CSC 471 Mobile Application Development (iOS)

Starbucks Tracker iOS App

Final Project Documentation

By Shashank Indukuri

Demo Video Link

Table of Contents

1. Introduction

1.1 Purpose:

To keep up with today's fast-paced world, everyone needs a cup of coffee or tea. It is more common for everyone to get their coffee from Starbucks. However, in order to preserve health, it is necessary to keep track of caffeine and calorie intake.

1.2 Solution:

To address this issue, I created an iOS app that tracks the caffeine and calorie content of Starbucks beverages.

2. Project Description

2.1 Description:

- The "Starbucks Tracker" iOS application counts the calories and caffeine consumed by the user based on the Starbucks menu.
- The features of this application are that the user may add any drink from the menu and select from the customizable choices provided.
- Calories and caffeine are computed and presented on the home screen based on the selection.
- Users can quickly search for and choose beverages from the menu.
- The user may either repeat the existing drink or delete it from the home screen.
- To create a better user experience, the detail order view page contains adjustable choices to pick from, and the options are shown based on the kind of drink.
- Users may add as many beverages as they like and conveniently manage them from their home screen.
- The data added to the summary is kept in a file system and is permanent. As a result, users may easily return at any moment to monitor the count.

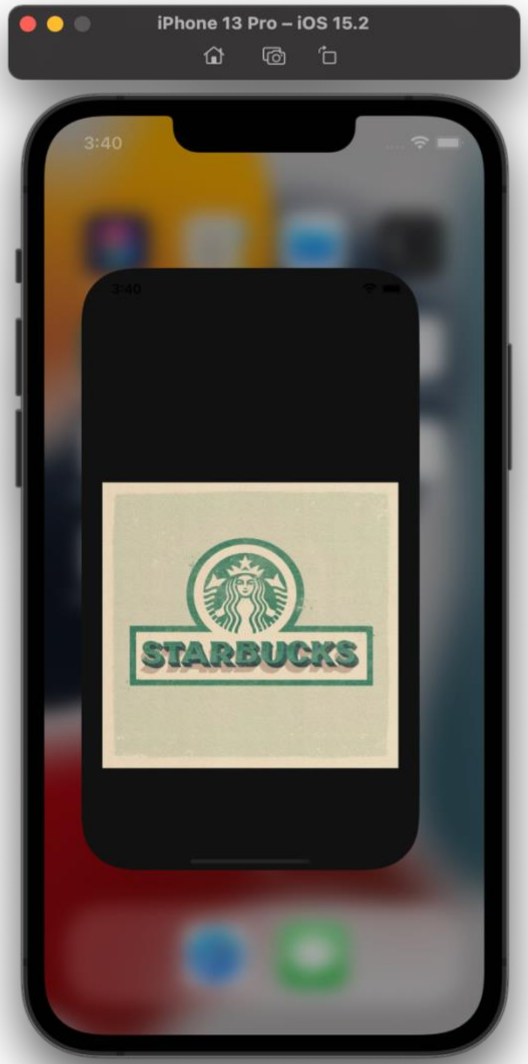
2.2 Technologies Used:

- Swift 5.5
- XCode 13
- SwiftUI
- GitHub
- JSON
- FileManager

2.3 Screens:

The screenshots from the app are shown below, along with explanations for each screen.

Launch Screen:



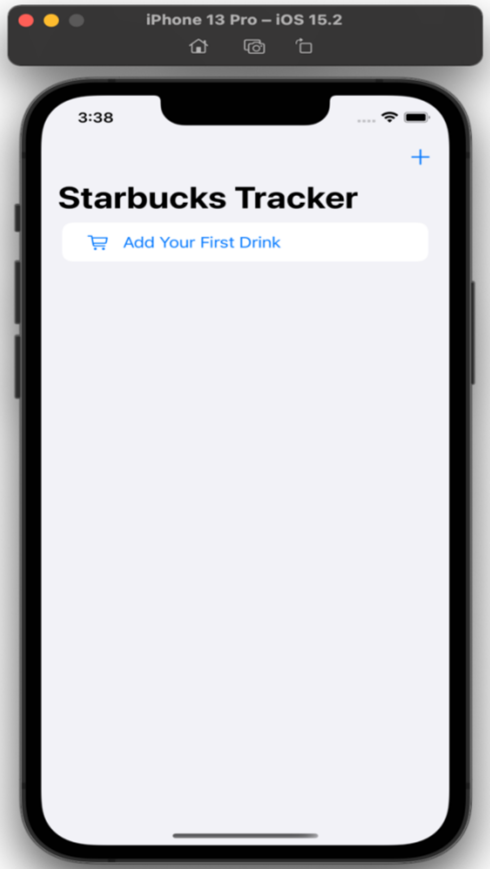
The start screen I added in my project is seen on the left in the screenshot. In Assets, I inserted an image and a color set for the background color to do this.

In info.pList, under the launch screen, I added an image with a string name linked to the asset image.

In addition, I set the property value with a color set and added a background color.

Finally, I added the image respect safe area insets attribute to ensure that the picture is completely shown on the screen.

Home Screen:

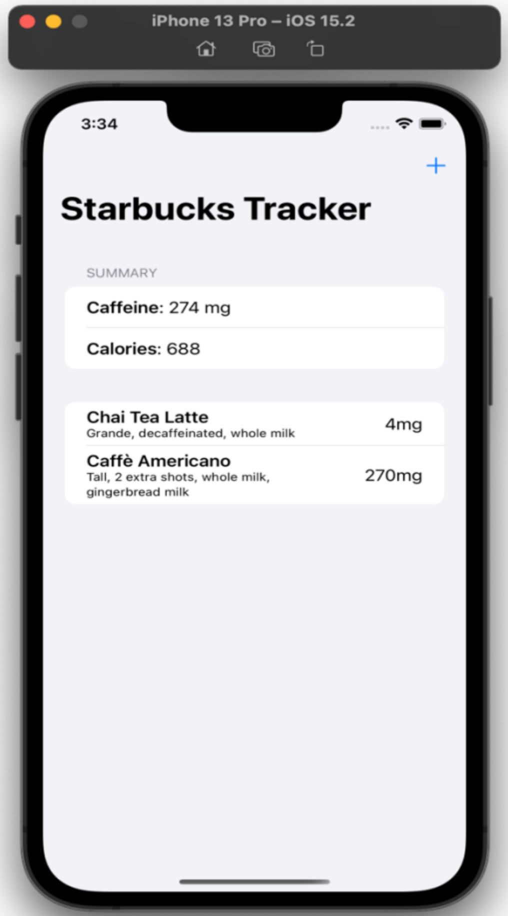

iPhone 13 Pro – iOS 15.2

3:38

Starbucks Tracker

Add Your First Drink

The screen on the right is the default home screen that appears when the app first loads. By clicking on "Add Your First Drink" or the "+" icon, we may add a new drink. I updated the navigation title to reflect the Starbucks Tracker.


iPhone 13 Pro – iOS 15.2

3:34

Starbucks Tracker

SUMMARY

Caffeine: 274 mg

Calories: 688

Chai Tea Latte
Grande, decaffeinated, whole milk
4mg

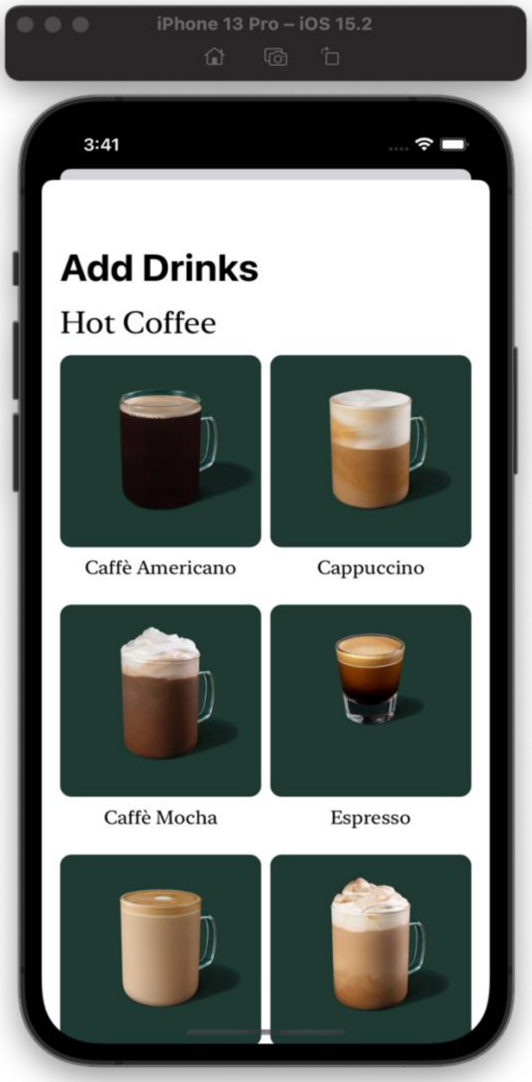
Caffè Americano
Tall, 2 extra shots, whole milk, gingerbread milk
270mg

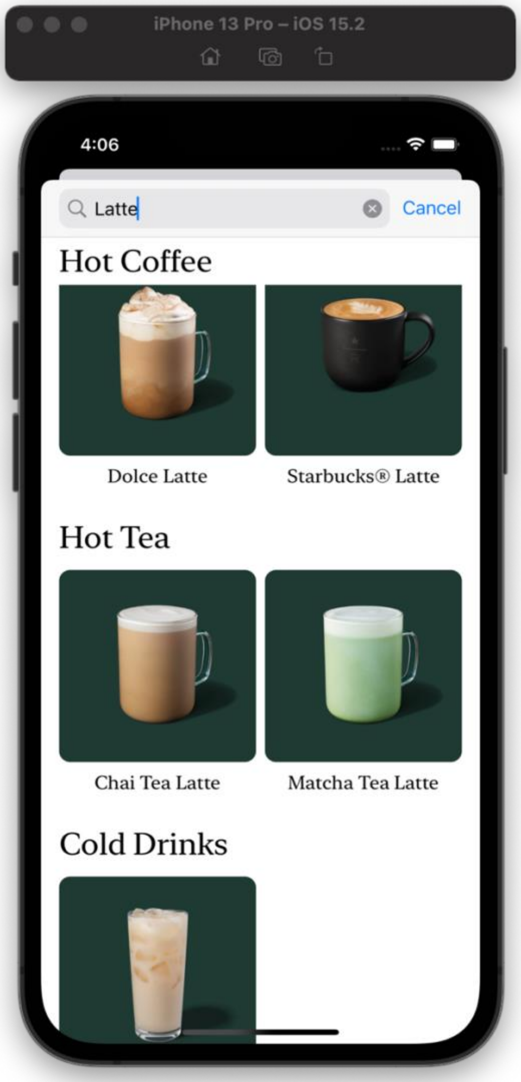
The homescreen on the left displays a summary of the beverages that have been added to the list. The Summary provides information about the overall caffeine and calorie content.

The drink list added to the menu is shown on the screenshot.

By just tapping on the button, the user may quickly repeat the order and erase the previous order.

Main Menu Screen:

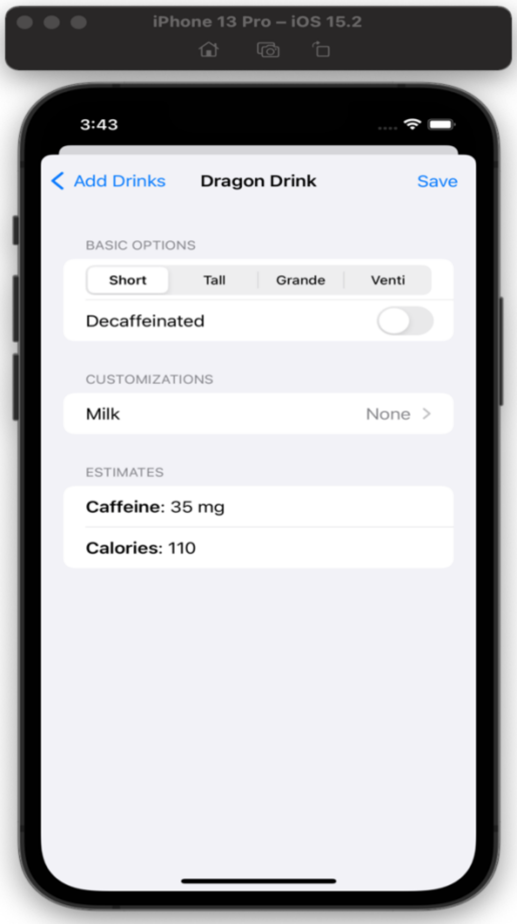




The screenshots shown above are from the main menu screen. When the user hits the "add drink" button, a menu appears, from which the user may select a drink from one of three categories. The categories are labeled "Hot Coffee," "Hot Tea," and "Cold Drinks" in this case.

I've included a search option so that users may look for beverages by entering in the name of the drink they're looking for. It filters the beverages and displays the corresponding drinks from each category. When a drink is selected, the user is sent to the detailed order screen.

Detail Order Screen:



Dragon Drink

BASIC OPTIONS

Short | Tall | Grande | Venti

Decaffeinated

CUSTOMIZATIONS

Milk None

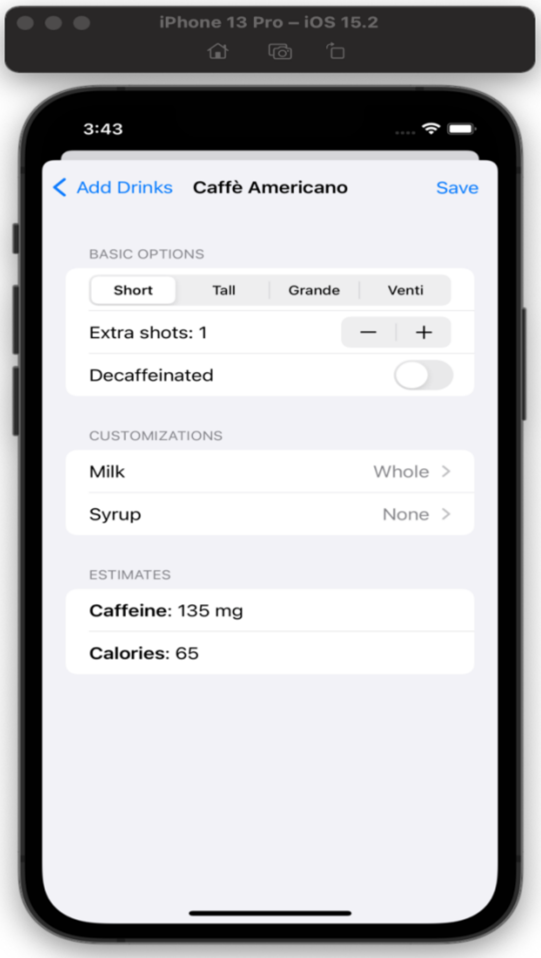
ESTIMATES

Caffeine: 35 mg

Calories: 110

The detail order view is shown on the right, where the user may personalize the drink depending on the available options. The customer may select the size, extra shots, milk, and syrup from this menu.

Caffeine and calorie estimates are automatically generated and shown based on the choices.



Caffè Americano

BASIC OPTIONS

Short | Tall | Grande | Venti

Extra shots: 1

Decaffeinated

CUSTOMIZATIONS

Milk Whole

Syrup None

ESTIMATES

Caffeine: 135 mg

Calories: 65

The Dragon Drink selections are displayed on the left side of the screen. Because it is neither coffee or milk-based, the alternatives for more shots and syrup are hidden here. Furthermore, it deselects the milk and sets it to the default of none.

B)

3. API Features

- I've saved all of the app's data in a JSON file. I'm taking data from a JSON file and mapping it to the app's relevant modal data using the safest methods. In this case, I kept sections, menu options, and syrup options data in JSON.
- I'm decoding the menu.json file and mapping it to the appropriate lists with JSONDecorder.
- The beverage summary on the home screen is stored in the local file manager permanently. I used the FileManager class to achieve this, and I constructed a storage path to save the summary and drink data whenever the user adds a new drink.
- To save the data in a file system, I used JSONEncoder to encode the serving data and write to the save path url.
- On the home screen, gesture and swipe motions are utilized to repeat and delete prior orders. Using SwiftUI's swipeActions, I added two buttons to a swipe gesture to repeat the order and erase the existing one.
- The drinks are sorted based on the order of addition. Moreover, all the drinks have their own unique UUID that identifies the specific item in the list.
- The search functionality is implemented to filter the menu based on the text entered. I used the method to filter the list based on the text and display the sections based on the return menu items.
- The auto layout issues are handled by implementing the interfaces through the use of SwiftUI. Hence, the app will have the same appearance on all Apple devices.
- I have used many controls like Buttons, Images, Switch, Stepper, Picker, Segment Control and Stacks.
- I have used navigation among the different screens and presenting the sheet to display the menu.

4. Biggest Challenges & Solutions

- One of the most difficult issues encountered throughout development was keeping the data organized among the views in a more effective manner. To overcome this, I studied and learnt how to communicate objects through the view from SwiftUI documentation. To address the problem, I utilized the @EnvironmentObject and @State properties.
- Another challenge was maintaining the data persistence in the filemanager storage. I spent many hours trying to understand the concepts of the FileManager class and how to save and retrieve the data. I got some references from the official documentation and debugged to find the bug and fix the error.

- Another difficulty I encountered was due to auto layout. It is really tough to implement restrictions in storyboards and keep the app consistent across all devices. To do this, I learned SwiftUI and created this app by utilizing vertical and horizontal stacks (VStack and HStack), which aided in the resolution of the problem.
- Another obvious problem I encountered was successfully storing and managing data. To address this, I utilized JSON to store the app's data, and the data is fetched and placed in the modal objects while the app loaded.

5. Limitations

- One of the app's limitations is that the menu data I acquired and included has fewer selections than the official Starbucks menu.
- Here, I was able to obtain caffeine and calorie information that matched the actual data. However, the caffeine and calorie calculations based on the drink and selections may not be exact and are just for research purposes.

6. Limitations on iOS SDK and XCode

- To begin with, storyboards make it simple to build user interfaces, and anybody with no prior coding skills may develop using drag and drop. The main disadvantage is the auto layout and imposing constraints. I utilized SwiftUI to address the problem.
- XCode is an all-in-one software for developing apps for all Apple platforms. However, older model MacBooks take a long time to open and load projects.

7. Experience

- Prior to attending this program, I was a Full Stack Web Developer with a strong interest in developing mobile apps. But I can't decide between Android and iOS. However, after considerable consideration, I decided to deal with iOS.
- The nicest thing about being an iOS developer is being able to effortlessly create elegant, efficient apps leveraging the power of Swift and XCode.
- Swift, in my opinion, is one of the cleanest and most feature-rich programming languages. I had a fantastic time making apps with Swift.
- As a newcomer to iOS, Storyboards made it simple for everyone to create user interfaces.
- The functionality and customizability of the user interfaces are amazing.
- I enjoy how Apple lets developers to handle the assets folder. It's the ideal feature to have because it makes our lives easier by organizing images effectively.
- Finally, I am really enthusiastic to continue my adventure as an iOS developer in order to learn and create more helpful apps.