Software Design Description Document

Version 2.0
March 2, 2018

Stock Screener

Kathleen Connell, Art Fogiel, & Michael Harrison

University of Maryland University College
CMSC 495 - 6382

Table of Contents

1.0. Introduction

1.1. Purpose

The purpose of this Software Design Document is to provide thorough documentation for the stock screener design. This document will be used to aid in the software development process by detailing how the stock screener will be built. Within this document, we will provide the full design layout, including its deployment, architecture, data structure, use cases, user interface, and help system design. This Software Design Document highlights the information necessary to describe the software plan for our application.

1.2. Scope

The scope of this document is to show how our stock screener application will be designed and how it will function.

1.3. Glossary

A list of the principal abbreviations and acronyms used within this document is provided below.

Abbreviation	**Definition**
MVVM	Model View View-Model
UI	User Interface
UML	Unified Modeling Language
WPF	Windows Presentation Foundation – graphical subsystem by Microsoft
XML	eXtensible Markup Language

1.4. References

Connell, K., Fogiel, A., Harrison, M. (2018) Software Requirements Specification for Stock Screener.

1.5. Overview of Document

The Software Design Document is divided in 7 sections, each with their respective subsections. The sections of this document are:

1 Introduction
2 Deployment Diagram
3 Architectural Design
4 Data Structure Design

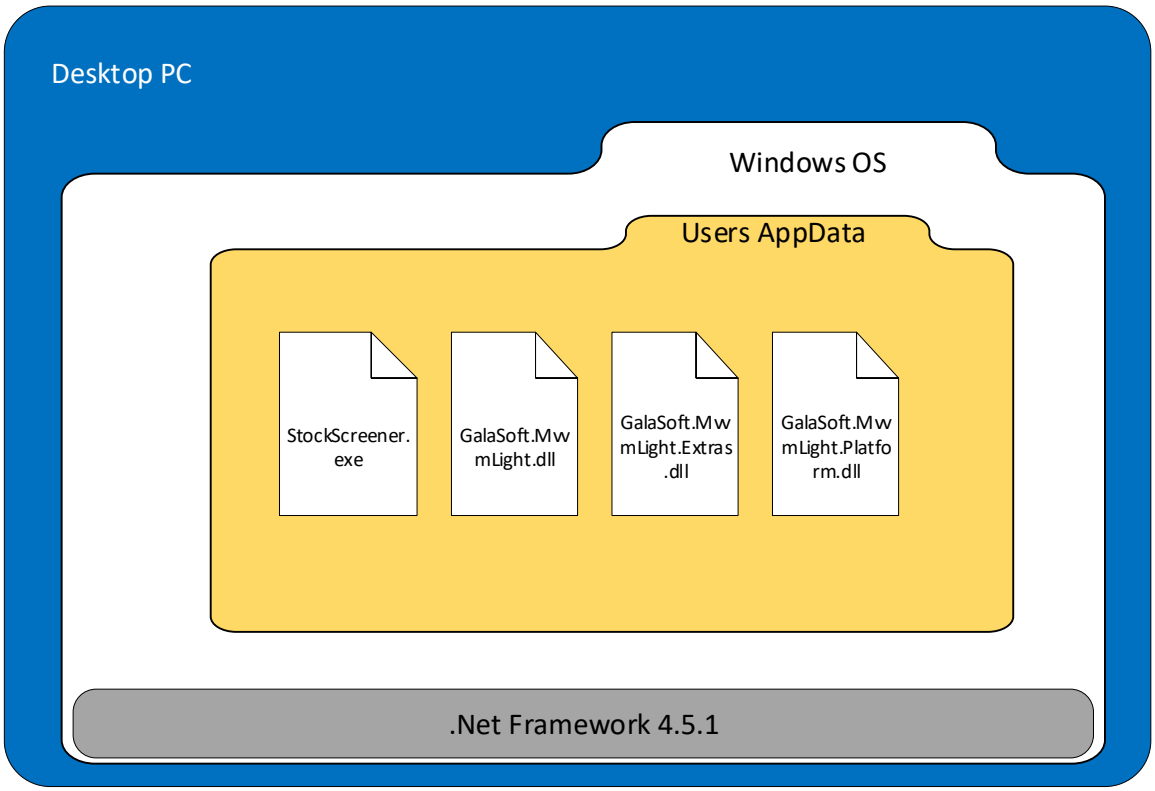
5 Use Case Realizations
6 User Interface Design
7 Help System Design

2.0. Deployment Diagram

The deployment is completed on a windows desktop using Microsoft's ClickOnce deployment system as described in:

https://msdn.microsoft.com/en-us/library/ff699204.aspx

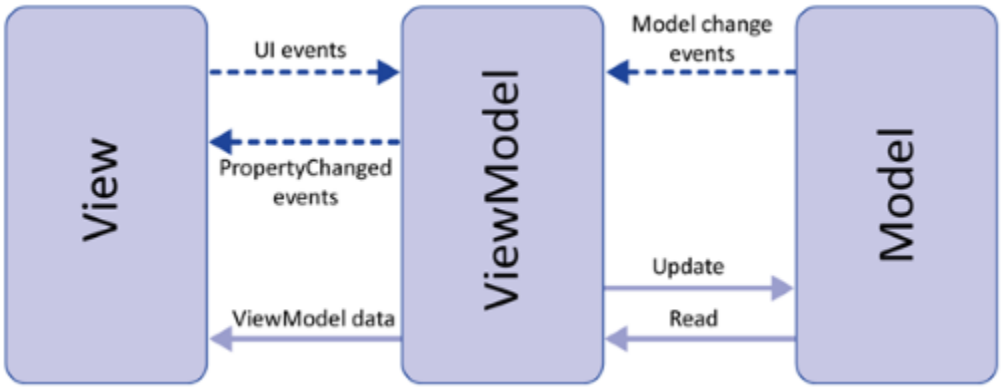
A single self-executing zip file called StockScreenerSetup.exe will be provided to the user to run. Upon running it will install the program inside the users AppData folder per ClickOnce's deployment system. The user can then start via the Start menu and uninstall if desired via the control panel. The Files installed on the computer will look as so:



3.0. Architectural Design

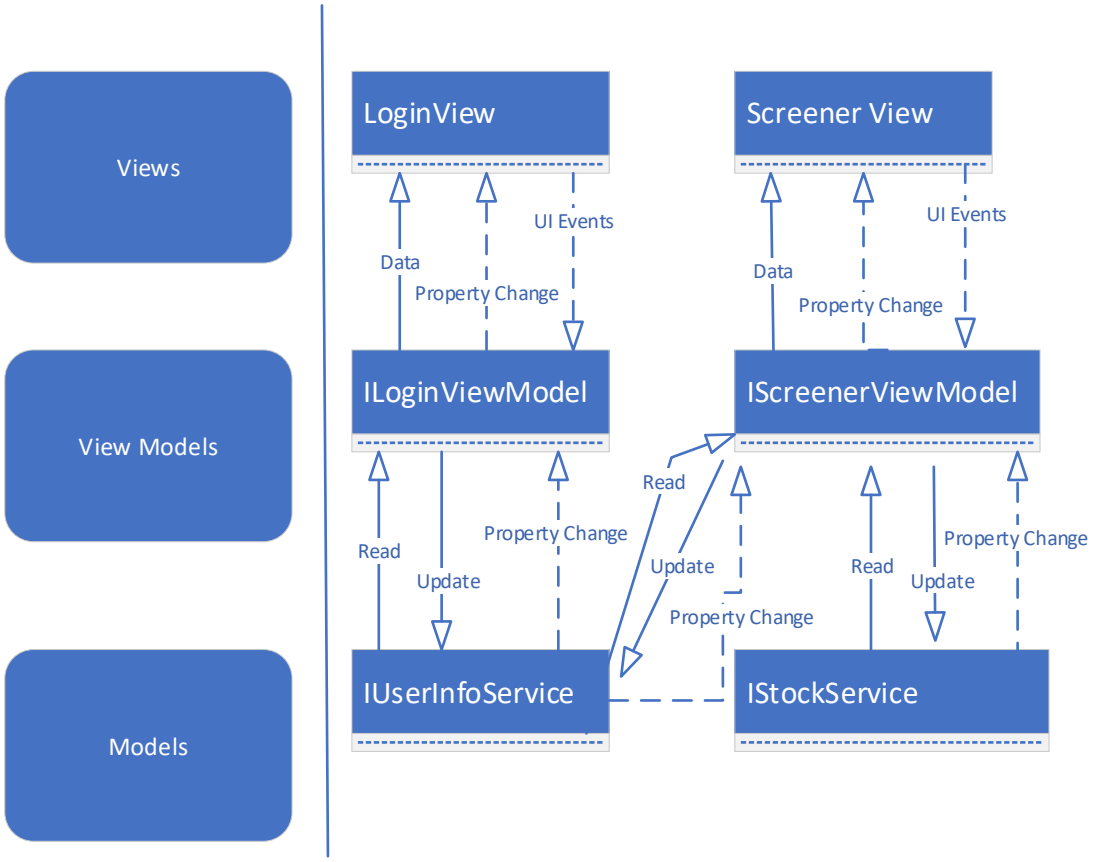
The application is a windows desktop application using C# and WPF technologies. The main design pattern is a Model View View-Model design which uses an Observable design pattern to communicate to the UI. It also leverages the Mvvm Light package to allow for an already created template for passing data between the layers. Documentation on Mvvm Light along with explanation of MVVM patterns can be found here: http://www.mvvmlight.net/doc/

Below is a common diagram of the MVVM design pattern.



This creates three distinct layers. The View is a lightweight layer that communicates to a view model via an interface abstraction. This allows multiple implementations of view models to exist as long as they implement the interface the view is expecting. The View Model Layer communicates between the view and the model layers. It implements the interface the view is expecting and interacts with the Model. Its main purpose is to convert button clicks, textboxes, etc. from a View into commands to talk to the Model. The Model has no concept of a UI and can live by itself and be re-used generically for other applications. Any one of the three layers can have its implementation changed and

the code will not break if the interfaces are followed. Our interfaces will be explained in more details in section 4 but from a high level the data communication appears as so:



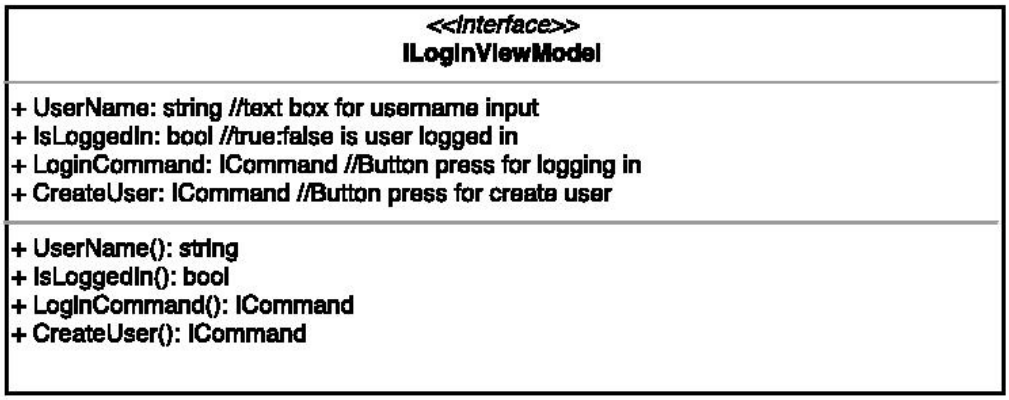
The Stock service implementation will now use IEXTrading's API calls over http to retrieve individual stock data. https://iextrading.com/ The Stock Service implements the IStockService interface, so if this method of retrieving stocks needs to be changed, a different implementation can be created implementing the IStockService and all other view and view models will still work fine.

Two different files will be created, written to, and read from. The Users.xml will contain a list of known users and their settings in an xml format. The class implementing the IUser interface seen in section 4 will be able to serialize and deserializing from the file. It will be stored on the user's PC at the environment variables path for

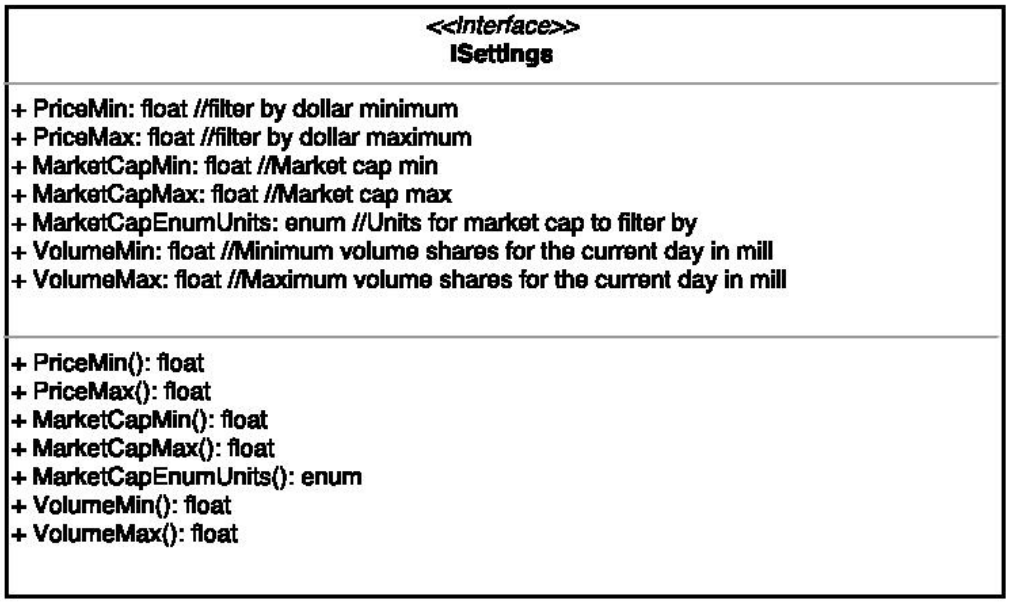
PROGRAMDATA. This way it is accessible to any user logged into the Windows Operating System. This file will contain all the known user names and their last used settings. The second file is still to be determined. Potentially we will need to store the last known stock values for all the known stocks. This will be either xml, or binary depending on the found performance of the 6000 stocks over xml. At the time we are unsure if this file will be needed or not. It is still to be determined the time it takes to gather all the stock information initially from IEXTrading using multiple threads. If the time is under 10 seconds, then we will initialize on each launching of the app. If it is more, we may store the last values to a file that is read on launch and have the newer values trickle in as available. See section 5 for the use cases and the sequential interaction between the implementations.

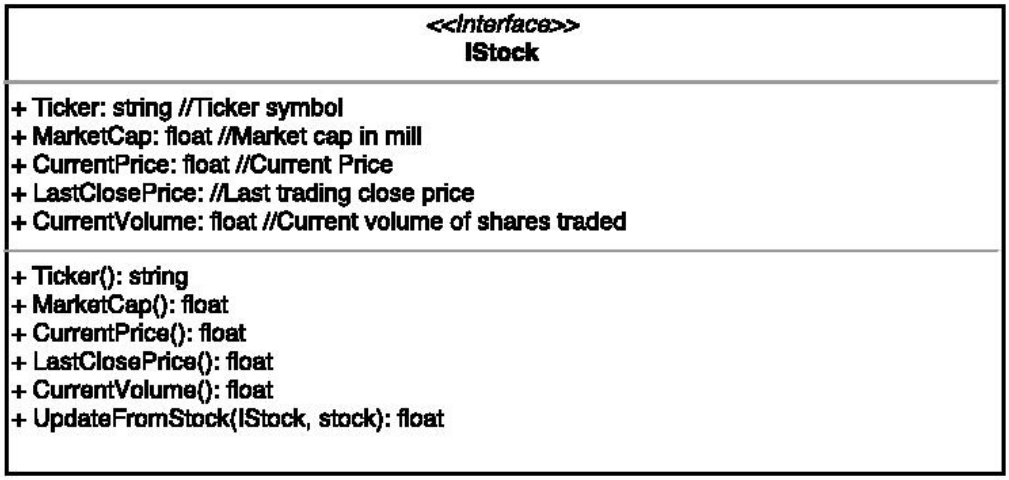
4.0. Data Structure Design

The following diagram represents the UML diagrams for each interface of the stock screener application.


<<Interface>>
ILoginViewModel
+ UserName: string //text box for username input
+ IsLoggedIn: bool //true:false is user logged in
+ LoginCommand: ICommand //Button press for logging in
+ CreateUser: ICommand //Button press for create user
+ UserName(): string
+ IsLoggedIn(): bool
+ LogInCommand(): ICommand
+ CreateUser(): ICommand


<<Enumeration>>
MarketCapUnitsEnum
+ Millions,
+ Billions


<<Interface>>
ISettings
+ PriceMin: float //filter by dollar minimum
+ PriceMax: float //filter by dollar maximum
+ MarketCapMin: float //Market cap min
+ MarketCapMax: float //Market cap max
+ MarketCapEnumUnits: enum //Units for market cap to filter by
+ VolumeMin: float //Minimum volume shares for the current day in mill
+ VolumeMax: float //Maximum volume shares for the current day in mill
+ PriceMin(): float
+ PriceMax(): float
+ MarketCapMin(): float
+ MarketCapMax(): float
+ MarketCapEnumUnits(): enum
+ VolumeMin(): float
+ VolumeMax(): float


<<Interface>>
IStock
+ Ticker: string //Ticker symbol
+ MarketCap: float //Market cap in mill
+ CurrentPrice: float //Current Price
+ LastClosePrice: //Last trading close price
+ CurrentVolume: float //Current volume of shares traded
+ Ticker(): string
+ MarketCap(): float
+ CurrentPrice(): float
+ LastClosePrice(): float
+ CurrentVolume(): float
+ UpdateFromStock(IStock, stock): float


<<Interface>

+ ObservableCollection<IStock>: FilteredStocks()
+ ObservableCollection<IStock>: WatchedStocks()
+ ICommand: SaveFavorite()
+ ICommand: Apply()

<<Interface>>
IStockService: INotifyPropertyChanged

+ List<IStock>: Stocks //List of all stocks that are constantly updated

+ List<IStock>: Stocks()

<<Interface>>
IUser

+ Name: string //Name of the user
+ List<string>: WatchedStocks //List of stocks to watch
+ ISettings: Settings //User settings, updated on apply

+ Name(): string
+ List<string>: WatchedStocks()
+ ISettings: Settings()

<<Interface>>
IUserInfoService

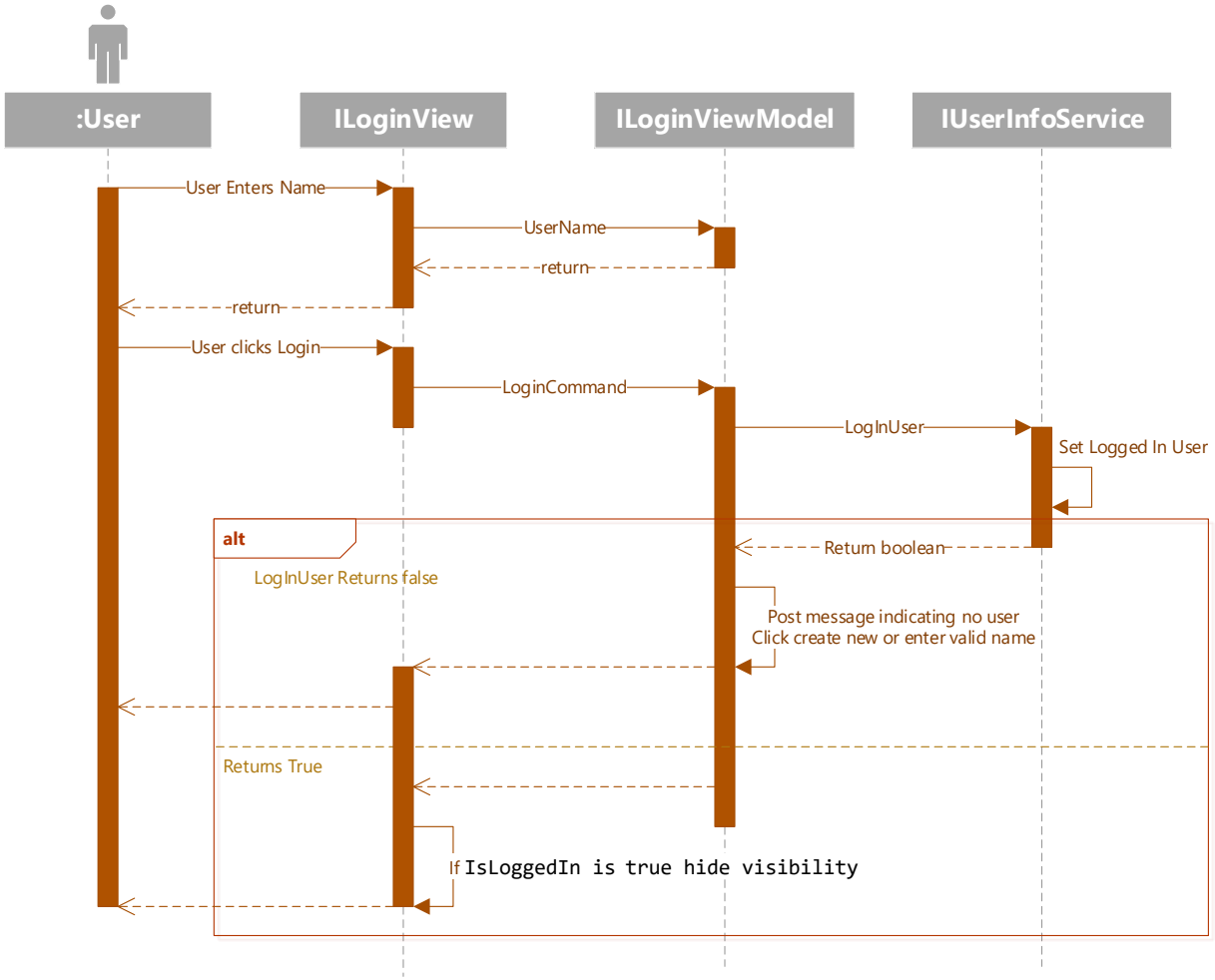
+ IUser: LoggedInUser
+ List<IUser>: Users

+ IUser: LoggedInUser()
+ LogInUser: bool(string user): name= user
+ CreateUser: bool(string user): name= user
+ LogOffUser(void)
+ List<IUser>: Users()
+ LoadUsersFromFile: bool(string filePath): name= filePath
+ SaveUsersToFile: bool(string filePath): name= filePath

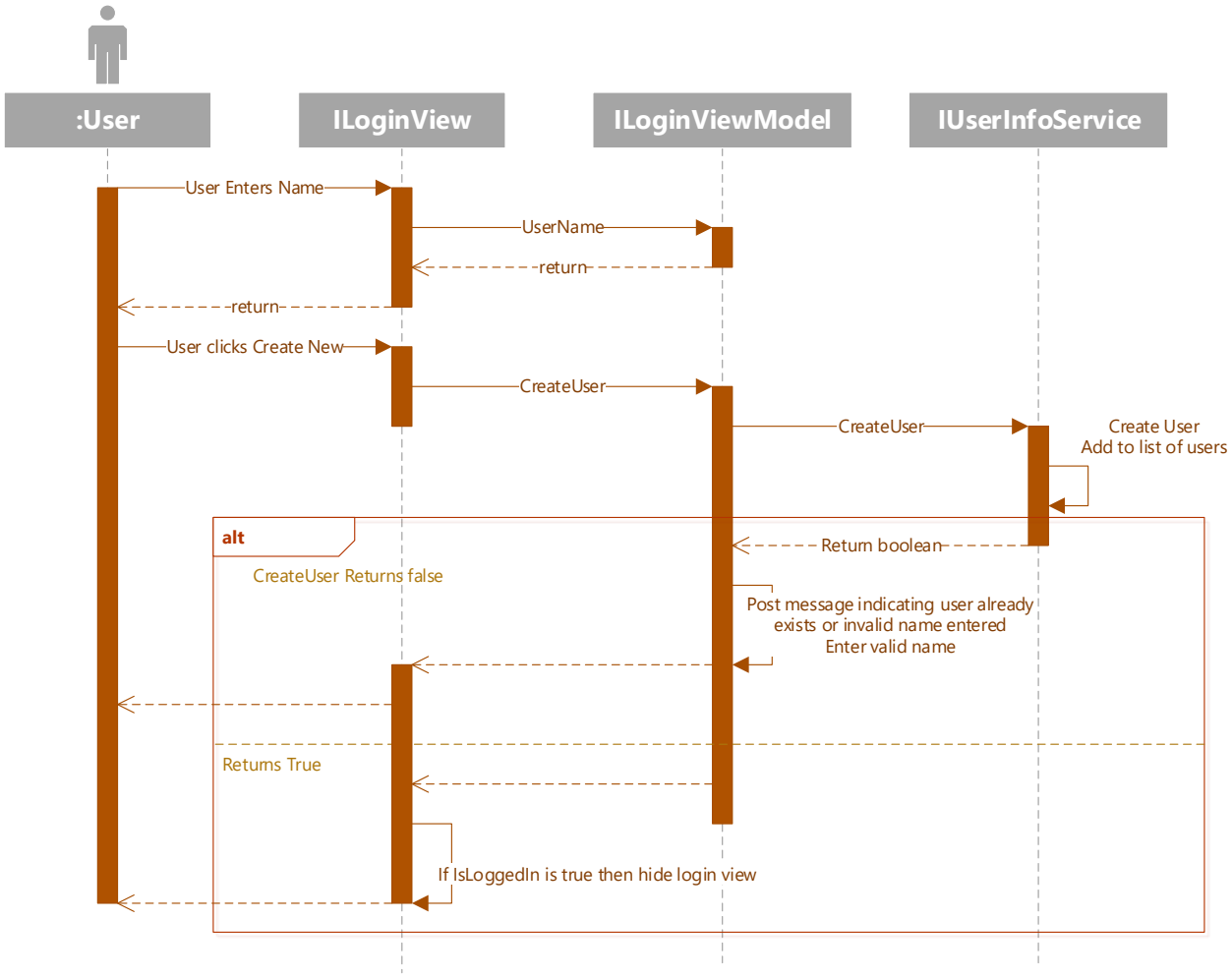
5.0 Use Case Realizations

Use case from SRS document section 4.1:

Login Button Case:

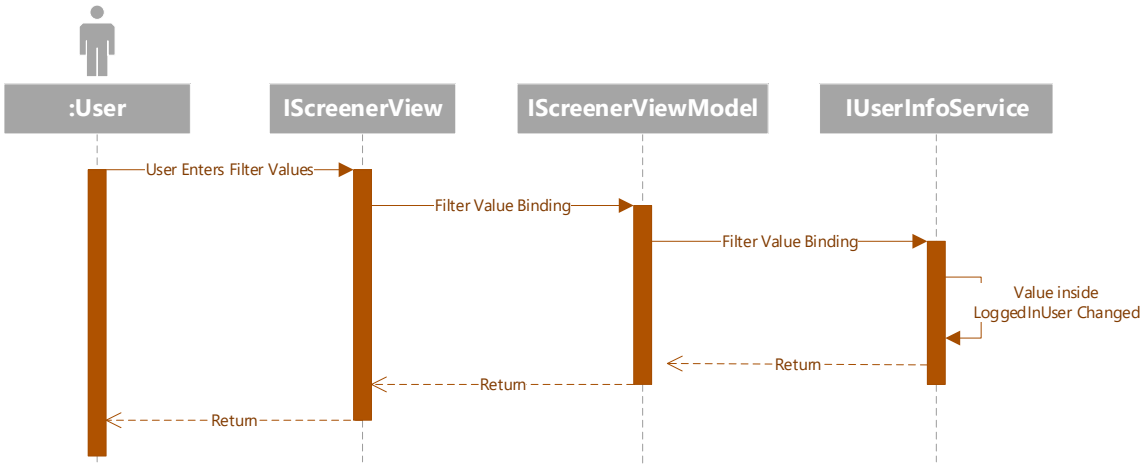


Create User Button Case:

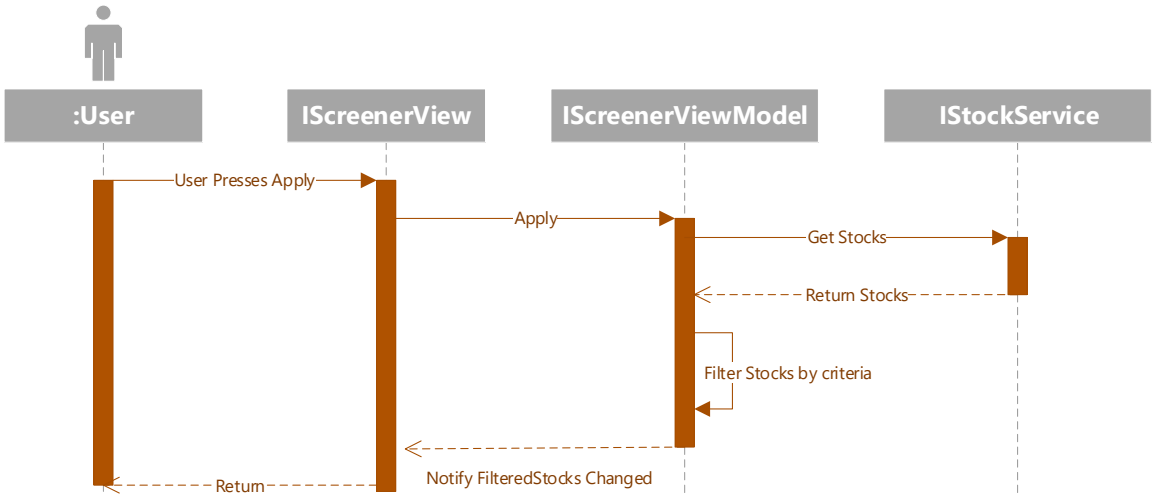


Use cases from SRS document section 4.2:

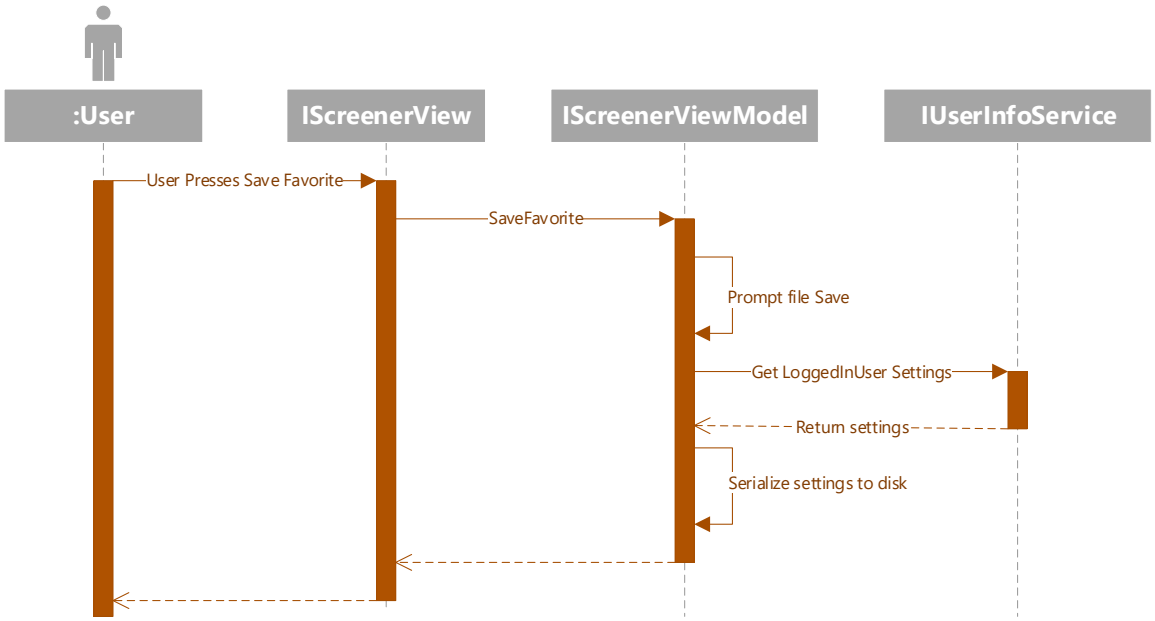
Filter Settings value change use case:



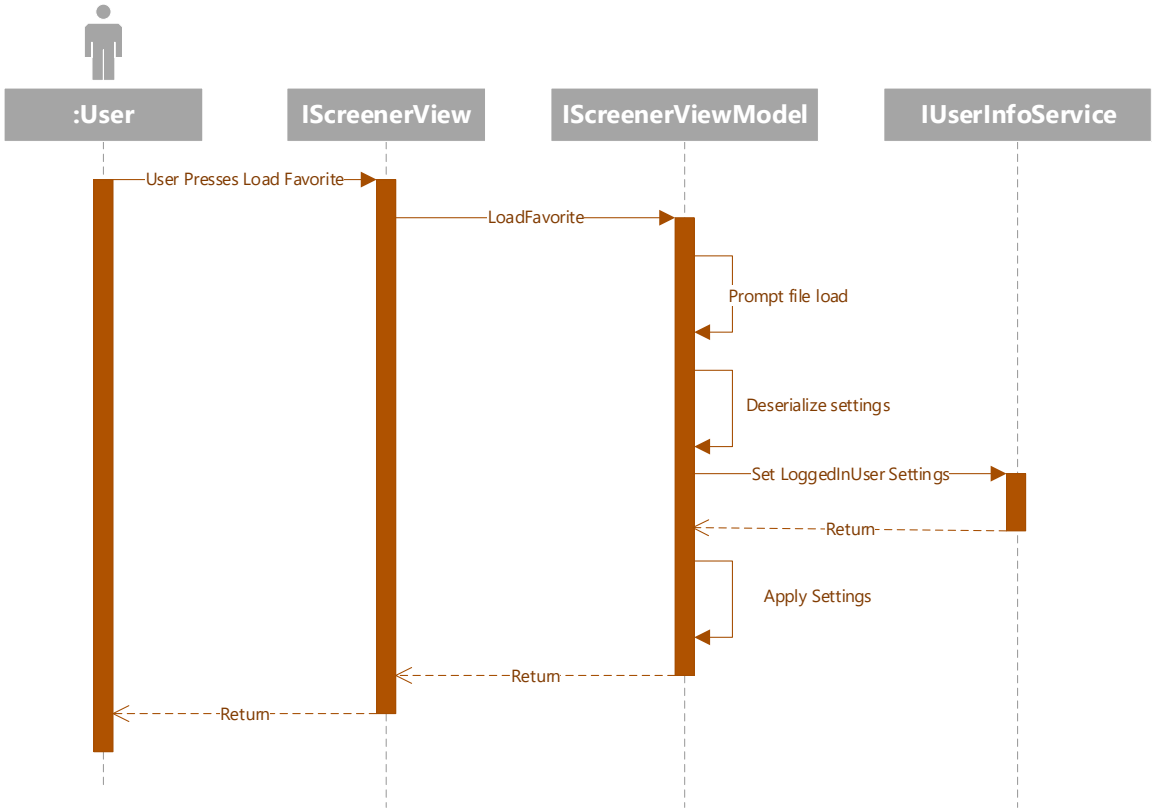
Filter Settings Apply use case:



Filter Settings Save Favorite use case:



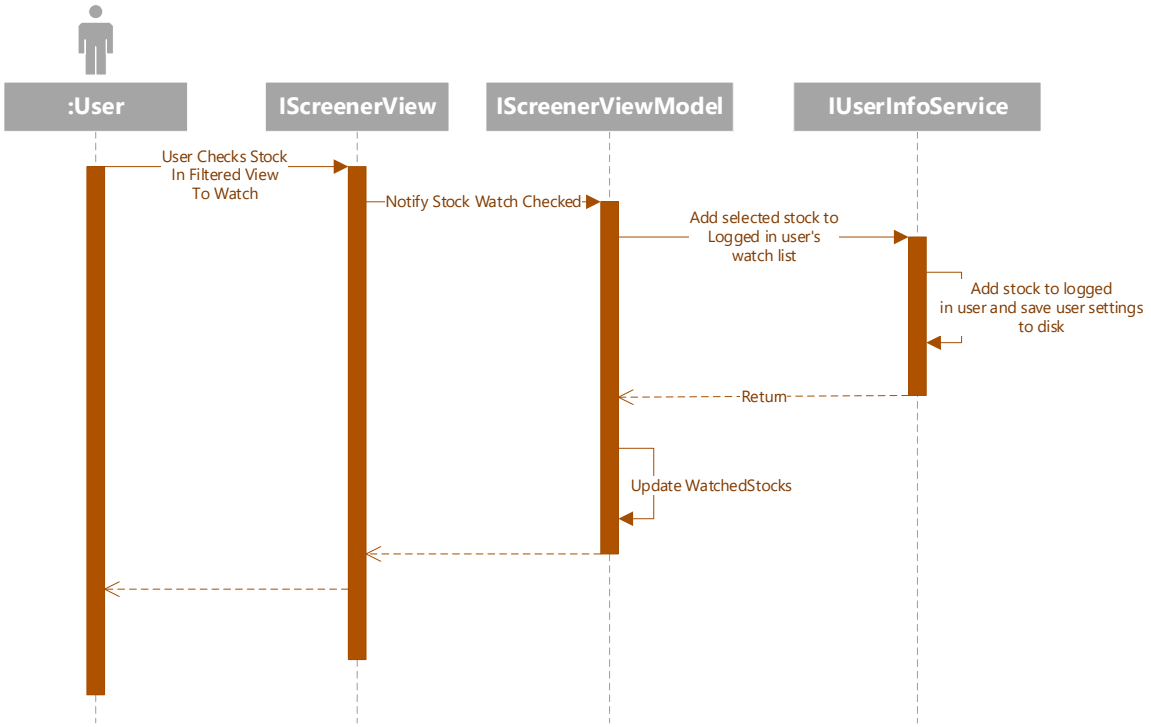
Filter Settings Load Favorite use case:



Use case from SRS document section 4.3:

For center view see Filter Settings Apply use case 4.2 above.

Watched stocks use case:



6.0 User Interface Design

This section presents the user interface design of the stock screener application. The user interface consists of the login and home page of the stock screener, along with the filters and stock data for a user to interact with. Referencing our SRS Document, the user interface of the stock screener is designed for users of all experience-levels to easily filter stocks based on user-defined metrics.

6.1 Description of the User Interface

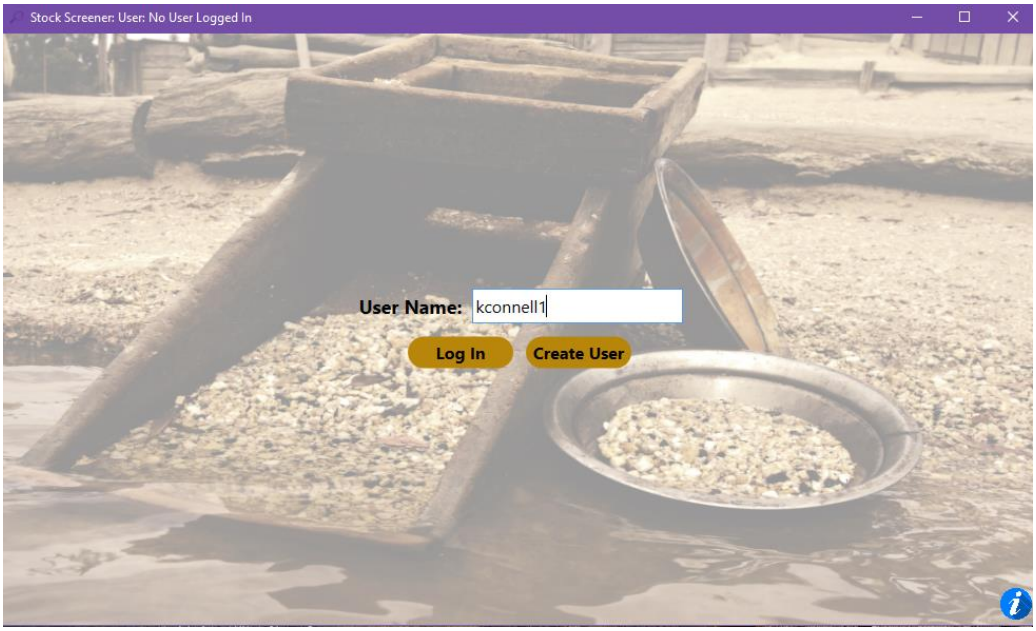
Each display consists of various GUI components, including select buttons, labels, text fields, and listed stock data. These components are arranged in a way that allows a user to easily understand the function of each display and how to easily interact with the UI.

6.2 Login Screen

Once the user has installed the application and opened it, the login/create user screen will appear.

The Login page has the following features:

- User Name text box
- Log In button
- Create User button
- User Guide button – “i” icon



As shown in the figure above, a User Name text box will appear on the screen, which allows the user to choose to Login with a current User Name, or Create User with a new User Name. User Guide is accessible via the information icon in the bottom right corner of the login screen.

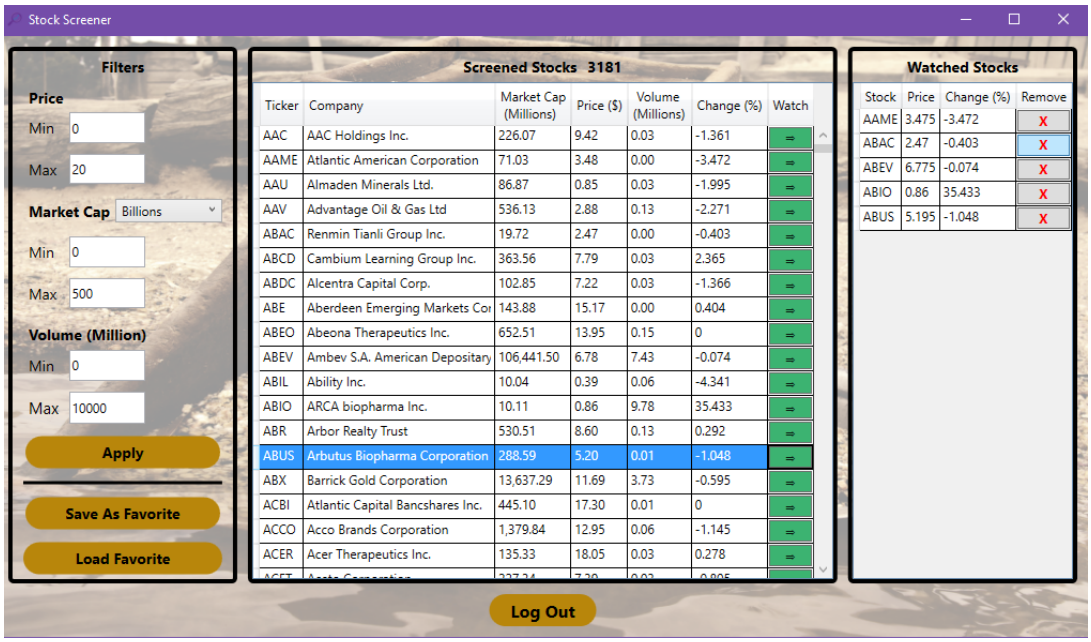
6.3 Stock Screener Home Page Screen

Once the user has logged on, the stock screener home page will appear. The home screen displays the stock screener interface, which includes 3 sections: Filters, Screened Stocks, & Watching.

The user Home Page has the following features:

- Filters
    - Price Min/Max

    - Market Cap Min/Max
    - Volume Min/Max
- Apply filters button
- Save As Favorite button
- Load Favorite button
- Screened Stocks data
    - Stock ticker
    - Company name
    - Last Price
    - Volume
    - Market Cap
    - Change %
    - Watch (green double arrow)
- Watched data
    - Stock ticker
    - Remove button
- Logout button



As shown in the figure above, the home screen will display the stock screener, showing the stocks and various filter/watch functions that it performs. Watched stocks always appear in the left pain regardless of the current filter settings.

The far left section of the home display will hold the filter metrics. Each individual filter metric has a respective text box (numerical values only), where the user can apply their desired metrics. There is an ‘Apply’ button that the user clicks to apply their entered stock filters. The UI also will have a ‘Save As Favorite’ and ‘Load As Favorite’ buttons, which will save filtered settings that they can access upon Login.

The central section holds the columns that display filtered stocks and their respective data. User will be able to see the stocks and their data that meet their filter requirements, as well as click a green arrow under the ‘Watch’ column, that moves the stock to the Watched Stocks pane.

The far right section of the home display holds the Watched Stocks. User will be able to see the names of the stocks they’re watching, and can click an ‘X’ button to remove stocks from this column.

The Log Out button will be located at the bottom of the screen that when clicked, will log the user out of the application. User is taken back to the home page.

For more information on the User Interface, please refer to Section 4 of the SRS Document.

7.0 Help System Design

The stock screener application’s help system design is the User Guide. The User Guide will be accessible via an info icon the user can click at the bottom right corner of the login screen. When clicked, the User Guide will pop up as a PDF document.