**Functional Specification Document**

“Scrooge” - A small business accounting suite

**Overview**

This document will provide a specification for the “Scrooge” application. It will focus on explaining the use case from a user’s perspective and technical documentation from a developer’s perspective. It will start out by giving an overview about the application itself, to get a sense for scope and scale. The technical specifications given will provide information related to architecture, technology used and conceptional designs.

Table of Contents

[1. General Overview 2](#_Toc444843490)

[1.1 Application description 2](#_Toc444843491)

[1.2 Use case (diagram) 2](#_Toc444843492)

[1.3 User stories 3](#_Toc444843493)

[1.4 Other requirements 3](#_Toc444843494)

[2 Technology overview 4](#_Toc444843495)

[2.1 Frontend 4](#_Toc444843496)

[2.2 Backend 4](#_Toc444843497)

[2.3 Programming languages 4](#_Toc444843498)

[3 UI Overview 5](#_Toc444843499)

[3.1 User Interface Concepts 5](#_Toc444843500)

[3.2 UI Concept Drawing 5](#_Toc444843501)

[4. Architecture overview 6](#_Toc444843502)

[4.1 Basic architecture discussion 6](#_Toc444843503)

[4.2 Service definition 6](#_Toc444843504)

[4.3 Service discovery 6](#_Toc444843505)

[4.4 Service communication 6](#_Toc444843506)

[4.5 Important services 6](#_Toc444843507)

[5. Outlook 7](#_Toc444843508)

[5.1 Further project work 7](#_Toc444843509)

# 1. General Overview

### 1.1 Application description

The application will be designed for use in small to medium companies. It will provide services to support accounting. It assists legal compliance by correctly computing necessary variable values. A focus on simplicity helps reduce the time wasted on notorious accounting tasks.

### 1.2 Use case (diagram)

### 1.3 User stories

# Basestory

A user wants to get away from paper-based accounting. They choose to switch to Scrooge.

# Inflow/Outflow

A user wants to store his inflow/outflow (sells/purchases) information digitally. They use the inflow/outflow feature of Scrooge, adding and removing entries as they like.

# Summing up

A user wants to summarize his inflow/outflow in a simple and useful way. Scrooge allows him to do this by clicking a prominently placed button or looking directly at the summarization tab.

# Managing inventory

A user wants to manage his inventory digitally. Scrooge allows him to do so in an officially recognized way using the inventory tab.

# Creating reports for the tax office

A user wants to create an official report for the taxation office. Scrooge allows him to do this by combining reports in an officially recognized format. To do this, he goes to the report tab, from where he can even export the report in a machine-readable way or print it directly.

### 1.4 Other requirements

The application must be able to…

* be distributed easily (aka. copy application files and run elsewhere)
* perform at least acceptable even on older machines
* require no external dependencies that cannot be easily distributed alongside the application itself
* be downloaded, installed and used without cost for the end user

# 2 Technology overview

### 2.1 Frontend

The user interface will be created using the Microsoft WPF Framework. This will provide clean separation between design and code while providing a modern look at the same time.

### 2.2 Backend

The data will be stored in a SQLite database. It will be accessed using Microsoft Entity Framework, version 6 (stable). The database will be created using code-first techniques, meaning that manual SQL statement dispatching is not required (except for testing).

### 2.3 Programming languages

As given in the program request, the entire coding will happen in C# .NET version 6. For data access SQL might be used for testing purposes, in production this will be abstracted away by code-first EntityFramework 6 database access.

**2.4 Collaboration**

For team collaboration, a distributed version control system will be used. Namely, this will be github.com. The Repository-URL at the time of writing is as follows:

<https://github.com/TheRealVira/Scrooge>

The semi-distributed architecture of the program (see 4.1) will assist in developing concurrently.

Weekly (at least) team meetings can be used for developer communication. These will, funnily enough, always be happening in our programming lessons – what a coincidence indeed.

# 3 UI Overview

### 3.1 User Interface Concepts

The user interface will be designed to be as simple as possible while still providing all necessary functionality.

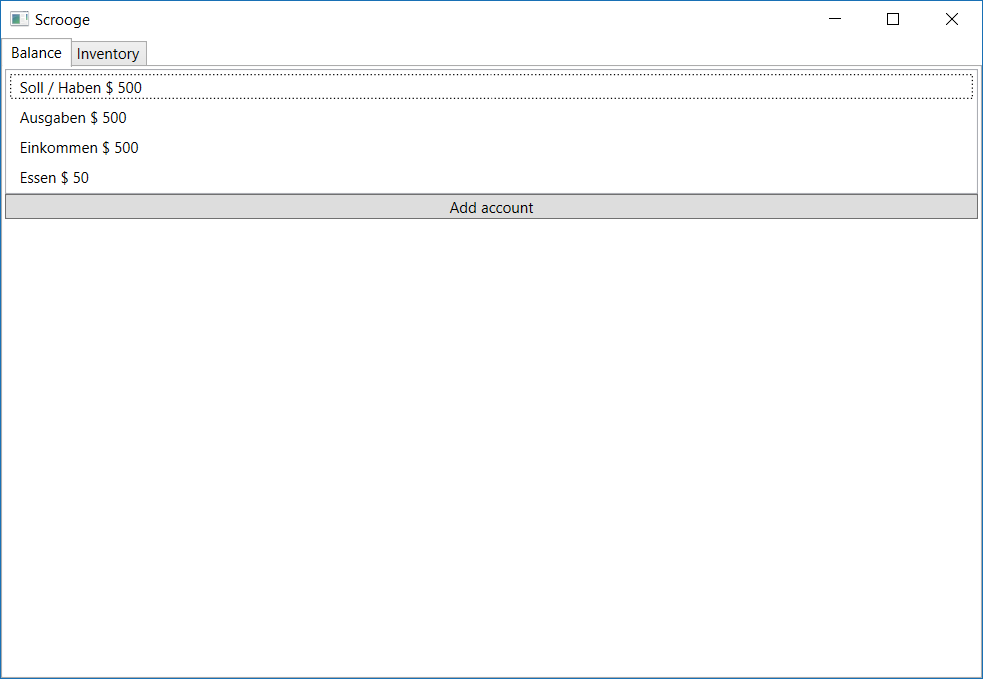
The general GUI functions will be split up into tabs, each of them designed for one specific use case. This means you have for example one tab for inventory data-entry and one for data-viewing and one for looking at income/outcome data plus one for the detailed summary of the data.

Also, a so called warning indicator will be placed at the top of the UI which will provide information about immediate actions that can or have to be taken.

The general gist of the application is to provide necessary functions in a simple way, following the “flow” will lead the user to every action they need to do.

### 3.2 UI Concept Drawings

### 



# 4. Architecture overview

### 4.1 Basic architecture discussion

Scrooge will use a pseudo-microservice architecture. The well-known microservice architecture will provide the flexibility to add modules to the program in a simple way while the “pseudo” part is important for performance reasons, meaning that services will not communicate over a message bus in a standardized format but instead communicate directly.

### 4.2 Service definition

A service in Scrooge will not be a separated process, instead it will be defined as a .NET class implementing a simple interface for basic communication. These can be loaded from the same assembly or library files located in a special location.

### 4.3 Service discovery

Service discovery will be provided by a central service controller (CSC) class. Services can register themselves on initialization and then be found using type-searching. The CSC will provide a service using one of these distribution techniques: First-only, Last-only, Round-Robin; defaulting to Last-only.

Code examples for this behavior from a services perspective would be:

ServiceController.Register(this);

...

ServiceController.GetService(typeof(IFooBarService));

### 4.4 Service communication

Services will communicate directly, disregarding the need for a performance-heavy message bus. This will happen through service interfaces, which can be received from the CSC.

### 4.5 Important services

* GUI service: Provide basic GUI functions
* Storage service: Provide an interface to the backend-database
* Report services: Generate different reports from finance data
* Calculation services: Calculate different summarizations from finance data

# 5. Outlook

### 5.1 Further project work

### The project will be considered open source from the moment the contracted work is done. From there onwards it will be distributed online and open for change requests (pull requests). The code will be licensed under the GNUv3 license.