Software Requirements Specification

for

Calculator App

Dan Richmond

Version: 4.0

Last revised: December 13th, 2017

Table of Contents

1. Introduction

Sections of the introduction will define important details about the application in general. It will include system requirements and limitations as well as the purpose of the SRS document.

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to provide a detailed description of the functionalities of the Calculator Application.

During the developmental stages, this SRS document should be referenced to ensure the application meets all of the required specifications. New team members should also reference this document to understand all of the systems specifications.

This SRS document should also be maintained and updated as needed throughout the lifetime of the application. As requirements change and new features are implemented, they should be reflected in this document.

1.2 Definitions, Acronyms, and Abbreviations

NFR (Non-Functional Requirements)
Req NF. # (Requirement Non-Functional Number)
Req F. # (Requirement Functional Number)

1.3 References

During the creation of this document, I referred to these sources. (Lisk/Acknowledge sources)

1.4 Overview

The differentiation feature of this calculator application is that it will be able to save calculations in long term memory. This prevents the users' from having to recalculate previously calculated calculations.

1.5 Delivery

To deliver the application for grading, it will be submitted to D2L as a zipped file. The latest version can always be found on github.com/danrichmond/calculator. The iOS app was developed using Xcode 9.1 and Swift 4.0. The deployment target is iOS 11. Reference section 1.6 for installation instructions.

1.6 Download Instructions

To successfully run this application, you must have a Mac with Xcode installed. It was originally developed with Xcode 9.1 and Swift 4.0. To install the application, follow one of the two methods below.

<u>Method 1 – Using Git:</u>

To download the application using this method, you must have Git installed.

Note: In the following directions, and instruction starting with ">" indicates a command that should be typed into your terminal. Simply type the text after the ">" then press enter.

Cloning the repository with Git:

1. *Open the terminal.*
2. *Go to a directory where you would like to download it to.*

3. *> git clone https://github.com/danrichmond/calculator.git*
4. *You will now see a folder named Calculator in your current working directory.*
5. *Within this folder, you can click open the Calculator.xcodeproj file. This should automatically open Xcode with the Calculator app imported.*

Method 2 – Downloading from Github

1. *Open a browser and go to https://github.com/danrichmond/calculator*
2. *Click on the green Clone or download button.*
3. *Click Download ZIP.*
4. *This will download the Calculator app to your downloads folder as calculator-master.zip.*
5. *Clicking on that file should automatically unzip it.*
6. *Within the unzipped calculator-master file, click on the Calculator.xcodeproj file. This should automatically open Xcode with the Calculator app imported.*

1.7 Running Calculator

After downloading the Calculator application as described in section 1.6, we can run the app from within Xcode. Simply click the grey arrow in the top left-hand corner of your Xcode environment. This will open Simulator and launch the Calculator app in the simulated iPhone.

2. Software Requirements

Software requirements are the descriptions of the functionalities and features that the system must provide. It also defines the limitations for which it must operate under. The requirements will

range from a high-level abstract statement to a detailed mathematical functional specification.

2.1 Types of Requirements

In section 2.2 and 2.3, we will define two types of software requirements; non-functional requirements and functional requirements. This will simply give insight to what the two different types or requirements are. A list of the actual requirements for each type can be found in sections 3 and 4.

2.2 Non-Functional Requirement Information

Non-functional requirements define the way the system should work. They describe the attributes of the system that are not required features instead they are just required attributes.

2.3 Functional Requirement Information

Functional requirements define what a system should do. They describe the behavior of the system in relation to the system's functionality. The functional requirements are the specific features which they system must implement.

3. Non-Functional Requirements

3.1 This section contains a table of the calculator's non-functional requirements.

Number	Non-Functional Requirements
Req NF. 1	The application should be developed for iOS in the Xcode IDE with the Swift programming language.
Req NF. 2	Users should be able to simply download the app then perform calculations without any issues.
Req NF. 3	The User Interface should be developed with constraints that will fit all of the different screen sizes.
Req NF. 4	The system must perform calculations in constant time to provide the quickest possible results.
Req NF. 5	The results of all calculations must be accurate. The user should be able to rely on this calculator to provide the correct results for their calculations.
Req NF. 6	Accessibility. Users should be able to download the application to their device.
Req NF. 7	The user interface must be simple and easy to understand.
Req NF. 8	The application should be supported on the latest version of iOS.

More requirements can be added to this table as needed.

4. Non-Functional Requirements

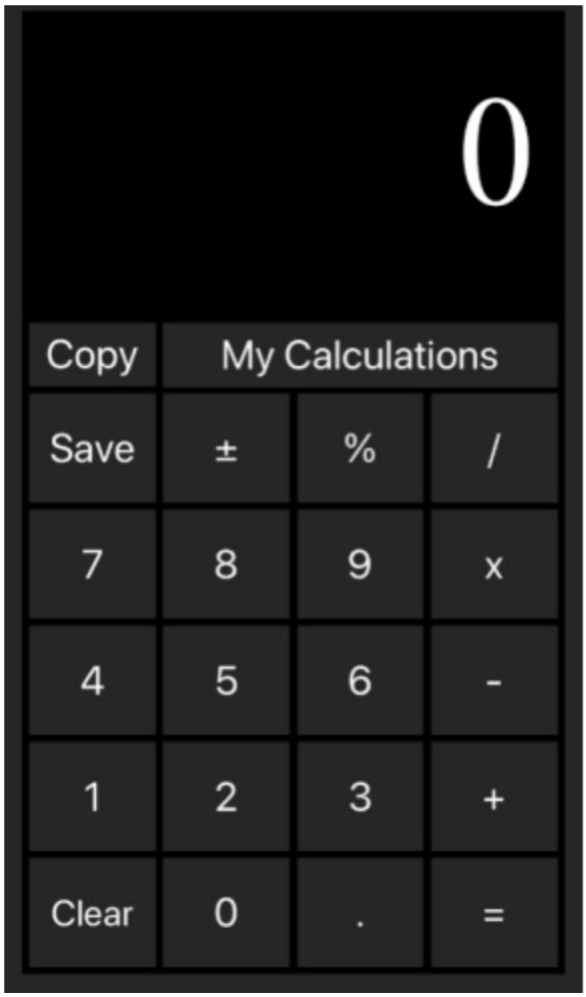
4.1 This section contains a table of the calculator's functional requirements.

Number	Functional Requirements
Req F. 1	The system should provide buttons for each operation as well as necessary calculator tasks such as clearing the current value. Refer to Image 1.0 on page 7.
Req F. 2	The number pad should be designed according to the standard calculator number pad scheme. Refer to Image 1.0 on page 7.
Req F. 3	The number label at the top of the screen should reduce the font size of the number as its length increases and fills more of the screen. Compare Image 2.0 and 2.1 on page 7.
Req F. 4	A user shall be able to search a list of saved calculations each possessing a label created by the user.
Req F. 5	The system must save all saved calculations in the devices long term memory so they can be accessed even after the application has terminated or the device has been rebooted.
Req F. 6	The system should allow the user to copy the current value to their clipboard so they can paste it elsewhere.
Req F. 7	The system should be scalable meaning the support for very large numbers and very small numbers should be available.
Req F. 8	After the user processes the equal button, a new calculation should begin upon them clicking on new numbers otherwise, the calculation should be continued.

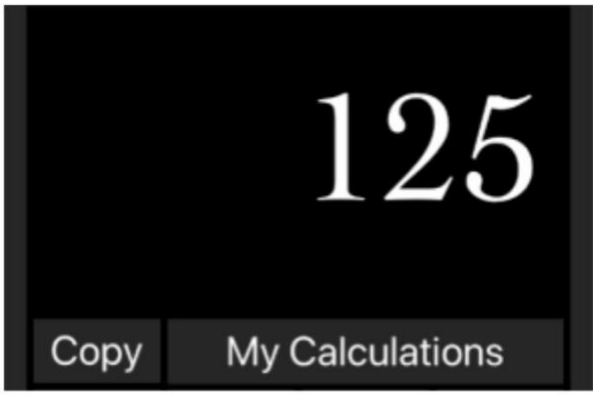
More requirements can be added to this table as needed.

4.2 This section provides pictures of the user interface which shall be referenced as needed.

(Image 4.1)



(Image 4.2)



(Image 4.3)

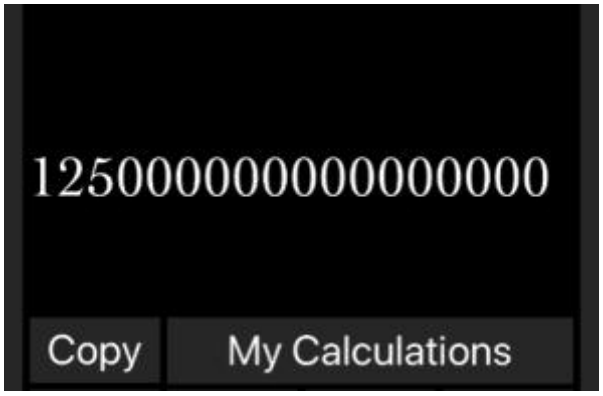


Image 4.1 shows a full-scale image of the main storyboard. Image 4.2 should be compared to image 4.3. As the number's length increases and fills the width of the label, the text size should decrease allowing more of the number to fit on the screen.

(Image 4.4)

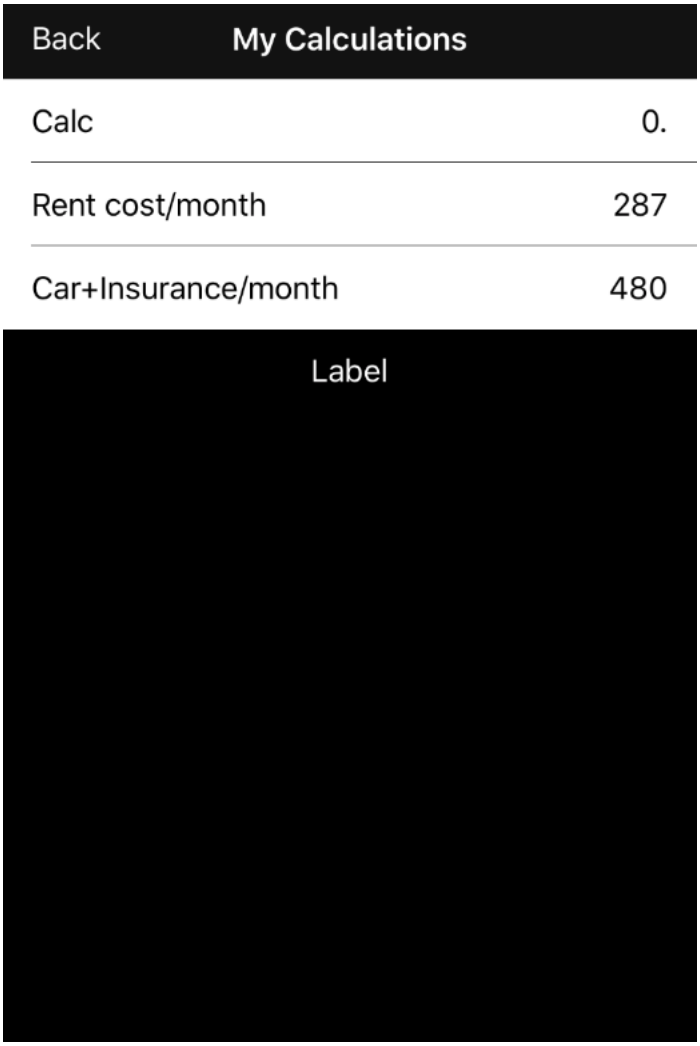


Image 4.4 should the saved calculations view. There are a few sample calculations currently in the view. A new list element should be created each time the user clicks the Save button as seen in image 4.1. The Back button in the top left corner of this view directs the user back to the main storyboard view so they can compute another calculation.

5. Improvements

5.1 Bugs to be Fixed

1. There is currently an issue with the decimal button. If it is pressed more than once on the same number, it will add multiple decimals to that number then the app will crash. There needs to be a check for this in the block that adds the decimal to the current number.

2. The application cannot currently persist the user's saved calculations in long term memory. A couple different approaches were used to try to get this to work but they resulted in several other issues. We need to use NSUserDefaults to store the array which contains the saved calculations.