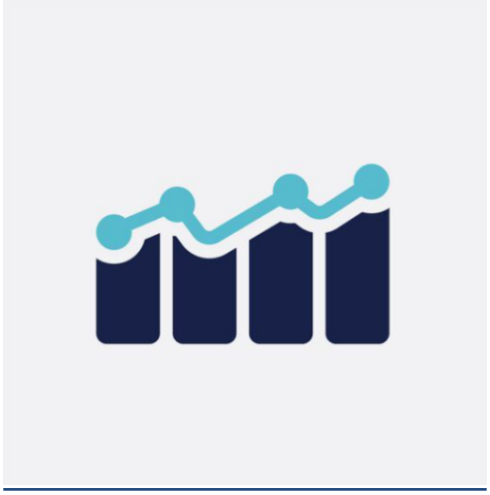


Homework 9
Stock Trading App

Prof. Marco Papa

Developed and Designed by
Aamulya Sehgal (iOS) & Nikhil Chakravartula (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 Finnhub 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 macOS iOS, iPadOS, watchOS and tvOS. First released in 2003, the latest stable release is XCode 13. It is available via the Mac App Store free of charge for macOS Moneterey.

Features including:

- Swift 5 support
- Playgrounds
- SwiftUI 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, macOS, 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 newer version, SwiftUI 2, was released earlier 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.

The Official SwiftUI homepage is located at:

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

2.5 SF Symbols

With over 3,300 symbols, SF Symbols is a library of iconography designed to integrate seamlessly with San Francisco, the system font for Apple platforms. Symbols come in nine weights and three scales, and automatically align with text labels. They can be exported and edited in vector graphics editing tools to create custom symbols with shared design characteristics and accessibility features. SF Symbols 3 features over 600 new symbols, enhanced color customization, a new inspector, and improved support for custom symbols.

The latest version, SF Symbols 3, was released earlier this September 2021. SF Symbols 3 features over 600 new symbols, including devices, health, gaming, and more.

These new symbols are available in apps running iOS 15, iPadOS 15, macOS Monterey Beta, tvOS 15, and watchOS 8.

All of the symbols used in this homework are available in SF Symbols 2 and above.

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 Node.js visit this page:

https://cloud.google.com/appengine/docs/standard/nodejs

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 Monterey (12).
You can develop the iOS app with MacOS Monterey. **MacOS versions earlier than Monterey may not be able to finish this homework. Use the latest MacOS version possible.**
If you meet problems such as missing symbols (system images) using SFSymbols, fail to compile or run SwiftUI code, your best option is to install latest MacOS.

IMPORTANT

This homework is developed with SwiftUI, **not** storyboards (except for the splash screen). We strongly suggest you develop the app with SwiftUI. 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 13**. Xcode 13 provides the crucial functionalities you will need to develop SwiftUI apps. You can download Xcode 13 by searching "Xcode" in your mac's app store.
- **Download and install SF Symbols 3**, this app provides all 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/SwiftUI Development, Hints are going to be your best friends!

4. High Level Design

This homework is a 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 9 API calls. They are the calls to the Finnhub APIs (through the Node.js) for company profile, stock quotes, symbol autocomplete, stock news, recommendation trends, social sentiments, company peers, company earnings, hourly chart data points and historical chart data points. Each of these 9 API calls are the same as Homework #8. So, you can use the same Node.js backend as HW8. 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 and up, not storyboard.

PS: This app has been designed and implemented in an iPhone13/Pro simulator. It is highly recommended that you use the same simulator to ensure consistency.

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

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

5. Implementation

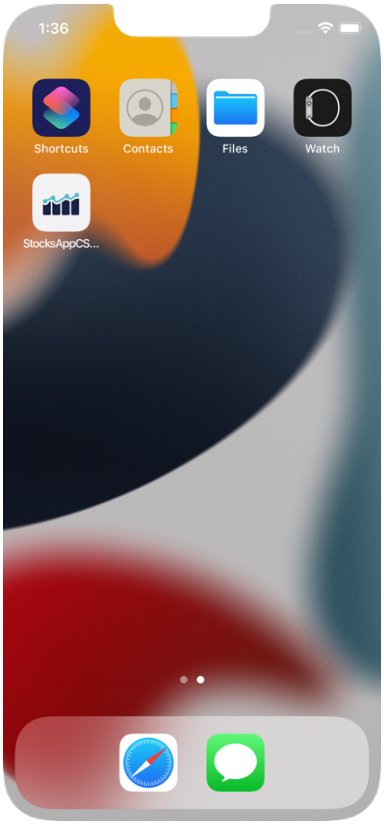
5.1 App Icon and Splash Screen

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.

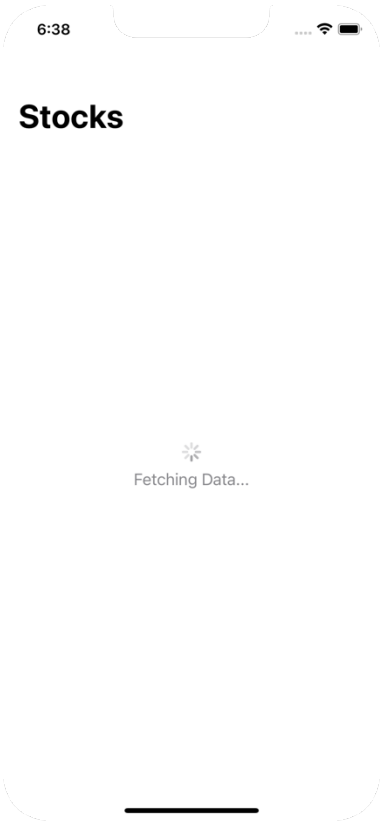


App Icon

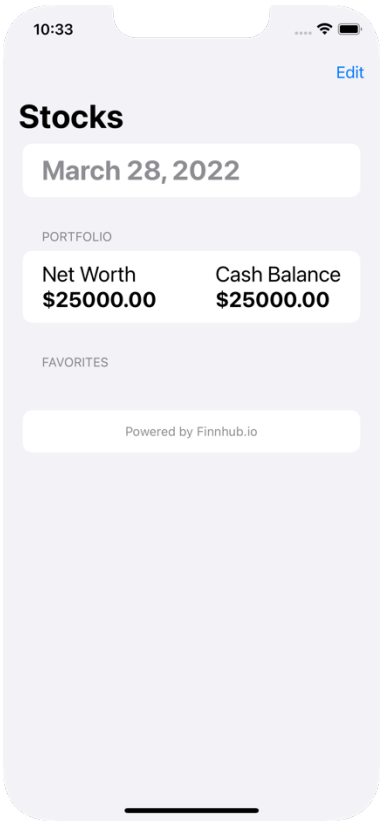
Splash Screen

5.2 Home screen

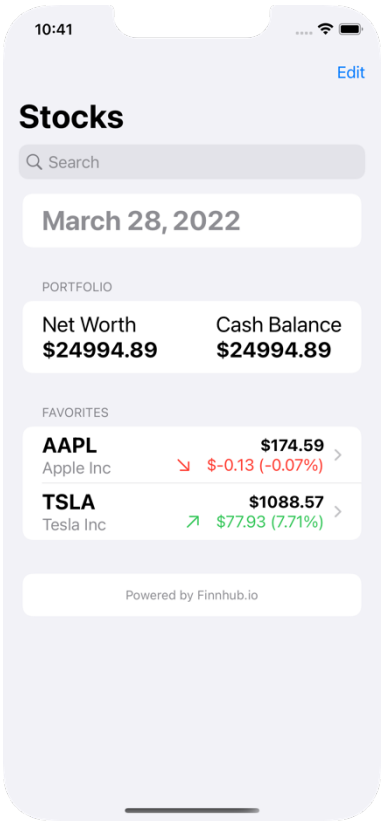
The home screen will have a Navigation Bar toolbar at the top with title “Stocks”. A search bar is attached to the navigation bar and will be revealed if you scroll up.



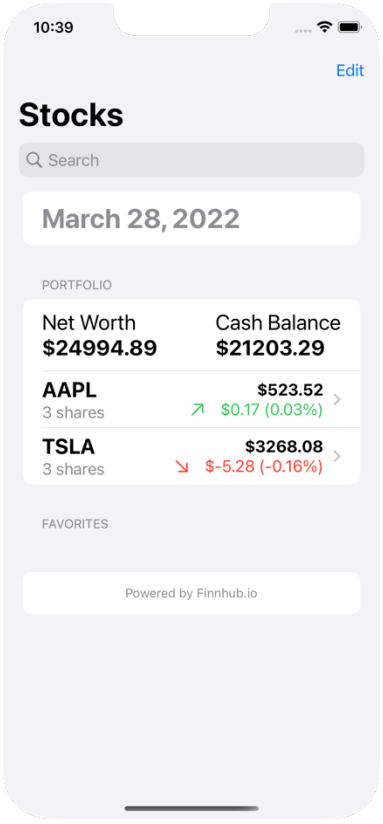
Fetching Data



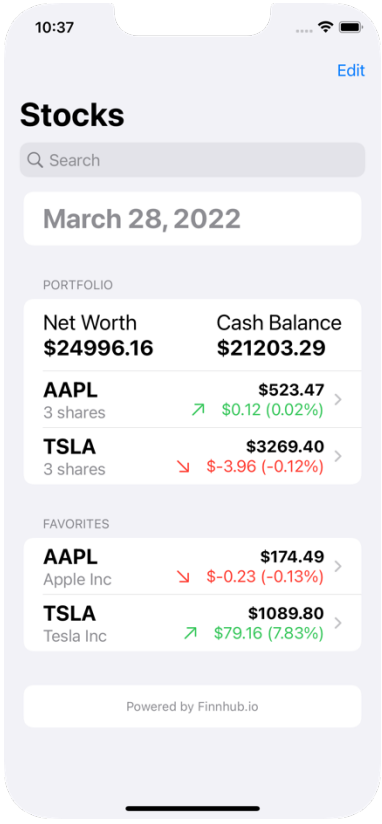
Home Screen (No Stocks)



Home Screen (Only Favorites)



Home Screen (Only Portfolio)



Home Screen (Portfolio and Favorites)

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:
        - Cash Balance - the money in the wallet currently for the user, which is uninvested cash
        - Net Worth – Cash Balance + total values of stocks owned
    - This is followed by the list of stocks in the user portfolio with that stock's:
        - Stock Symbol
        - Market Value - latest stock quote * number of shares owned
        - Change In Price From Total Cost – (latest stock quote – stock avg cost) * number of shares owned
        - Change In Price From Total Cost Percentage - (change in price from total cost / total cost of stock (stock avg cost * number of shares owned)) * 100
        - Total Shares Owned
    - 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 **$25,000** 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 and selling stocks, it can become more, or less than $25,000)
- ***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.
    - For each favorited stock show:
        - Stock Symbol
        - Current price - Latest Stock Quote
        - Change In Price (Since Last Close)
        - Change In Price Percentage (Since Last Close)

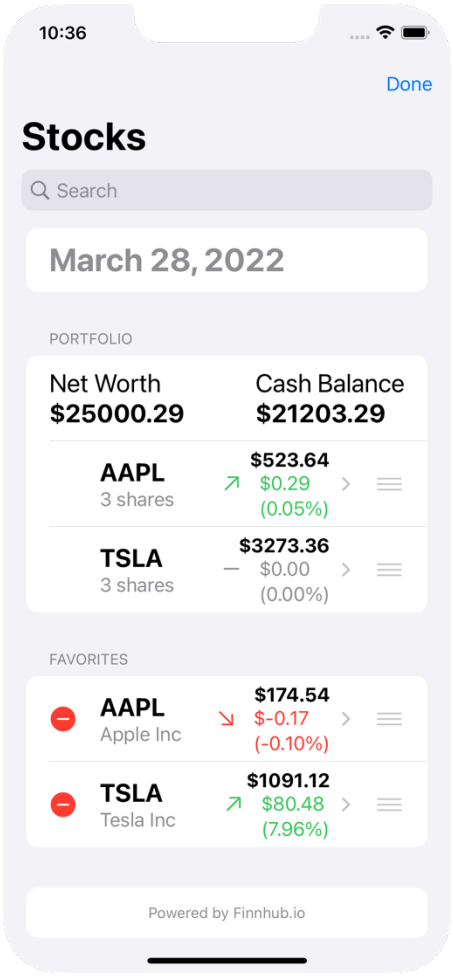
Additionally, the symbol next to the change in price values should either be trending down or up based on the change price value. For example, if the change in price is positive, an arrow.up.forward SFSymbol should be displayed and the symbol as well as the change price value should have green color. In case the change in price is negative, an arrow.down.forward SFSymbol should be displayed and the symbol as well as the change price value should have red color. If there is no change, then use the minus SFSymbol with gray color. **All of the symbols can be found in the SFSymbol 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.

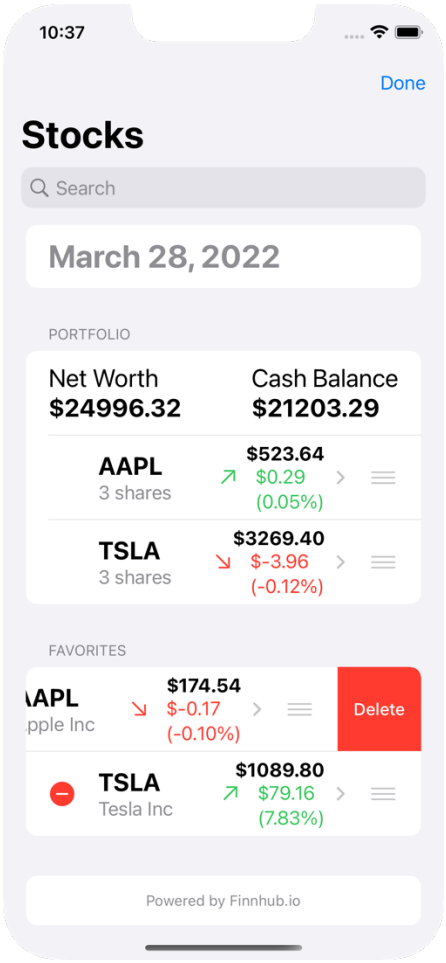
The List View also support delete and move functionalities, the ListView should change the layout after Edit button on the top right is clicked:

- **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.

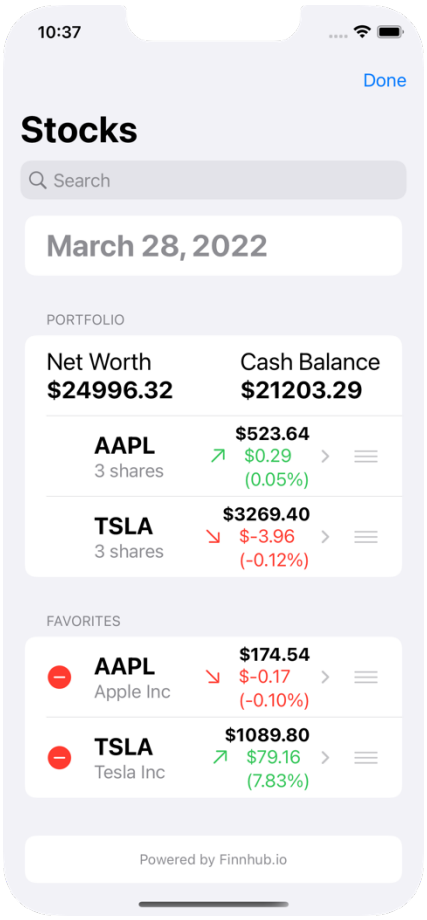
- **Move functionality** allows the user to reorder the stocks in **either section**. The user should be able to long press on the right of the stock listing and drag it to a new position. The list should be updated accordingly to ensure the new order going forward. **Note: The user cannot drag the stock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



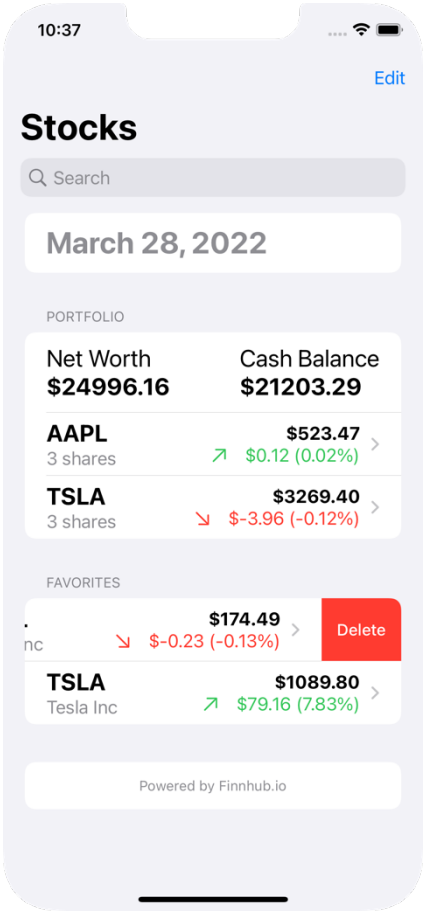
Edit Mode



Delete in Edit Mode



Move in Edit Mode



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.

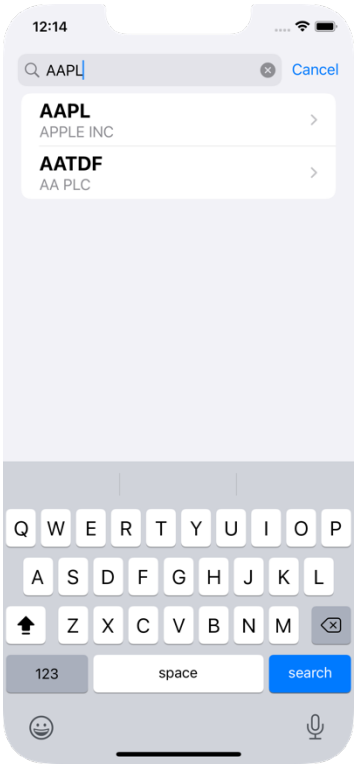
At the bottom of the ListView, we have a ‘Powered by Finnhub.io” text. On clicking this text, the App should open "https://www.finnhub.io"in safari.

All of the information is obtained same as homework 8 through the same backend.
The api response field mappings for the stock’s information are the same as Homework #8.

Note: The price information for each stock and “Net Worth” 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 prices and “Net Worth” 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.**

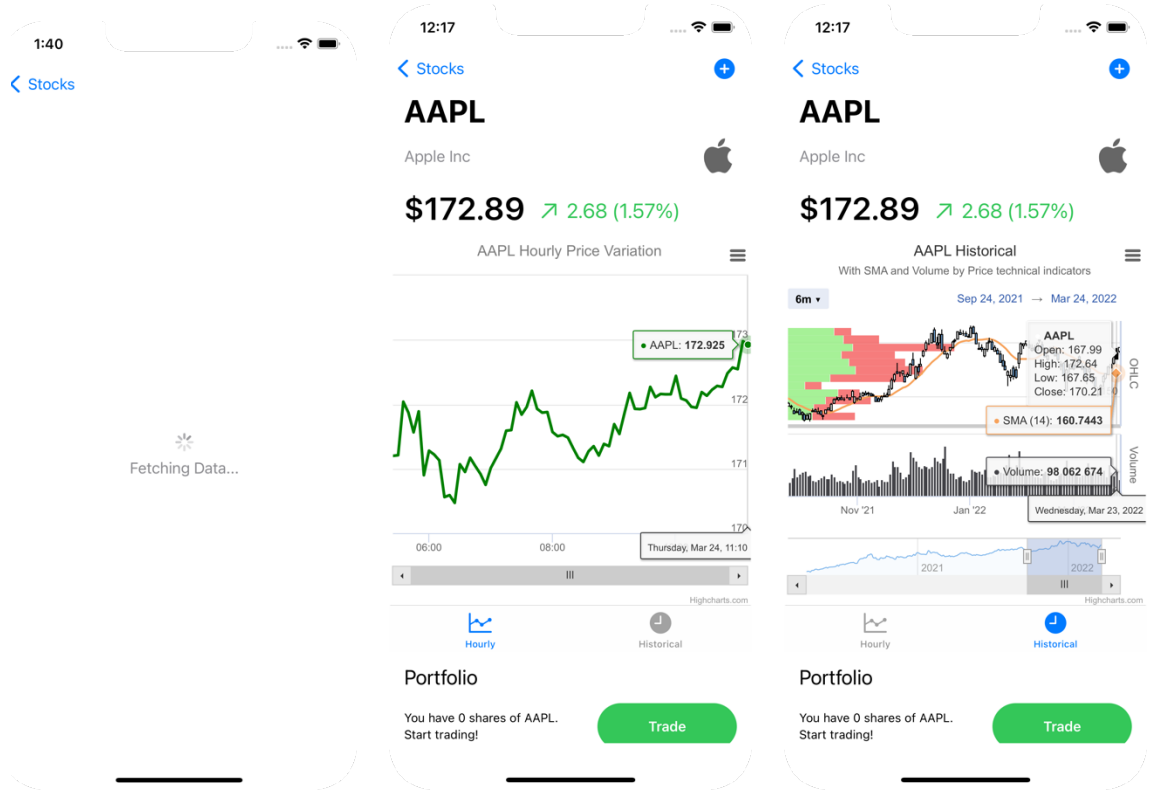
5.3 Search Functionality



Search Functionality

- When pulled 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.
- SwiftUI 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 through your Node.js 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.
- 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 | Stock Detail (Hourly) | Stock Detail (Historical)

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.

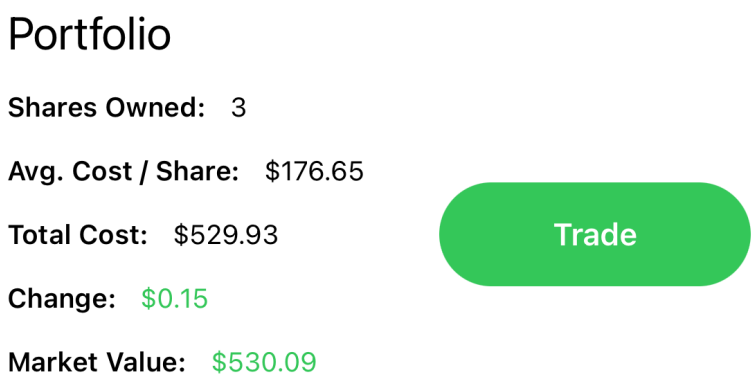
The NavigationBar should have a Stocks back button shrunk in size. This should be achieved through NavigationBar's native functionalities, **not by creating a custom component**. Clicking back button will go back to the home screen (which has the autosuggest result list that was used for the current search if the user enters the stock detail screen by using the search functionality). The NavigationTitle 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 5 fields: company name, company logo, current price with '$' sign, the change price with '$' sign (the text color should be green, red, or grey based on the change price value being positive, negative or zero respectively) and the change percentage with '%' sign.

Below the stock information is a TabView containing two Highchart charts for hourly and historical charts with chart.xyaxis.line and clock icons respectively. (More details about it later)

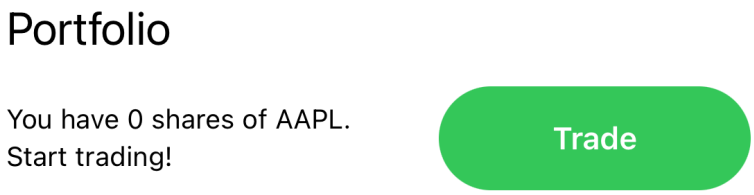
5.4.1 Portfolio Section

The **Portfolio section** allows the user to trade the shares of the stock. It contains a left section which shows the number of stocks the user owns, the avg cost per share, total cost of shares owned, change in price from total cost and the market value of the shares owned (2 decimals, same for all money related fields) of the stock in the user portfolio. All of these are obtained same as homework 8, **except for the change in price from total cost which is calculated as mentioned in the Portfolio Section of the Home Screen (Section 5.2)**. The right section contains a customized green trade button.



Portfolio when stocks are owned

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



Portfolio when no stocks are owned

5.4.2 Stats Section and About Section

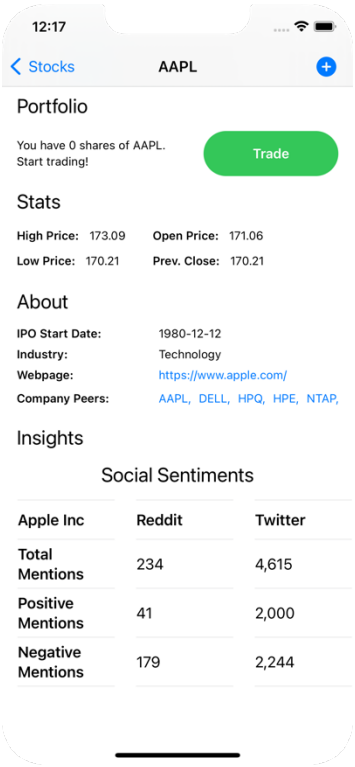
The **Stats section** shows various stats of the given stock symbol. These are:

- High Price
- Low Price
- Open Price
- Prev. Close

The **About section** shows various details about the given stock symbol. These are:

- IPO Start Date
- Industry
- Webpage (link should be clickable and open in Safari)
- Company Peers (as a horizontal scrollable list and should allow clicking a peer symbol to open its detail stock information screen)

All of these are obtained same as homework 8.



Stats and About Section

5.4.4 Insights Section

The **Insights section** has following three components:

- Social Sentiments Table
- Recommendation Trends Chart
- Historical EPS Surprises Chart

5.4.4.1 Social Sentiments Table

The **Social Sentiments Table** shows information organized in three lists:

- First List
  - Stock Name Header
  - Total Mentions Header
  - Positive Mentions Header
  - Negative Mentions Header
- Second List
  - Reddit Header
  - Reddit Total Mentions Value
  - Reddit Positive Mentions Value
  - Reddit Negative Mentions Value
- Third List
  - Twitter Header
  - Twitter Total Mentions Value
  - Twitter Positive Mentions Value
  - Twitter Negative Mentions Value

All of these are obtained same as homework 8.

5.4.4.2 Recommendation Trends Chart

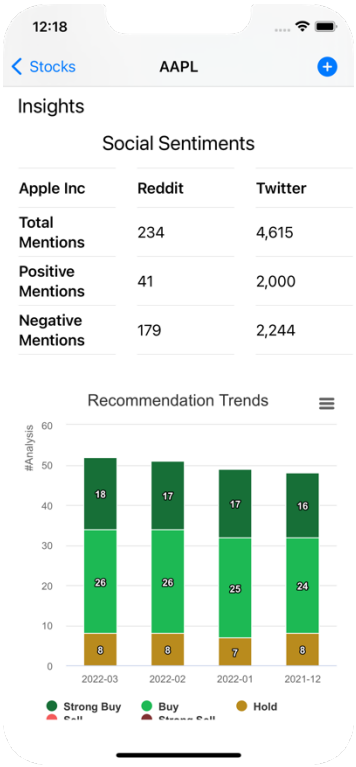
The **Recommendation Trends Chart** shows recommendation trends for the stock symbol.

All of these are obtained same as homework 8.

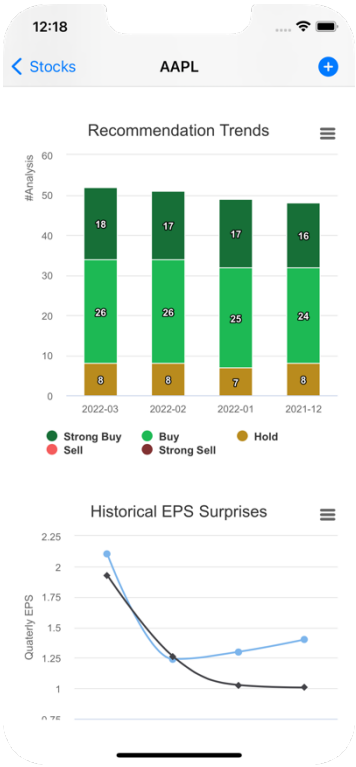
5.4.4.3 Historical EPS Surprises Chart

The **Historical EPS Surprises Chart** shows company earnings data for the stock symbol.

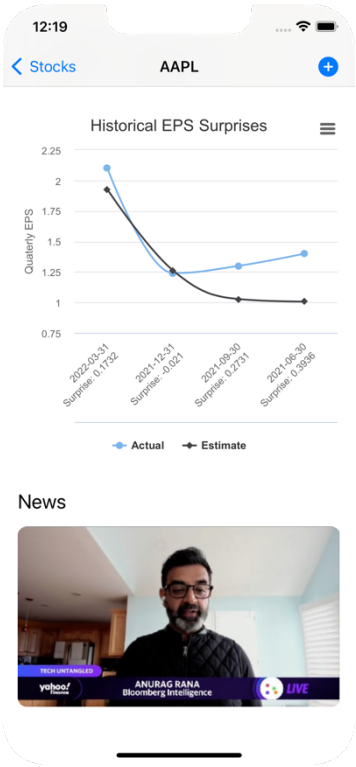
All of these are obtained same as homework 8.



Social Sentiments



Recommendation Trends



Historical EPS Surprises

5.4.5 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, article published ago (time since article was published relative to now), article image and article title.

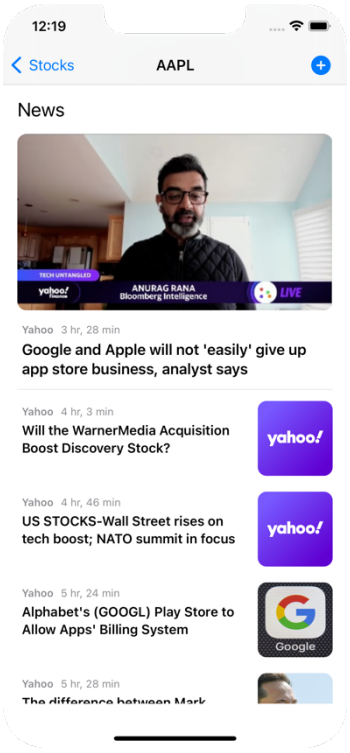
- You can load web images with KingFisher. (see the hints)
- Images should keep the aspect ratio, be cropped, and have a rounded corner.
- The news section uses **ListView** elements.

Clicking on a news article opens a sheet for more detail.

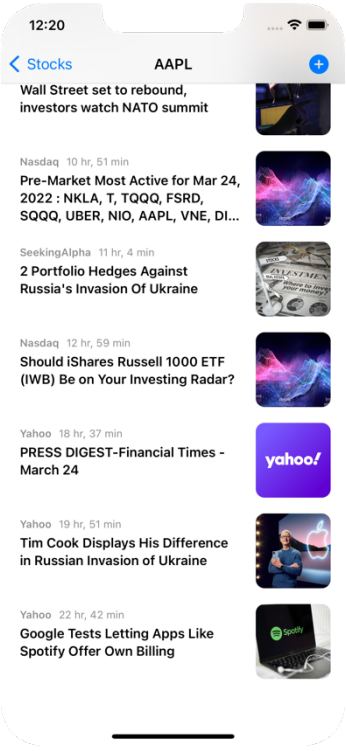
The sheet shows a close button on the top right corner. The following information should be shown below it:

- Article Source

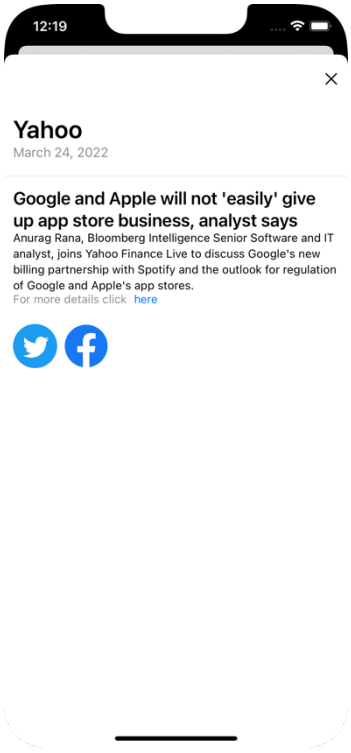
- Article Published Date
- Article Title
- Article Description
- Article Link
- Twitter Button (clicking it should open Twitter into safari with title and url being shared)
- Facebook Button (clicking it should open Facebook into safari with url being shared)



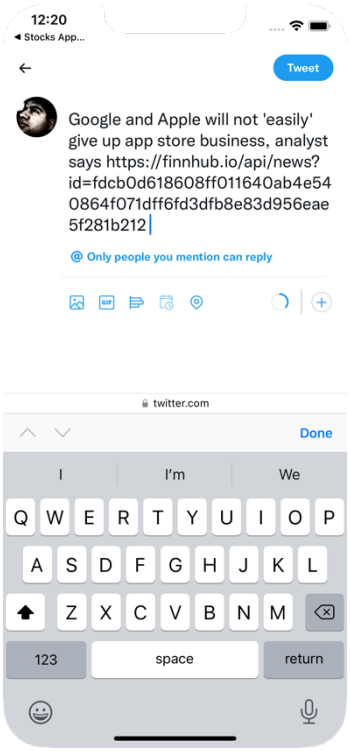
Large News Article



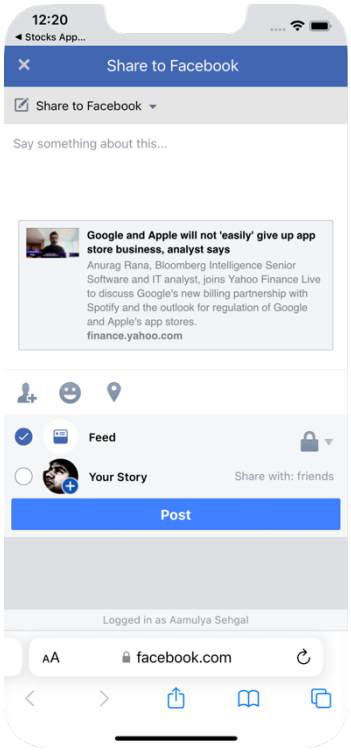
Small News Article



News Details



Twitter



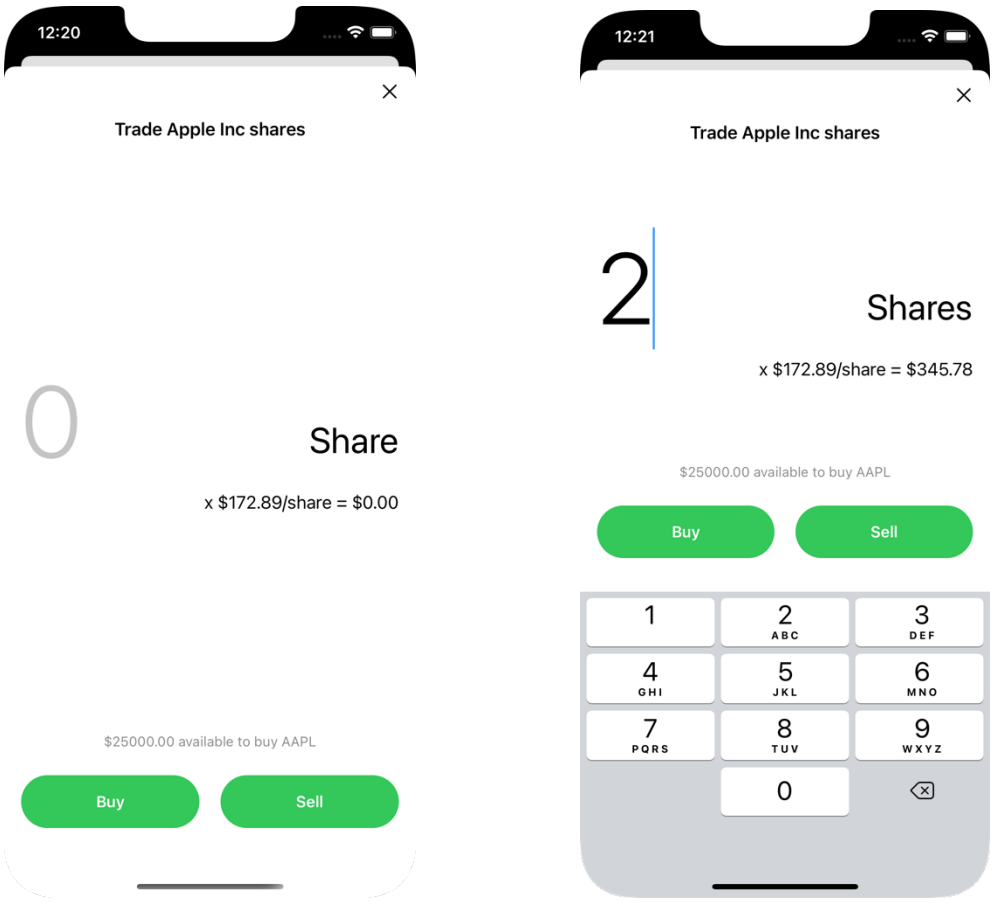
Facebook

5.4.6 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 right 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.

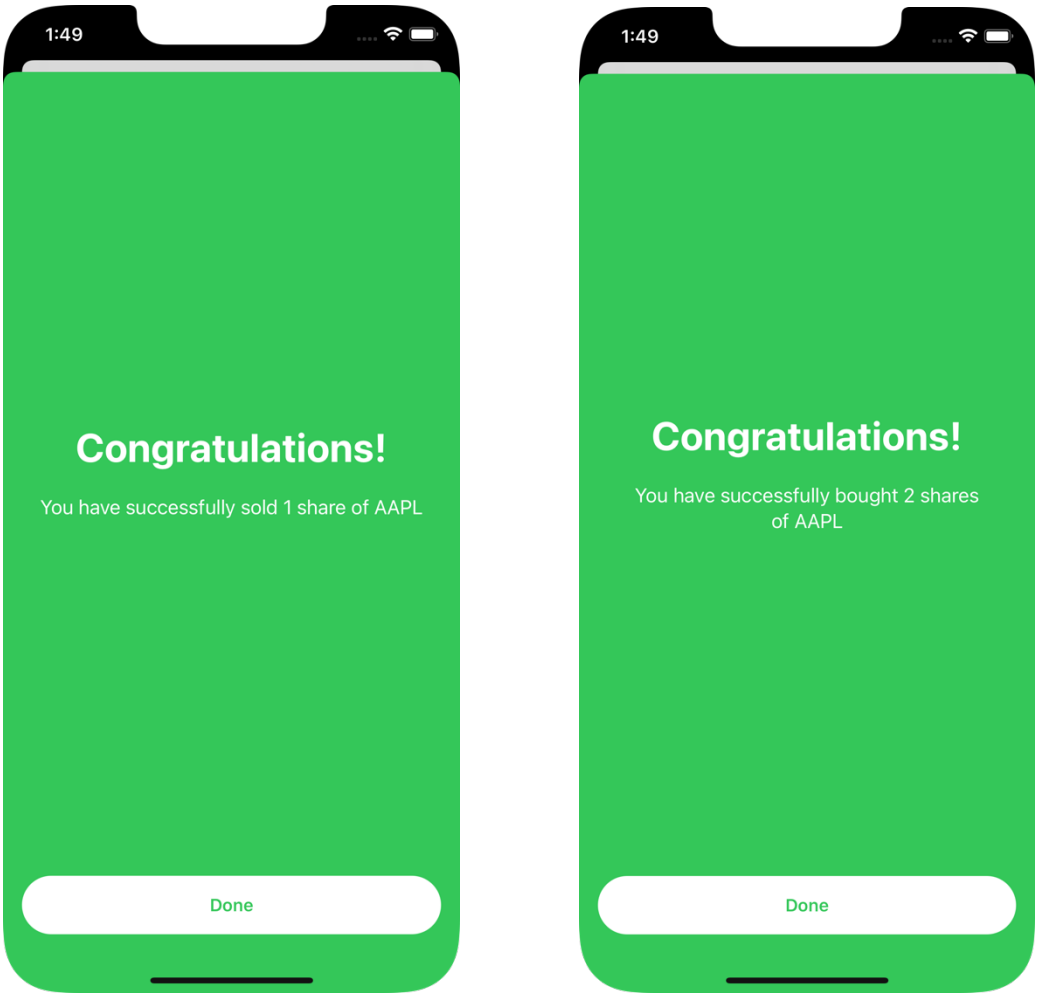
At the bottom, the trade sheet displays the current available amount of uninvested money available to trade for the user, and two buttons for the user to either buy or sell the number of shares they have entered.

- 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.



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 non-positive shares' should be displayed.
- ***User tries to buy zero or negative shares -*** a toast message with text 'Cannot buy non-positive shares' 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.

5.4.7 Highcharts in iOS

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 use the chart for the last working day, chart for the historical

stock market data, charts for recommendation trends, and chart for company earnings data, the same as Homework #8.

The hard part is that WKWebView is a UIKit view and is fundamentally different from SwiftUI. 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 SwiftUI, 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 for Core Data.

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.

Note: 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, like what you have developed for homework 8.

5.8 Additional Info

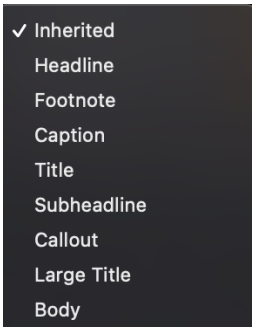
For things not specified in the document, grading guideline, the video, or in Piazza, you can make your own decisions. But keep in mind about the following points:

- 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 text input in the trading view.



All 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

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 TabView

Adding TabView and tabItem()

TabView

6.10 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?

6.11 Adding ToolBarItem to Toolbar

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

6.12 SearchBar

SwiftUI Search Bar in the Navigation Bar

6.13 Debounce

How to do throttle and debounce using DispatchWorkItem in Swift

6.14 Open Link in browser

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

6.15 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

6.16 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

6.17 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.

You will have to demo your submission using **Zoom**. Details and logistics for the demo will be provided in class, on the Announcement age and on Piazza. **Demo is done on a MacBook using the simulator, and not a physical mobile device.**

IMPORTANT

All videos, 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.