



Project: Enterprise Financial Application

(Household Budgeter)

SOFTWARE REQUIREMENTS SPECIFICATIONS

VERSION 3.2

Note: This template is largely based on the template provided by the Unified Process for Education (www.upedu.com) with minor modifications.

CODER FOUNDRY
1231 SHIELDS ROAD SUITE 5
KERNERSVILLE, NC 27284
PHONE: 336.231.8632

Revision History

Date	Version	Description	Author
08/13/16	2.3	Removed Third party logins as a requirement	Antonio Raynor
03/03/18	3.0	Updated for Coder Foundry	Antonio Raynor
04/16/18	3.1	Project modified to Enterprise level application	Antonio Raynor
07/21/18	3.0	Supplemental project(s) documentation and details	Antonio Raynor

Table of Contents

Revision History	2
Table of Contents	3
1. Document Overview	4
Purpose	4
2. Project Mission Statement	4
Project Introduction	4
Stakeholders	5
Assumptions and Constraints	5
3. Requirements	5
Functional Requirements	5
End User Requirements	5
User Interface Requirements	6
Non-functional Requirements	7
Security Requirements	7
Software Quality Attributes	7
Business Requirements	7
4. Classification of Functional Requirements	7
End User Requirements	7
User Interface Requirements	8
5. Appendices	8
Appendix A: User Interface Screen Mockups	9
Accounts	9
Budget	10
Dashboard	11
Household	12
Transactions	13
Appendix B: User Processes Interface / Features Map	14
Appendix C: Glossary	14

1. Document Overview

Purpose

This document is intended as a guide to Coder Foundry students for developing a modular enterprise level financial application built using the tools and skillset achieved up to this point in the course. Technologies and frameworks include but is not limited to: Microsoft MVC with C#, Entity Framework, Code-First database migration, LINQ, Web API2, SQL, Database Stored Procedures, Xamarin, XAML, JSON & Http.

As with most enterprise applications, there are many requirements and multiple areas of design and logic. For this application there are 3 separate project phases with three individual architectures/design patterns.

Enterprise Financial Application		
Phase I	Web Application	MVC
Phase II	Web API2 REST Application	SOA – Application Service
Phase III	Mobile Cross Platform Application	MVVM

This document describes the **MVC web application** project specification and includes requirements and sample screen mockups.

Section 2 discusses the project's purpose, stakeholders, and any assumptions and constraints under which the student is expected to operate.

Section 3 identifies and discusses all project requirements (business, functional, and non-functional).

Section 4 provides classification and prioritization of the functional requirements.

Section 5 is the Appendices section, which includes screen mockups and any additional relevant information.

2. Project Mission Statement

Project Introduction

The Household Budgeter is a Web-based software application designed for creating and managing budgets for household finances and the balancing of bank accounts. It must function as an online service that may be used in multiple households/families throughout the world.

The purposes of all Coder Foundry student projects are two-fold: 1) teach students relevant and marketable coding skills and best practices, and 2) serve as audition pieces for prospective employers. As such, it is imperative that each student project be of a type that is immediately recognizable and relatable to prospective employers. If a prospective employer has to ask questions such as: "Why did you write this?" or "What is this used for?" or "How does this work?", then we have not adequately met the needs of our students.

The Household Budgeter project is similar in nature to many such commercial-grade products (i.e. *AceMoney*, *BudgetPulse*, *Moneydance*, *Pocketsmith*, *QuickBooks*, *EveryDollar*, etc.) with which most employers are familiar.

Stakeholders

This project has no direct client. The stakeholders in this case are the student and the Coder Foundry administrative staff, who will be responsible for grading the project as Pass/Fail.

Assumptions and Constraints

This project is to be built as a Web-based application using the Microsoft Visual Studio IDE (any edition is acceptable, though we encourage the use of the Community edition). Student projects will be hosted online through Microsoft Azure web hosting, with a link to each student's version of the project placed on the student's personal website.

3. Requirements

This section describes all requirements, both functional and non-functional, for this project as set forth by Coder Foundry staff. All requirements outlined herein must be met and demonstrated in the resulting software application for the project to be considered "complete." Any missing requirements/features will result in the project being deemed inadequate.

Functional Requirements

End User Requirements

End users must be able to perform the following actions:

Req. #	Description	Purpose
1	Create bank accounts	Track the transactions that belong to the accounts
2	Delete bank accounts	Get rid of all account data
3	Create, edit, and delete transactions	Maintain accurate record of monthly account transactions
4	Search transaction history by transaction properties	Locate specific transactions
5	View account balance	Prevent overdraft
6	Reconcile transactions with my bank statements	Identify cleared transactions Spot fraudulent activity on accounts
7	Invite other users to a household	Share account data with a spouse
8	Leave a household	Prevent sharing of account data with other users
9	Record household income and expenses	Create a budget
10	Categorize transactions	Keep spending within budget limits
11	View monthly expenses vs. budget	Control monthly spending

User Interface Requirements

To facilitate intuitive and user-friendly interaction, the user interface must be designed according to the following requirements:

Req. #	Page Description	Feature List
1	Login / Register	1) Anonymous users must be directed here when accessing the site
2	Dashboard	1) Users are directed here on successful login 2) Display the most recent transactions for the user's household 3) Display data visualization of spending versus budget for the current budget period (month) 4) Display account balances
3	Account Management	1) Account creation 2) Account deletion 3) Navigate to transaction management page for account
4	Transaction Management	1) Paged listing of transactions 2) Find specific transactions by: <ul style="list-style-type: none"> • Amount Range • Reconciliation Status • Date Range • Description • Budget Category 3) Create/Edit/Delete transactions 4) Reconcile transactions using any arbitrary amount found on the user's bank statement 5) Sort transactions by their various properties 6) View the actual and reconciled account balances
5	Household Management	1) View members of household 2) Leave the current multi-member household 3) Invite other users to join the household
6	User Profile	1) Change password 2) Edit profile information (name, email, etc.) 3) Associate profile with social network identities
7	Budgets	1) Create a monthly budget 2) Add/delete/edit budget categories

Non-functional Requirements

Non-function requirements are specific requirements, usually set forth by the client or by third-party product dependencies, which are not directly related to application features. These may include, but are not limited to, performance, security, and software quality requirements.

Security Requirements

This project must be fully secured to prevent unauthorized persons from accessing user profile and account information. Stored Procedures or LINQ, among other considerations, may be used to prevent SQL injection attacks.

Software Quality Attributes

This project is expected to appear and function as a comparable commercial-grade product. It is intended to serve as an audition piece to prospective employers, and therefore must adhere to commercial standards of software development.

Business Requirements

For this version of the software, developers need only consider currency for their own locale, though an application that is versatile enough to be used regardless of currency form would be more desirable.

4. Classification of Functional Requirements

All software requirements are categorized according to their priority (there is only one non-essential requirement for this project):

- 1) Essential
- 2) Desirable
- 3) Optional

End User Requirements

Description	Priority
Create bank accounts	Essential
Delete bank accounts	Essential
Create, edit, and delete transactions	Essential
Search transaction history by transaction properties	Essential
View account balance	Essential
Reconcile transactions with my bank statements	Essential
Invite other users to a household	Essential
Leave a household	Essential
Record household income and expenses	Essential

Categorize transactions	Essential
View budget overview / outlook	Essential

User Interface Requirements

Description	Priority
Anonymous users must be directed to Login when accessing the site	Essential
Users are directed to Dashboard on successful login	Essential
Display the most recent transactions for the user's household	Essential
Display data visualization of monthly spending for budget period (i.e. month)	Essential
Display account balances	Essential
Account creation	Essential
Account deletion	Essential
Navigate to transaction management page from Account Management	Essential
Paged listing of transactions	Essential
Find specific transactions by: <ul style="list-style-type: none"> • Amount Range • Reconciliation Status • Date Range • Description • Category 	Essential
Create/Edit/Delete transactions	Essential
Reconcile transactions against amount found on user's bank statement	Essential
Sort transactions by their various properties	Essential
View the actual and reconciled account balances	Essential
View members of household	Essential
Leave the current multi-member household	Essential
Invite other users to join the household	Essential
Edit profile information (username, email, password etc.)	Essential

5. Appendices

This section consists of two appendices:

- A) user interface screen mockups
- B) user processes interface / features map

Appendix A: User Interface Screen Mockups

Accounts

Account	Balance	Reconciled
<input checked="" type="checkbox"/> Checking	\$1,234.56	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> Savings	\$12,345.67	<input type="checkbox"/>

Button 1 - Transactions
Button 2 - Delete

Budget

A Web Page

http://

My Budget

Income

Description	Category	Amount	Annual Frequency
Delete Husband Salary	Salary	10000	12
Delete Wife Salary	Salary	3000	12

Expenses

Description	Category	Amount	Annual Frequency
Delete Mortgage	Bills	600	12
Delete Light Bill	Bills	115	12
Delete Water Bill	Bills	15	12
Delete Garbage	Bills	4.5	4

Add Income / Expense

☒ Expense ☐ Income

Category: Bills

Description: Mortgage

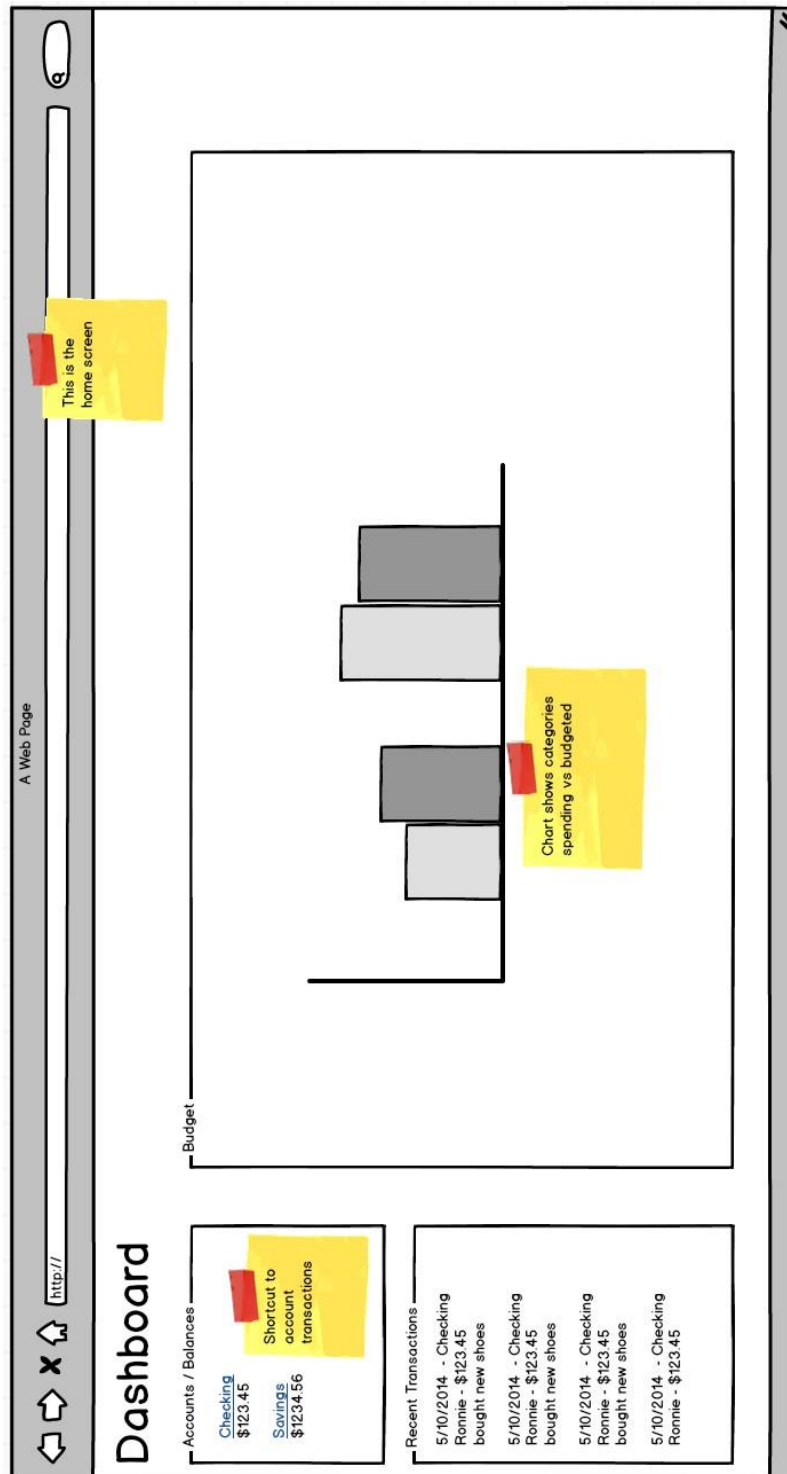
Amount: 1234.56

Annual Frequency: 12

[Save](#)



required
Type ahead finds existing
categories in household.
new categories are stored

Dashboard



Household

A Web Page



Household

Members

Name	Email
Ronnie Overby (You)	ronnie@ronnieo.net
Tina Overby	tinaann1821@gmail.com

Invite

Invite another user to join your household.

Email

Leave

Leave your current household and start a new one.

☐ I understand that I will no longer have access to my current household's accounts.

About Households

Households are how we group accounts and users.

Each user belongs to only 1 household, but the household can have many users.

Each user in your household has full access to all account data.

Transactions

Transactions in Checking

Balance

Reconciled

\$2,684.31

\$3,188.98

Edit Transaction

Description

some expense

Amount

\$ 67.79 + ☒

Reconciled Amount

\$ 67.79

Category

Gas

Date

5/10/2014

Toggles reconciliation (use icon)

+ / - (toggles between deposit/purchase)

save, discard changes, delete

Search Transactions

Clicking hides edit/create form and shows search form

Date	Description	Category	Amount	Reconciled	Updated By
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe
Edit Delete	New Shoes	Clothing	\$123.45	<input checked="" type="checkbox"/>	Jane Doe

<<

<

6

7

8

9

>

>>

Appendix B: User Processes Interface / Features Map



Appendix C: Glossary

ⁱ What is an **Enterprise Application**?

An enterprise application is a business application, obviously. As most people use the term, it is a big business application. In today's corporate environment, enterprise applications are complex, scalable, distributed, component-based, and mission-critical. They may be deployed on a variety of platforms across corporate networks, intranets, or the Internet. They are data-centric, user-friendly, and must meet stringent requirements for security, administration, and maintenance. In short, they are highly complex systems.