system shall maintain an uptime of 99%, excluding scheduled Azure maintenance.

All API responses shall be processed and displayed within 5 seconds of receipt.

3.4 Logical Database Requirements

- *No current database, work in progress*

3.5 Design Constraints

The system must be developed using C# and Blazor Server Framework.

The system must deploy on Microsoft Azure App Service.

The system must integrate with OpenAI API (v1)

The codebase shall comply with Microsoft .NET coding standards.

3.5.1 Standards Compliance

The software shall produce properly formatted reports and logs consistent with project conventions.

3.6 Software System Attributes

There are a number of attributes of software that can serve as requirements. It is important that required attributes by specified so that their achievement can be objectively verified. The following items provide a partial list of examples.

3.6.1 Reliability

The system shall recover from API or network failures without data loss. Azure automatically performs service restarts and data redundancy for high reliability.

3.6.2 Availability

The system shall maintain 24/7 availability through Azure cloud hosting with a minimum uptime target of 99%.

3.6.3 Security

All communications between the client and server shall use HTTPS

3.6.4 Maintainability

*Future updates can be made without major restructuring of existing modules.*3.6.5 Portability