Homework 9
Stock Trading App

Prof. Marco Papa

Developed and Designed by
Zezhen Xu (iOS) & Rushabh Kapadia (Android)

Table of Contents

1. Objectives

- Become familiar with Swift language, Xcode and IOS App development.
- Learn the latest SwiftUI framework.
- Practice the Model-View-ViewModel (MVVM) design pattern.
- Build a beautiful native iOS app, and dive deep into native functionalities provided by Apple.
- Learn to use the tiingo APIs
- Manage and use third-party libraries through Swift Package Manager.

The objective is to create an iOS application as specified in the document below and in the video.

2. Background

2.1 Xcode

Xcode is an integrated development environment (IDE) containing a suite of software development tools developed by Apple for developing software for OS X, iOS, iPadOS, watchOS and tvOS. First released in 2003, the latest stable release is XCode 12. It is available via the Mac App Store free of charge for OS X 10.15 Catalina and 11 Big Sur users.

Features including:

- Swift 5 support
- Playgrounds
- SwiftUI2 Support
- Live Preview
- Device simulator and testing
- User Interface Testing
- Code Coverage

The Official homepage of the Xcode is located at:

https://developer.apple.com/xcode/

2.2 iOS

iOS (originally iPhone OS) is a mobile operating system created and developed by Apple Inc. and distributed exclusively for Apple hardware. It is the operating system that presently powers many of the company's mobile devices, including the iPhone, iPad, and iPod touch. It is the second most popular mobile operating system in the world by sales, after Android.

The Official iOS home page is located at:

http://www.apple.com/ios/

The Official iOS Developer homepage is located at:

https://developer.apple.com/ios/

2.3 Swift

Swift is a general-purpose, multi-paradigm, compiled programming language created for iOS, OS X, watchOS, tvOS and Linux development by Apple Inc. Swift is designed to work with Apple's Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products. Swift is intended to be more resilient to erroneous code ("safer") than Objective-C and also more concise. It is built with the LLVM compiler framework included in Xcode 6 and later and uses the Objective-C runtime, which allows C, Objective-C, C++ and Swift code to run within a single program.

The Official Swift homepage is located at:

https://developer.apple.com/swift/

2.4 Swift UI

SwiftUI is an innovative, exceptionally simple way to build user interfaces across all Apple platforms with the power of Swift. Build user interfaces for any Apple device using just one set of tools and APIs. With a declarative Swift syntax that's easy to read and natural to write, SwiftUI works seamlessly with new Xcode design tools to keep your code and design perfectly in sync. Automatic support for Dynamic Type, Dark Mode, localization, and accessibility means your first line of SwiftUI code is already the most powerful UI code you've ever written.

SwiftUI was initially released on WWDC in 2019, the latest version is SwiftUI 2 released earlier this June at WWDC 2020. With SwiftUI 2, developers are now able to build complete apps with swift language only for both the UI and the logic. It requires macOS 10.15.4 or later, preferably macOS 11 Big Sur.

The Official SwiftUI homepage is located at:

https://developer.apple.com/xcode/swiftui/

2.5 SF Symbols

With over 2,400 configurable symbols, SF Symbols is designed to integrate seamlessly with San Francisco, the system font for Apple platforms. Each symbol comes in a wide range of weights and scales that automatically align with text labels, and supports Dynamic Type and the Bold Text accessibility feature. You can also export symbols and edit them in vector graphics editing tools to create custom symbols with shared design characteristics and accessibility features.

The latest version, SF Symbols 2, was released earlier this June. SF Symbols 2 features over 750 new symbols, including devices, health, transportation symbols, and more. These new symbols are available in apps running the beta versions of iOS 14, iPadOS 14, or macOS Big Sur. It requires macOS 10.15.3 or later, preferably macOS 11 Big Sur.

The Official SF Symbols homepage is located at:

https://developer.apple.com/sf-symbols/

2.6 Amazon Web Services (AWS)

AWS is Amazon's implementation of cloud computing. Included in AWS is Amazon Elastic Compute Cloud (EC2), which delivers scalable, pay-as-you-go compute capacity in the cloud, and AWS Elastic Beanstalk, an even easier way to quickly deploy and manage applications in the AWS cloud. You simply upload your application, and Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring. Elastic Beanstalk is built using familiar software stacks such as the Apache HTTP Server, PHP, and Python, Passenger for Ruby, IIS for .NET, and Apache Tomcat for Java.

The Amazon Web Services homepage is available at: http://aws.amazon.com/

2.7 Google App Engine (GAE)

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go. App Engine applications automatically scale based on incoming traffic. Load balancing, micro services, authorization, SQL and noSQL databases, memcache, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable.

To learn more about GAE support for PHP visit this page:

https://cloud.google.com/appengine/docs/php/

To learn more about GAE support for Node.js visit this page:

https://cloud.google.com/appengine/docs/flexible/Node.js/

2.7 Microsoft Azure

The Azure cloud platform is more than 200 products and cloud services designed to help you bring new solutions to life—to solve today's challenges and create the future. Build, run, and manage applications across multiple clouds, on-premises, and at the edge, with the tools and frameworks of your choice. To learn more about the Azure services, visit this page:

https://azure.microsoft.com/en-us/solutions/

To learn more about Azure support for Node.js visit this page:

https://docs.microsoft.com/en-us/azure/devops/pipelines/targets/webapp?view=azure-devops&tabs=yaml#deploy-a-javascript-nodejs-app

3. Prerequisites

IMPORTANT
This homework is developed on the latest MacOS Big Sur (11). By the time of this document was written, Big Sur is still in beta. You can install the public beta version by following this guide.
You can develop the iOS app with MacOS Catalina later than 10.15.4. **MacOS versions earlier than 10.15.4 will not be able to finish this homework. The latest and preferred MacOS version is Catalina 10.15.7.**
If you meet problems such as missing symbols (system images) using SFSymbols2, fail to compile or run SwiftUI2 code, your best option is to install MacOS Big Sur.

IMPORTANT
This homework is developed with SwiftUI2, **not** storyboard. We strongly suggest you to develop the app with SwiftUI2. If you use storyboards, you might find some of the functionalities harder to implement, and we do not provide support in such cases.

This homework requires the use of the following components:

- **Download and install Xcode 12**. Xcode 12 provides the crucial functionalities you will need to develop SwiftUI2 apps. You can download Xcode 12 by searching "Xcode" in your mac's app store.
- **Download and install** SF Symbols 2, this app provides all of the necessary icons for this homework. **SF Symbols 1 might not contain all required icons.** If you see a symbol fail to load during development, it might be because your MacOS version is too old.
- If you are new to iOS/SwiftUI2 Development, Hints are going to be your best friends!

4. High Level Design

This homework is an extended mobile app version of Homework 6 and Homework 8.
In this exercise, you will develop a native SwiftUI application, which allows users to search for different stock symbols/tickers and look at the detailed information about them. Additionally, the users can trade with virtual money and create a portfolio. Users can also favorite stock symbols to track their stock prices. The App contains 2 screens: Home screen and the Detailed Stock Information screen. However, the App has multiple features on each of these screens.

This homework contains 5 API calls. There are 4 calls to the tiingo APIs for company description, stock prices, autocomplete and chart data points, and the additional newsapi endpoint. Each of these 5 API calls are the same as Homework #8. So, you can use the same Node.js backend as HW8 with an additional call added as a new endpoint. In case you need to change something in Node, make sure you do not break your Angular assignment (or deploy a separate copy) as the grading for homework will not be finished at least until 1 week later.

We strongly suggest you develop with SwiftUI2, not storyboard.

PS: This app has been designed and implemented in a iPhone11/Pro simulator. It is highly recommended that you use the same simulator to ensure consistency.

Demo will be on a simulator, using Zoom remote control, no personal devices allowed, see the rules:

https://csci571.com/courseinfo.html#homeworks

5. Implementation

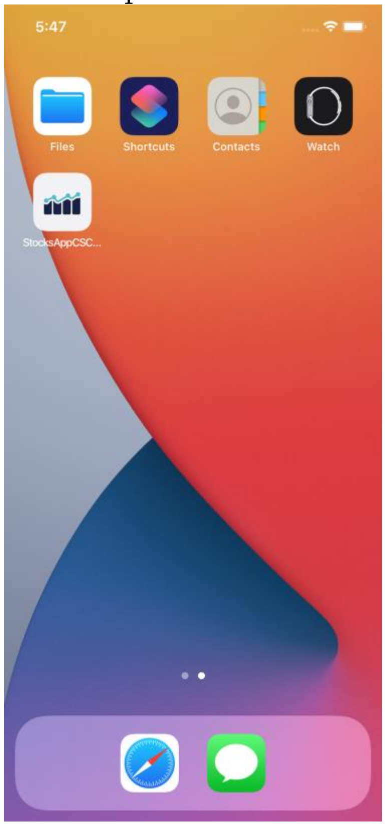
5.1 App Icon and Splash Screen

In order to get the app icon/image assets, please see the **hints** section.

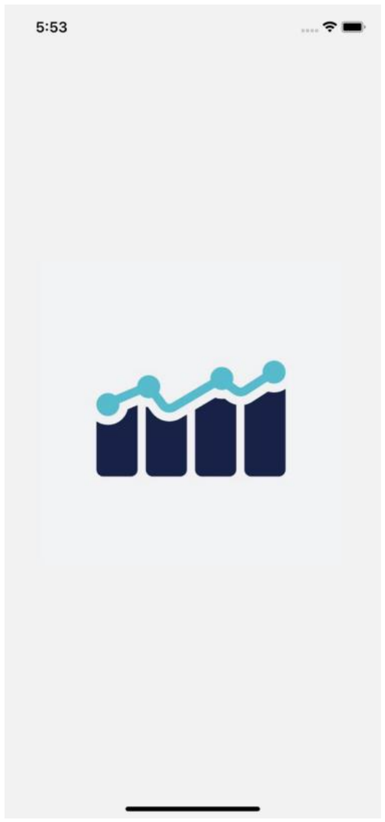
The app begins with a welcome screen which displays the icon provided in the hint above.
This screen is called Splash Screen and can be implemented using two methods: plist or storyboards.

See this link for more info: Launch screens in Xcode: All the options explained

The image is in the assets provided in the hints.



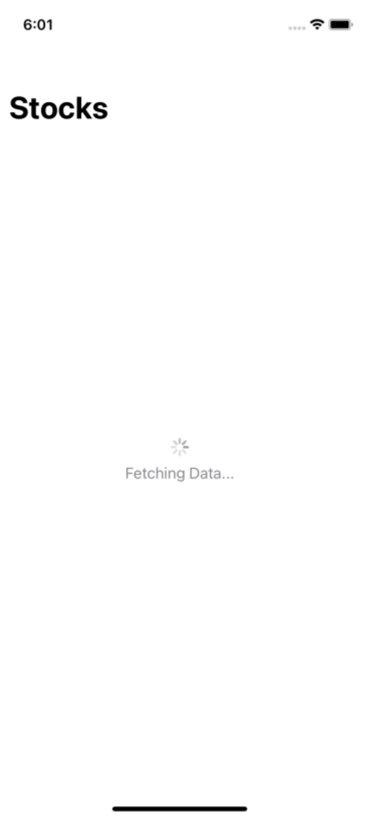
App Icon



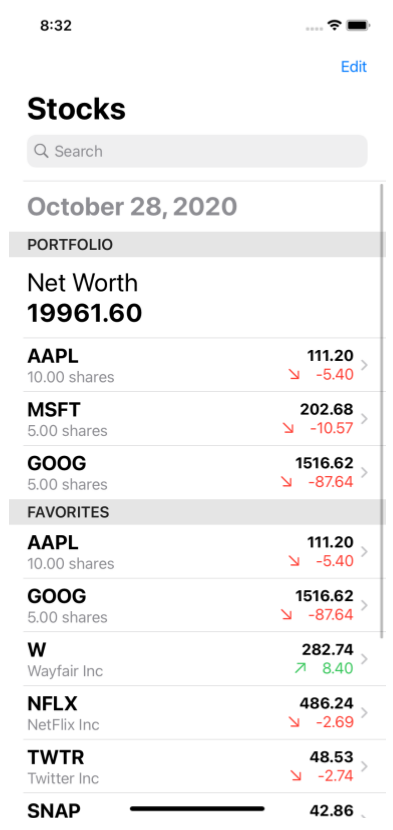
Splash Screen

5.2 Home screen

When you open the app, there will be an initial spinner while the data is being fetched using ProgressView as shown below. The home screen will have a Navigation Bar toolbar at the top with Stocks title. A search bar is attached to the navigation bar and will be revealed if you scroll up.



Fetching Data



6:32

Edit

Stocks

Search

October 28, 2020

PORTFOLIO

Net Worth
19961.60

AAPL 10.00 shares	111.20 ↘ -5.40
MSFT 5.00 shares	202.68 ↘ -10.57
GOOG 5.00 shares	1516.62 ↘ -87.64

FAVORITES

AAPL 10.00 shares	111.20 ↘ -5.40
GOOG 5.00 shares	1516.62 ↘ -87.64
W Wayfair Inc	282.74 ↗ 8.40
NFLX NetFlix Inc	486.24 ↘ -2.69
TWTR Twitter Inc	48.53 ↘ -2.74
SNAP	42.86

Home Screen

Under the Navigation Bar, a single ListView is used. The first row will show the current date as shown.

In the same ListView, there are 2 Sections on the home screen:

- ***Portfolio Section*** - This section will show the total net worth of the user, which is calculated as the sum of number of shares of a stock multiplied by the current price, plus uninvested cash. This is followed by the list of stocks in the user portfolio with their current price, change in price and total shares owned information.
  If the user starts the app for the first time, they will not have any stocks/shares in the portfolio and an initial pre-loaded amount of **$20,000** to trade on the app. This amount can change based on the trading done by the user. (For example, if the market fluctuates after buying stocks, it can become more, or less than 20,000)
- ***Favorites Section*** - This section will show all the stocks that have been favorited by the user to allow the user to easily check the prices of stocks in their watchlist. The stock symbol, current price, change in price and company name should be displayed as shown. In case the favorited stock is present in the user portfolio, instead of the company name, the stocks owned should be displayed.

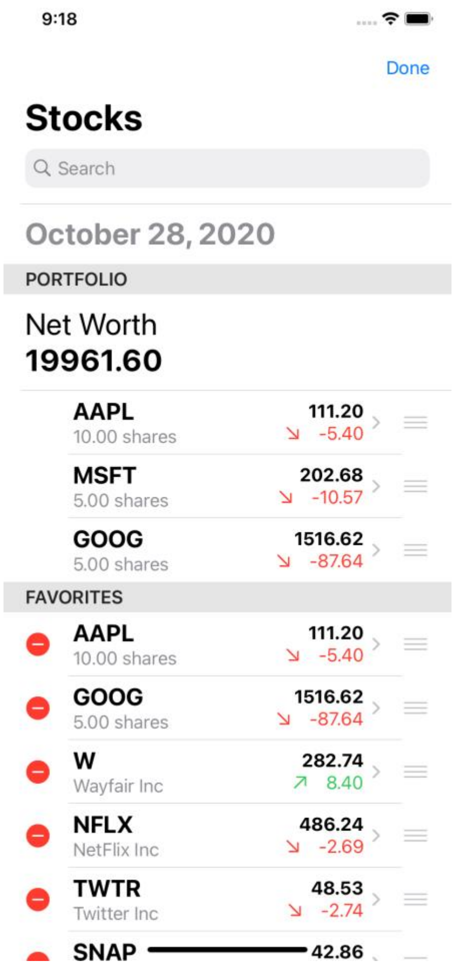
Additionally, the symbol next to the change in price value should either be trending down or up based on the change price value. For example, if the change in price is positive, an arrow.up.forward

SFSymbol should be displayed and the symbol as well as the change price value should have green color. In case the change in price is zero i.e. no change, no symbol should be displayed, and the change price value should be grey. **Both of the symbols can be found in the SFSymbol2 app, every symbols for this homework can be found in the app.**

Each stock listing also has a right arrow on the right of the current price field. This should be automatically added by SwiftUI if you embed your ListView inside a NavigationView. When clicking on a list item, the stock detail screen will open for the selected stock.

The List View also support delete and move functionalities, the ListView should change the layout after Edit button on the top right is clicked:

- **Delete functionality** allows the user to remove/delete the stock from the **Favorites** section. On removing a stock from the favorite section, the stock should be removed from the favorite stocks section.
- **Move functionality** allows the user to reorder the stocks in **either section**. The user should be able to long press on the right of the stock listing and drag it to a new position. The list should be updated accordingly to ensure the new order going forward. Note: The user cannot drag the stock from the favorite section into the portfolio section. The stock can only be dragged and dropped in the same section.


9:18

Done

Stocks

Search

October 28, 2020

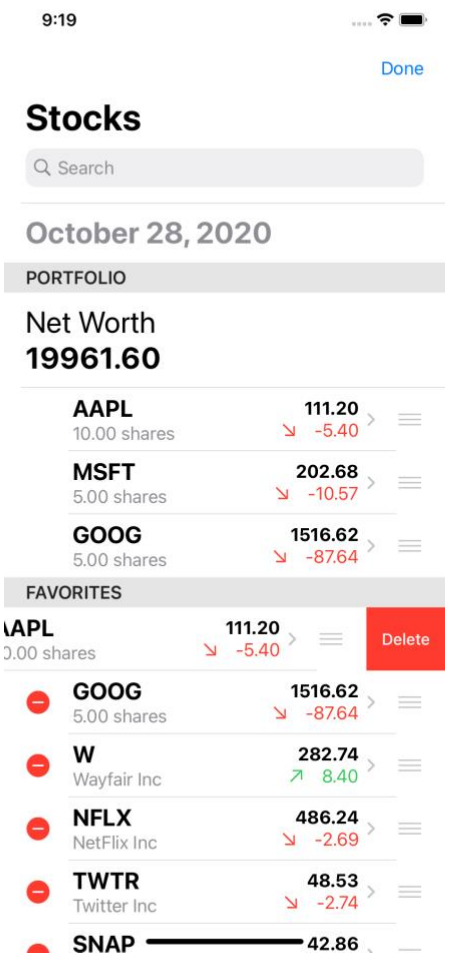
PORTFOLIO

Net Worth
19961.60

AAPL	10.00 shares	111.20 ↘ -5.40
MSFT	5.00 shares	202.68 ↘ -10.57
GOOG	5.00 shares	1516.62 ↘ -87.64

FAVORITES

AAPL	10.00 shares	111.20 ↘ -5.40
GOOG	5.00 shares	1516.62 ↘ -87.64
W	Wayfair Inc	282.74 ↗ 8.40
NFLX	NetFlix Inc	486.24 ↘ -2.69
TWTR	Twitter Inc	48.53 ↘ -2.74
SNAP		42.86


9:19

Done

Stocks

Search

October 28, 2020

PORTFOLIO

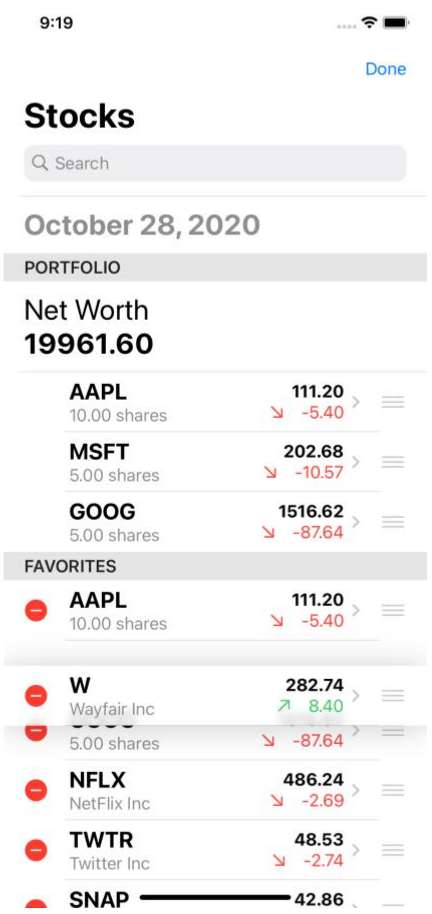
Net Worth
19961.60

AAPL	10.00 shares	111.20 ↘ -5.40
MSFT	5.00 shares	202.68 ↘ -10.57
GOOG	5.00 shares	1516.62 ↘ -87.64

FAVORITES

APL	.00 shares	111.20 ↘ -5.40 Delete
GOOG	5.00 shares	1516.62 ↘ -87.64
W	Wayfair Inc	282.74 ↗ 8.40
NFLX	NetFlix Inc	486.24 ↘ -2.69
TWTR	Twitter Inc	48.53 ↘ -2.74
SNAP		42.86

Edit Mode


9:19

Done

Stocks

Search

October 28, 2020

PORTFOLIO

Net Worth
19961.60

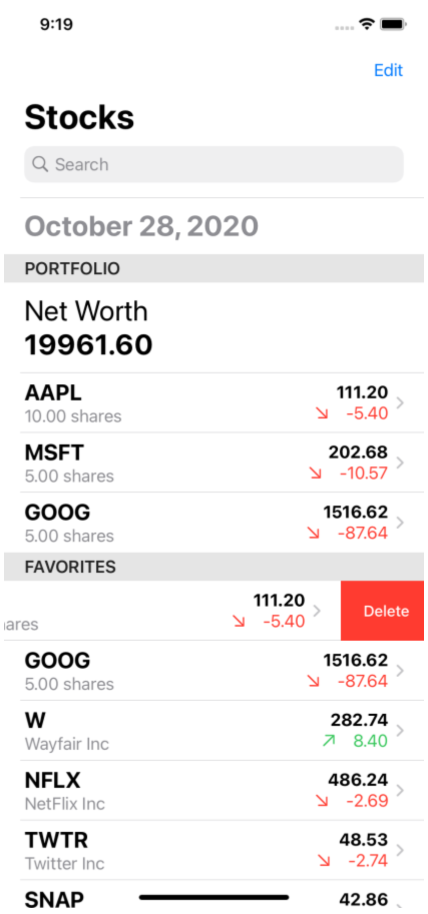
AAPL	10.00 shares	111.20 -5.40
MSFT	5.00 shares	202.68 -10.57
GOOG	5.00 shares	1516.62 -87.64

FAVORITES

AAPL	10.00 shares	111.20 -5.40
W	Wayfair Inc	282.74 8.40
	5.00 shares	-87.64
NFLX	NetFlix Inc	486.24 -2.69
TWTR	Twitter Inc	48.53 -2.74
SNAP		42.86

Move in Edit Mode

Delete in Edit Mode


9:19

Edit

Stocks

Search

October 28, 2020

PORTFOLIO

Net Worth
19961.60

AAPL	10.00 shares	111.20 -5.40
MSFT	5.00 shares	202.68 -10.57
GOOG	5.00 shares	1516.62 -87.64

FAVORITES

	ares	111.20 -5.40 Delete
GOOG	5.00 shares	1516.62 -87.64
W	Wayfair Inc	282.74 8.40
NFLX	NetFlix Inc	486.24 -2.69
TWTR	Twitter Inc	48.53 -2.74
SNAP		42.86

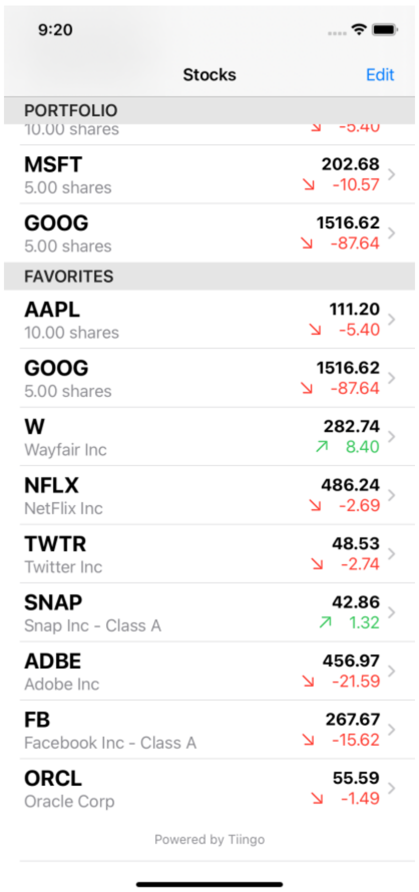
Swipe to Delete

The layout and behavior should be automatically added by SwiftUI after adding .onDelete and .onMove modifiers. You will need to add code in the modifiers to update your own data structure and the ListView.

When the "Edit" button is clicked, the text should change to "Done". When "Done" is clicked, the text should change back to "Edit".

SwiftUI also natively supports swipe to delete when .onDelete modifier is added. You need to have swipe to delete functionality in your app.

At the bottom of the ListView, we have a ‘Powered by Tiingo’ text. On clicking this text, the App should open "https://www.tiingo.com" in safari.


9:20

Stocks Edit

PORTFOLIO

10.00 shares	↘ -5.40
MSFT 5.00 shares	202.68 ↘ -10.57
GOOG 5.00 shares	1516.62 ↘ -87.64

FAVORITES

AAPL 10.00 shares	111.20 ↘ -5.40
GOOG 5.00 shares	1516.62 ↘ -87.64
W Wayfair Inc	282.74 ↗ 8.40
NFLX NetFlix Inc	486.24 ↘ -2.69
TWTR Twitter Inc	48.53 ↘ -2.74
SNAP Snap Inc - Class A	42.86 ↗ 1.32
ADBE Adobe Inc	456.97 ↘ -21.59
FB Facebook Inc - Class A	267.67 ↘ -15.62
ORCL Oracle Corp	55.59 ↘ -1.49

Powered by Tiingo

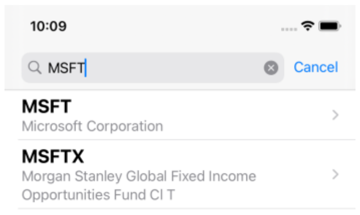
Powered by Tiingo Text

The field mappings for the stock's information is the same as Homework #8.

Note: The price information for each stock should be updated every 15 seconds, same as Homework #8. Additionally, the ProgressView should only be shown whenever the home screen is opened initially. While refreshing the stock price every 15 seconds, the spinner SHOULD NOT be displayed.

The home screen has been implemented by using a **NavigationView** and a **ListView**. Each of the stock listings has been implemented using **NavigationLink, HStack, VStack and Spacer.**

5.3 Search Functionality



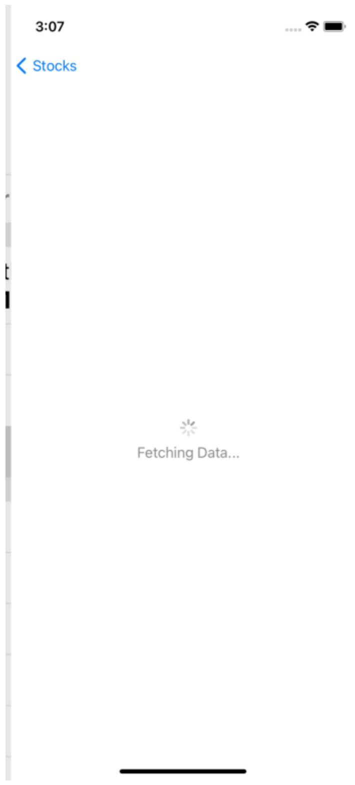
MSFT
Microsoft Corporation

MSFTX
Morgan Stanley Global Fixed Income Opportunities Fund Cl T

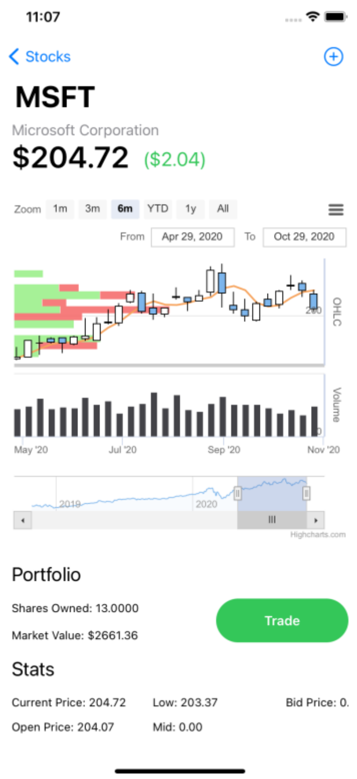
Search Functionality

- When pulling down from the home screen, a search bar should appear. When the search bar is clicked, the users will be able to enter a keyword to search for stock symbols.
- SwiftUI2 does not have an easy way to implement a native search bar. You **will need to** copy the code from here and use it. You will not be penalized for copying this code. Take a look at the README of this repo to use it.
- The user will get provided with suggestions of tickers from Tiingo Autocomplete API, through your nodejs backend. (Same as homework #8)
- When the user taps on a suggestion, the user goes to the stock detail screen.
- The search should redirect to the detailed information screen if and only if the user selected one of the autocomplete suggestions to fill the search field. Pressing Enter should not navigate to another screen.
- In the Autosuggest, only make an API call after the user enters 3 characters. Example: “am” should not display any suggestions but “ama” should have suggestions.
- Use debouncing to avoid calling the backend everytime a character is entered. You can implement debounce with "DispatchWorkItem?". See hints for example.

5.4 Detailed Stock Information Screen



Fetching Data



Shares Owned: 13.0000
Market Value: $2661.36

Current Price: 204.72 Low: 203.37 Bid Price: 0.
Open Price: 204.07 Mid: 0.00

Stock Detail Screen

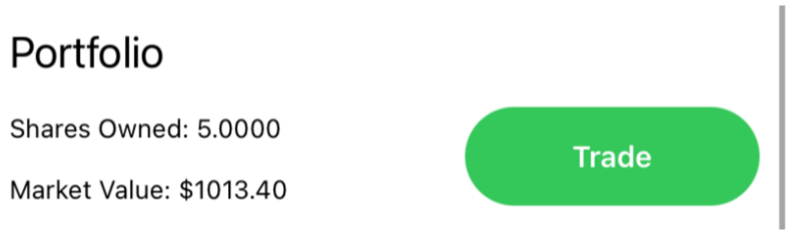
On clicking any stock on the home screen or in the search result list, the user is navigated to the stock detail screen. A ProgressView should be displayed while the details are being fetched while the stock detail page is pushed in from the right. Once the data has been fetched, except the chart (see detail of chart later), the spinner should disappear and information regarding the stock should be available to the user.

The NavigationBar should have a Stocks back button shrunk in size. This should be achieved through NavigationBar's native functionalities, **not by creating a custom component**. Clicking back button will go back to the home screen (which has the autosuggest result list that was used for the current search if the user enters the stock detail screen by using the search functionality). The NavigationBarTitle will be set to the ticker of the stock current viewing. The NavigationBar bar should also contain a favorite icon on the right to add or remove the stock from favorites. The favorite icon will either be filled or bordered based on whether the stock is favorited or not. **Adding/Removing the stock from favorites should also display different toast messages as shown in the video**. See hints for implementing toast.

Below the NavigationBar, there should be 3 fields: company name, current price with '$' sign, and the change price with '$' sign (the text color should be green, red or grey based on the change price value being positive, negative or zero respectively). The App then has a View which is blank till the Highcharts chart loads. (More details about it later in this section)

5.4.1 Portfolio Section

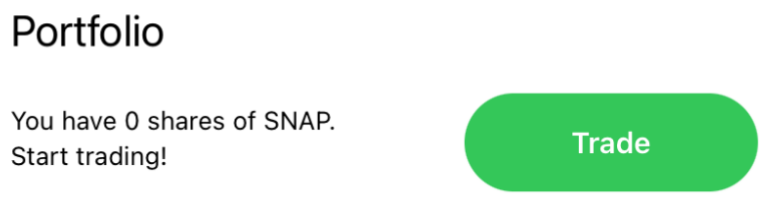
The **Portfolio section** allows the user to trade the shares of the stock. It contains a left section which shows the market value (2 decimals, same for all money related fields) of the stock in the user portfolio and the number of shares the user owns (4 decimals). The right section contains a customized green trade button.


Portfolio

Shares Owned: 5.0000

Market Value: $1013.40

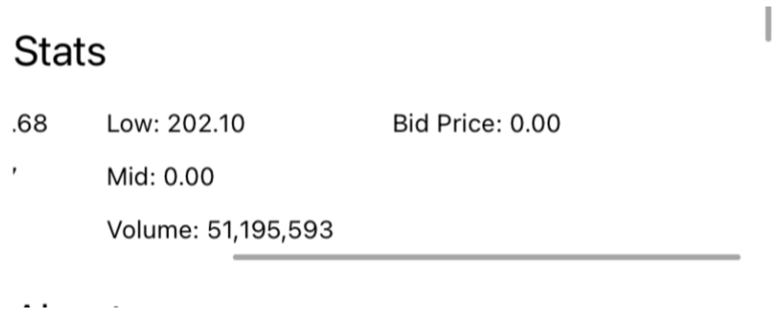
If the user does not own any shares of the given stock, the left section will have the message as shown below.


Portfolio

You have 0 shares of SNAP.
Start trading!

5.4.2 Portfolio Section

The **Stats section** displays the trading statistics for the given stock in a grid that is scrollable horizontally. The grid has 7 fields namely: Current Price (last price in API), Open Price, High, Low, Mid, Volume, and Bid Price. If any of these fields are missing in the JSON, set them as 0.0. Swift might also default null to 0.0 when you parse the JSON object and set it to float. **ScrollView** and **LazyHGrid** elements are to be used for this section.


Stats

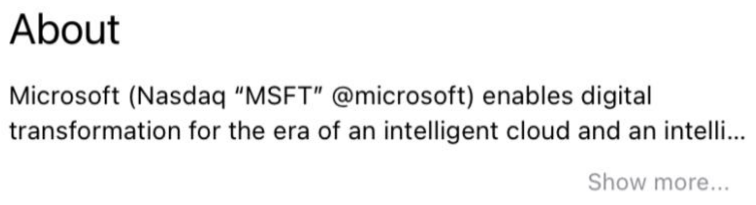
.68 Low: 202.10 Bid Price: 0.00

Mid: 0.00

Volume: 51,195,593

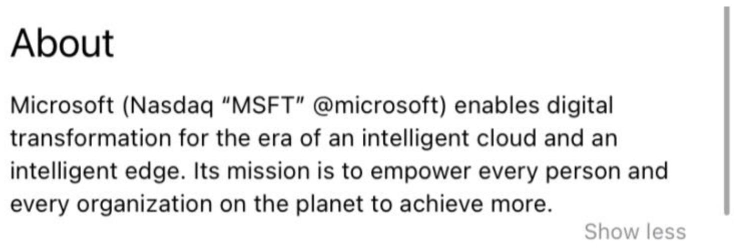
5.4.3 About Section

The **About section** displays the description of the company. If the description is longer than 2 lines, ellipsize the end of the 2nd line and display a 'Show more…' button.


About

Microsoft (Nasdaq “MSFT” @microsoft) enables digital transformation for the era of an intelligent cloud and an intelli...

On clicking this button, the complete description becomes visible and the button text changes to ‘Show less’. If the description is less than 2 lines, you can still display the buttons since there is no easy way for SwiftUI to get the number of lines yet.


About

Microsoft (Nasdaq “MSFT” @microsoft) enables digital transformation for the era of an intelligent cloud and an intelligent edge. Its mission is to empower every person and every organization on the planet to achieve more.

The API response used is the same **as HW8.**

5.4.4 News Section

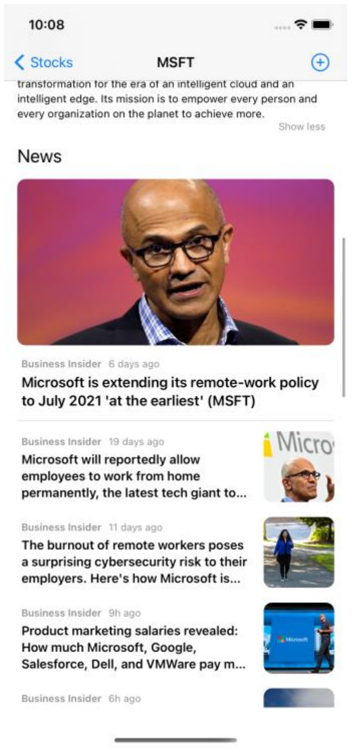
The **News section** displays the news articles related to the given stock symbol. The first article has a different format/layout than the rest of the articles in the list. A Divider() separates it from other news articles.

For each article, the information displayed is Article source, the time ago when the article was published, Article title, and Article image.

- You can load web images with KingFisher.(see the hints)
- Images should keep the aspect ratio, be cropped, and have a rounded corner.
- The time ago part should show ‘days ago’ if the timestamp is more than 24 hours old, 'hours ago' if the timestamp is more than 60 minutes old, and 'minutes ago' by calculating the difference between the timestamp the article was published and the current timestamp. (see the hints)

Do not limit title length for the first news article, limit title length to 3 lines for the rest of the news articles.

The news section uses **ListView** and **ContextMenu** elements.



transformation for the era of an intelligent cloud and an intelligent edge. Its mission is to empower every person and every organization on the planet to achieve more.

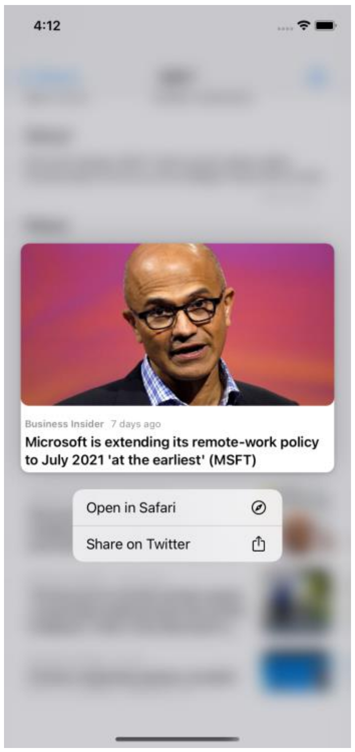
Business Insider 6 days ago
Microsoft is extending its remote-work policy to July 2021 'at the earliest' (MSFT)

Business Insider 19 days ago
Microsoft will reportedly allow employees to work from home permanently, the latest tech giant to...

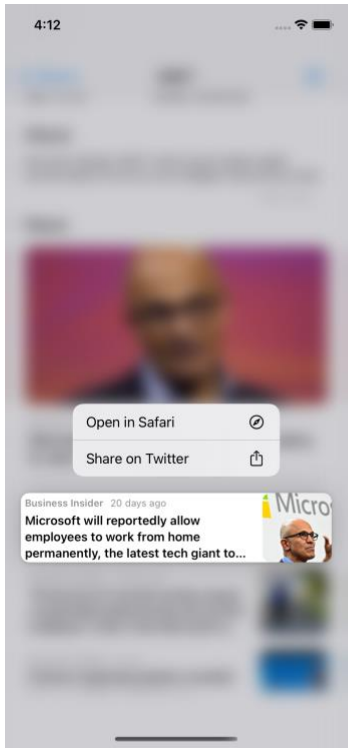
Business Insider 11 days ago
The burnout of remote workers poses a surprising cybersecurity risk to their employers. Here's how Microsoft is...

Business Insider 9h ago
Product marketing salaries revealed: How much Microsoft, Google, Salesforce, Dell, and VMWare pay m...

Business Insider 6h ago



Business Insider 7 days ago
Microsoft is extending its remote-work policy to July 2021 'at the earliest' (MSFT)



Business Insider 20 days ago
Microsoft will reportedly allow employees to work from home permanently, the latest tech giant to...

On clicking the news article, the original article is opened in safari using the article URL.

On long pressing the news article, a ContextMenu popup with options to open in safari and share on twitter.

- When the context menu is shown, the news article card that separates from the rest should have a rounded corner.
- When the context menu is shown, the news article card should have a consistent white background.
- Each item in the ContextMenu should be a Label, which is new in SwiftUI2. You can find the icons in the SFSymbols2 app.
- There might be a hard to debug layout problem when ContextMenu is opened because the list items have cropped images, refer to hints for help.

On clicking the button, the article should be shared by opening a browser with Twitter Intent. Use the following link to implement the Twitter Intent.
The Twitter message should be "Check out this link: <news url> #CSCI571StockApp". There are three parameters you will need to set for the intent.

Opening urls are the same as how you open "www.tiingo.com" on the home screen.

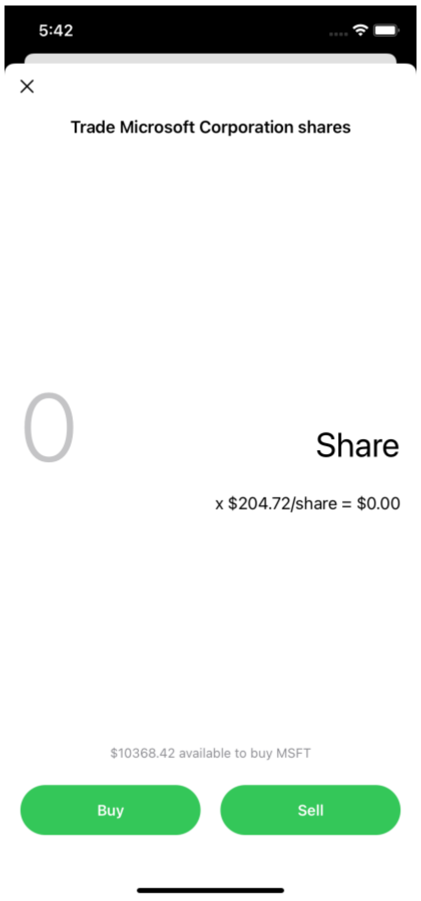
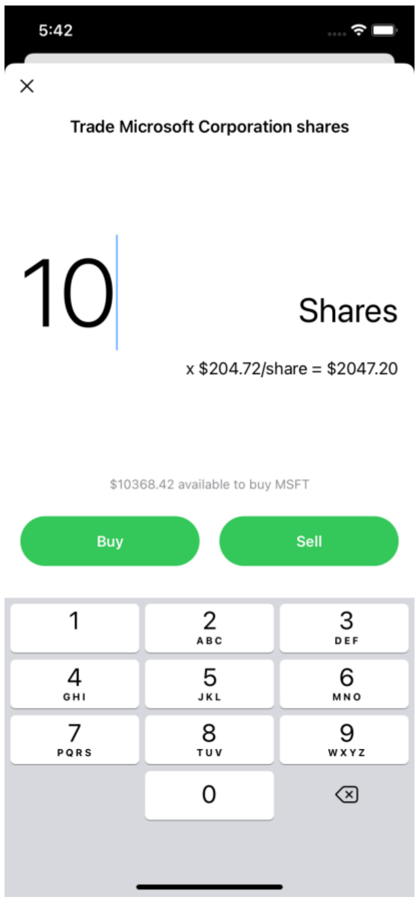
5.4.5 Trade Sheet

The Trade button in the **Portfolio section** opens a sheet for trading, see the hints for sheets in SwiftUI.

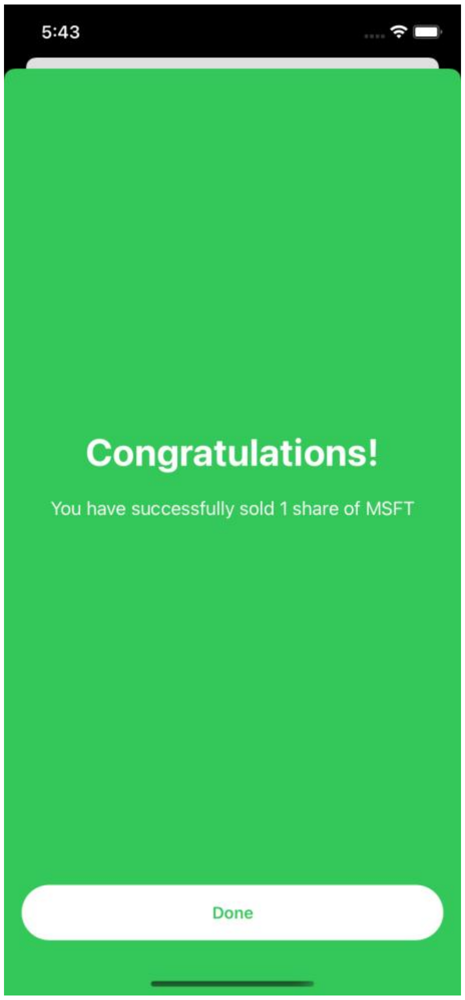
The sheet shows a close button on the top left corner. Beneath that is a "Trade <name> shares" text. In the middle is a TextField. You will also show a **numeric** keyboard on clicking the TextField (press command+k to toggle the keyboard in the simulator). Below the TextField, there is a calculation result which updates based on the numeric input to display the final price of the trade.

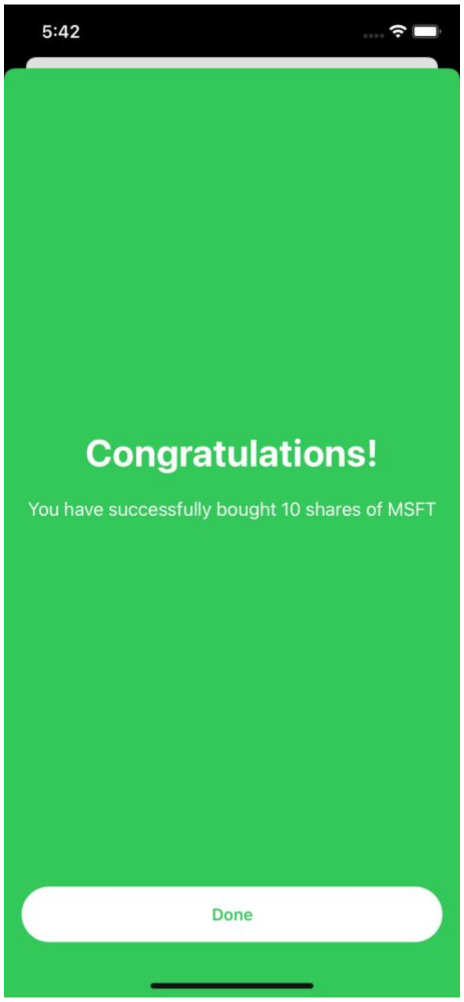
At the bottom, the trade sheet displays the current available amount of uninvested money available to trade for the user, and two buttons for the user to either buy or sell the amount of shares they have entered.

- If you implement this View correctly, the layout should shrink vertically when the software keyboard is toggled.
- It is okay to keep the TextInput's font size if the entered value is too long. Just keep its default behavior.
- Notice the text "Share" changes to "Shares" if the value is greater than 1.


5:42
×
Trade Microsoft Corporation shares
0
Share
x $204.72/share = $0.00
$10368.42 available to buy MSFT
Buy
Sell

5:42
×
Trade Microsoft Corporation shares
10
Shares
x $204.72/share = $2047.20
$10368.42 available to buy MSFT
Buy
Sell
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0
Trade Sheet

After the user presses Buy or Sell, and a trade is successful, you will show the trade success message. The message is different based on whether you have bought or sold 1 share or more shares.





Trade Success Message

There are 5 error conditions to be checked before executing the trade and displaying the trade successful message. The error conditions are:

- ***Users try to sell more shares than they own*** - a toast message with text 'Not enough shares to sell' should be displayed.
- ***Users try to buy more shares than uninvested money available*** - a toast message with text 'Not enough money to buy' should be displayed.
- ***User tries to sell zero or negative shares*** - a toast message with text 'Cannot sell less than 0 share' should be displayed.
- ***User tries to buy zero or negative shares*** - a toast message with text 'Cannot buy less than 0 share' should be displayed.
- ***User enters invalid input like text*** - The trade dialog box should remain open and a toast message with text 'Please enter a valid amount' should be displayed.

Refer to the video to see the toasts. You don't need to deal with corner cases such as user entering "inf"

5.4.6 Highcharts in iOS

The Chart at the top of the stock detail page is especially difficult to implement in SwiftUI2. The general idea is to display a WKWebView, load an HTML page, then call a JS function to fetch data and load the chart. We uses SMA Volume by price, the same as Homework #8, with almost no change to the sample code provided by highcharts.

The hard part is that WKWebView is a UIKit view and is fundamentally different from SwiftUI2. To embed UIKit views into SwiftUI, Apple has provided us a protocol called UIViewRepresentable. It might seem familiar to you because we used the same technique to implement the search bar on the home screen.

Thankfully, there are tutorials on how to implement a custom View to display html pages. Refer to these three links:

Using WebKit Delegates

Load local web files & resources in WKWebView

How to load HTTP content in WKWebView and UIWebView - free Swift 5.1 example code and tips

5.5 Local Storage

You should save both the list of favorite stocks, and your portfolio to local storage. When the app is opened, you should load them, then fetch data from your backend.

The old way of dealing with small amounts of data storage is using UserDefaults. In SwiftUI2, a new @AppStorage wrapper is introduced, and is just a new way to use the same UserDefaults.

The only difference is @AppStorage will trigger an UI update, while using UserDefaults will not.

- You cannot store objects or arrays of objects into UserDefaults(same applies to @AppStoreage). You need to manually encode and decode your objects into data. A lot of online resources teach you how to do that, do your searches.

If you have time to do some experiment, Core Data is a better way to store larger amounts of data. Using it is significantly harder. You will not get support from us.

5.6 ProgressView

Every time the user has to wait before they can see the data, you should display a ProgressView. The ProgressView is to be present across the **Home screen, Detailed Stock screen** and just says “Fetching Data…”. One idea would be to use an if-else statement in your view's body to check if certain fields of your view model is loaded (not nil). When those fields are fetched and loaded, your view model should trigger a view update because you are using @ObservableObject and @Published. Checkout Optional value from Swift language.

<u>Note:</u> **Based on your implementation, you might need to put progress bars on different places. During the demo, the only place that is allowed to be blank, is the WebView that loads HightCharts.**

5.7 Summary of detailing and error handling

1. Make sure there are no conditions under which the app crashes
2. Make sure all icons and texts are correctly positioned as in the video/screenshots
3. Make sure the screens, Views, and toasts are correctly displayed.
4. Make sure there is a progress bar on initial loading of the home screen, and loading stock detail page.
5. Make sure the styling matches the video/screenshots.
6. All API calls are to be made using Node.js backend, similar to what you have developed for homework 8.

5.8 Additional Info

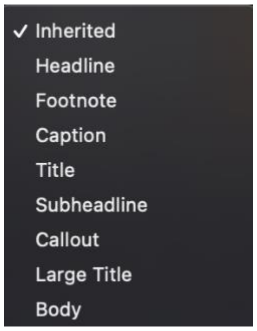
For things not specified in the document, grading guideline, or the video, you can make your own decisions. But keep in mind about the following points:

- Always display a proper message and don't crash if an error happens.
- You can only make HTTP/HTTPS requests to your backend Node.js on GAE/AWS/Azure. iOS can only load HTTPS responses, but you can manually allow it in plist. (see the hints)
- All HTTP requests should be asynchronous and should not block the main UI thread. You can use third party libraries like Alamofire to achieve this in a simple manner.

6. Implementation Hints

6.1 What are the fonts, font sizes, and colors used in this app?

The font is iOS's default font San Francisco. The font styles are all provided by the system, except the share number textinput in the trading view.



All of the colors used are all provided by SwiftUI. When you are using a modifier to change color, type ".", and XCode will provide autosuggestions for available colors.

6.2 Are Animations required?

Animations are required if the animation is provided by the system by default. For example, NavigationBar changes, search bar on click, edit home screen, delete and move functionalities on list items. The only exception are Toast messages, you will need to animate them. See slides for how to animate in SwiftUI.

The animation for the company description's Stats section expanding and collapsing, is not required, but it should be simple with two lines of code.

6.3 Other assets used

The images used in this homework are available in the zip file mentioned in the Piazza post for Homework #9.

You can either replace the Assets.xcassets folder in your project with the given one, or add the assets and edit Assets.xcassets folder under your project in Xcode yourself.

6.4 Good Starting Point

There will be a Piazza with links to great tutorials, here is a few core ones. Make sure to understand SwiftUI and iOS development before you start the homework. It will save you a lot of time.

Introduction to SwiftUI - WWDC 2020 - Videos

SwiftUI Tutorials

SwiftUI MVVM Programming with ObservableObject @Published @ObservedObject

All available View Modifiers

- SwiftUI cheat sheet

6.5 Third party libraries

Sometimes using 3rd party libraries can make your implementation much easier and quicker. Some libraries you may have to use are:

6.5.1 Alamofire

Alamofire is an HTTP networking library written in Swift.
https://github.com/Alamofire/Alamofire
We have provided code in slides for a way to use Alamofire

6.5.2 SwiftyJSON

SwiftyJSON makes it easy to deal with JSON data in Swift..
https://github.com/SwiftyJSON/SwiftyJSON
One easy way to use SwiftyJSON: How to create objects from SwiftyJSON, second answer. You can find the same code in slides.

6.5.3 Kingfisher

Kingfisher is a powerful, pure-Swift library for downloading and caching images from the web. It provides you a chance to use a pure-Swift way to work with remote images in your next app.
https://github.com/onevcat/Kingfisher

6.6 Displaying ProgressView

https://developer.apple.com/documentation/swiftui/progressview

6.7 Implementing Splash Screen

Launch screens in Xcode: All the options explained

6.8 Working with NavigationBar and App Navigation

Adding a navigation bar - a free Hacking with iOS: SwiftUI Edition tutorial

The Complete Guide to NavigationView in SwiftUI

6.9 Working with date and time

Working with dates - a free Hacking with iOS: SwiftUI Edition tutorial

DateFormatter
How to get time (hour, minute, second) in Swift 3 using NSDate?
How can I utilize SimpleDateFormat with Calendar?

6.10 Adding ToolBarItem to Toolbar

How to create a toolbar and add buttons to it - a free SwiftUI by Example tutorial

6.11 SearchBar

SwiftUI Search Bar in the Navigation Bar

6.12 Debounce

How to do throttle and debounce using DispatchWorkItem in Swift

6.13 Open Link in browser

How to open web links in Safari - a free SwiftUI by Example tutorial

6.14 Delete feature in ListView

How to let users delete rows from a list - a free SwiftUI by Example tutorial

List with drag and drop to reorder on SwiftUI

6.15 Move feature in ListView

How to let users move rows in a list - a free SwiftUI by Example tutorial

List with drag and drop to reorder on SwiftUI

6.16 Implementing ContextMenu

How to show a context menu - a free SwiftUI by Example tutorial

6.17 Image related

How to disable the overlay color for images inside Button and NavigationLink - a free SwiftUI by Example tutorial

SwiftUI Image clipped to shape has transparent padding in context menu

6.18 Highcharts related

Using WebKit Delegates

Load local web files & resources in WKWebView

How to load HTTP content in WKWebView and UIWebView - free Swift 5.1 example code and tips

6.19 Adding a Sheet for Trade page

How to present a new view using sheets - a free SwiftUI by Example tutorial

6.20 Adding Toasts

SwiftUI: Global Overlay That Can Be Triggered From Any View second answer

7. Files to Submit

You should also ZIP all your source code (the .swift/ and directories exclude other files) and submit the resulting ZIP file **if requested by the course staff**.

Unlike other semesters, you will have to demo your submission using **Zoom Remote Control** during a special grading session. Details and logistics for the demo will be provided in class, on the Announcement page and on Piazza. **Demo is done on a MacBook using the simulator, and not a physical mobile device.**

IMPORTANT
All videos are part of the homework description. All discussions and explanations on Piazza related to this homework are part of the homework description and will be accounted into grading. So please review all Piazza threads before finishing the assignment.