Homework 9
Yelp Review App

Prof. Marco Papa

Developed and Designed by

Dr. Marco Papa and Megha Bagri

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 Yelp 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 Monterey.

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 have 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, **Section 6 Implementation 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 business information using the Yelp API and look at the detailed information about each business matching the search criteria. Additionally, users should also be able to make reservations at a restaurant and be able to manage various reservations. The App primarily contains the Business Search screen and the detailed business information view. However, the App also has multiple features on each of these views.

The API calls to be made to Yelp in this homework are exactly the same as Homework 8. Yelp APIs have to be called from the backend and hence the same Node.js backend can be used for this homework. 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 smartphone devices allowed.

5. Implementation

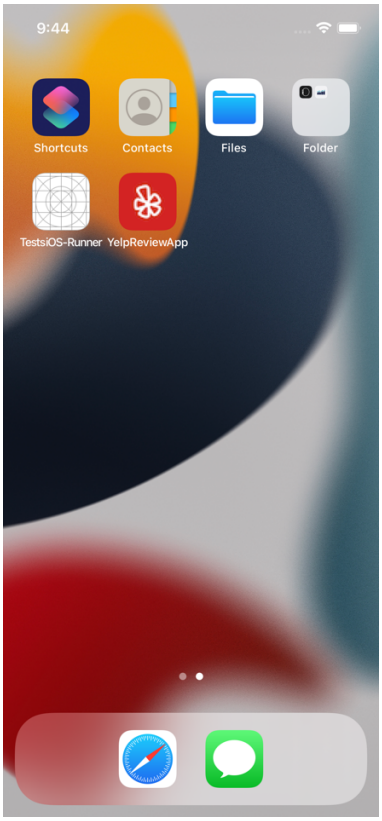
5.1 App Icon and Splash Screen

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: https://www.avanderlee.com/xcode/launch-screen/

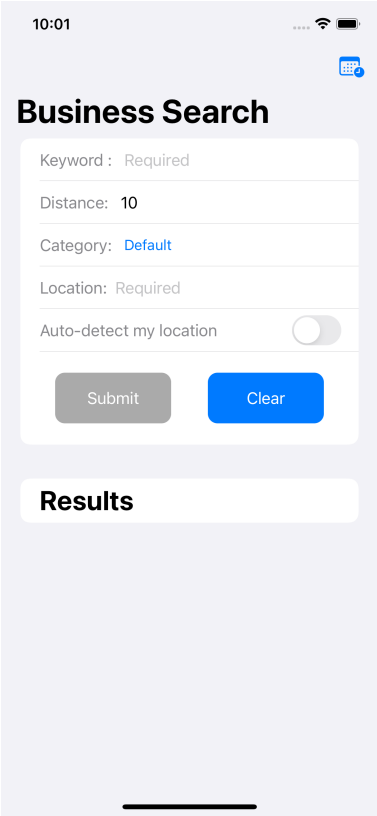
The image is in the assets provided.



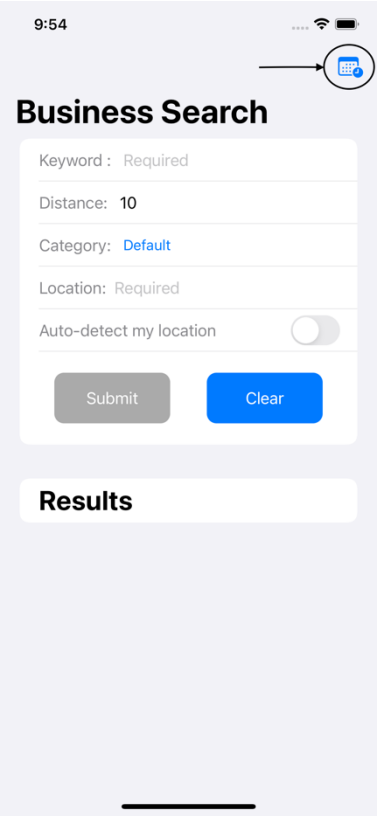
App Icon

5.2 Home screen

The home screen will have a Navigation Bar toolbar at the top with title “Business Search”. The Navigation Bar will contain an icon to view the bookings made by the user. The SF symbol “**calender.badge.clock**” must be used for the icon image.



Default Home Screen



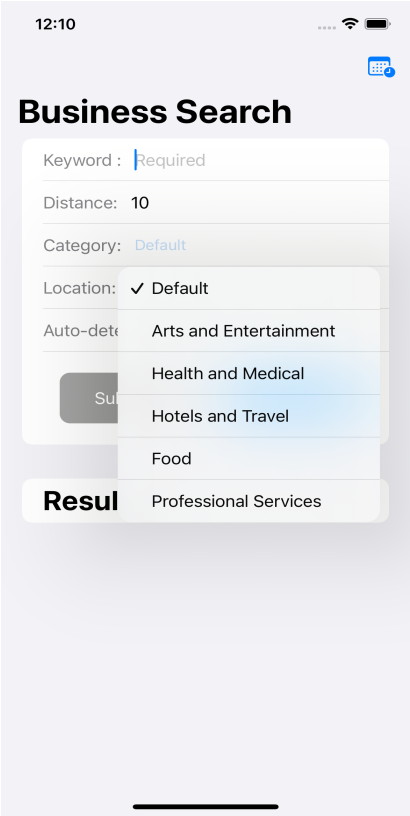
My Bookings icon on top left corner

Under the Navigation Bar, there are two main sections:

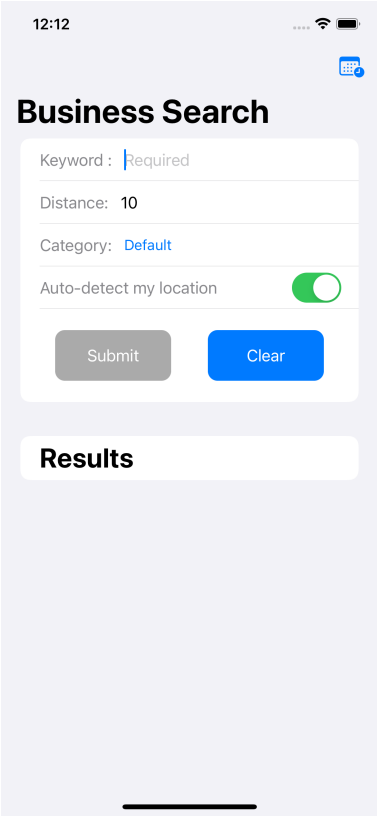
1. Business Search section
2. Results Section

5.2.1 Business Search section

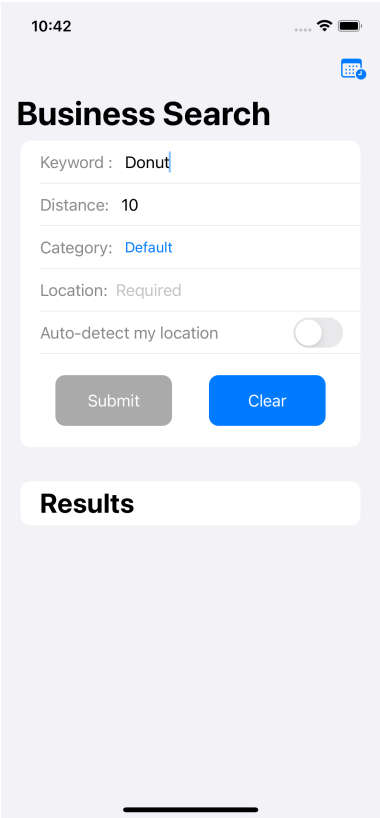
1. **Keyword**: This field should have prompt text of “Required” shown by default.
2. **Distance**: 10 miles has to be the default distance (Showing the value 10 is good enough)
3. **Category**: “Default” category option must be selected. When the user clicks on the value of the Category field, a “picker” must be shown bearing items “Defaults, Arts and Entertainment, Health and Medical, Hotels and Travel, Food, Professional Services”. The picker is shown in the figure below.
4. **Location:** This field should have prompt text of “Required” shown by default.
5. **Auto-detect my location**: A toggle switch must be shown. If the user wants to automatically detect location, then the Location field in point 4 must disappear.
6. **Submit Button**: Submit button must be greyed out initially. Only once the user enters all the fields mentioned above (either Location or Auto-Detect location), the Submit button should be in enabled by changing the color of the button to “red”.
7. **Clear Button**: When this button is clicked, all fields must be cleared out and the results section must be empty. The **Default Home Screen** must be shown.



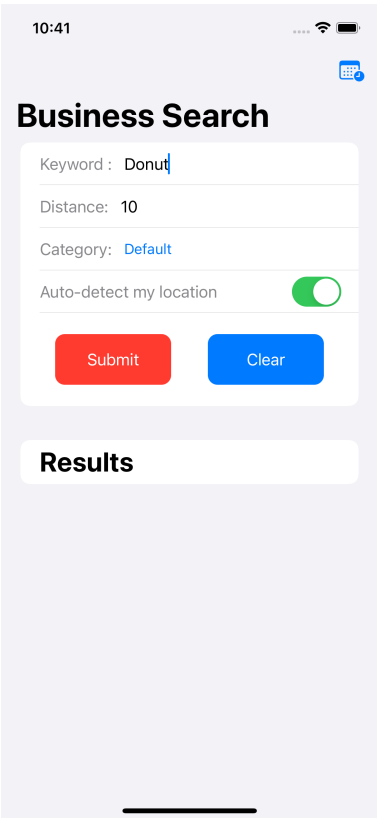
Category picker menu



Location field disappers when Auto-Detect is ON



“Submit” button DISABLED

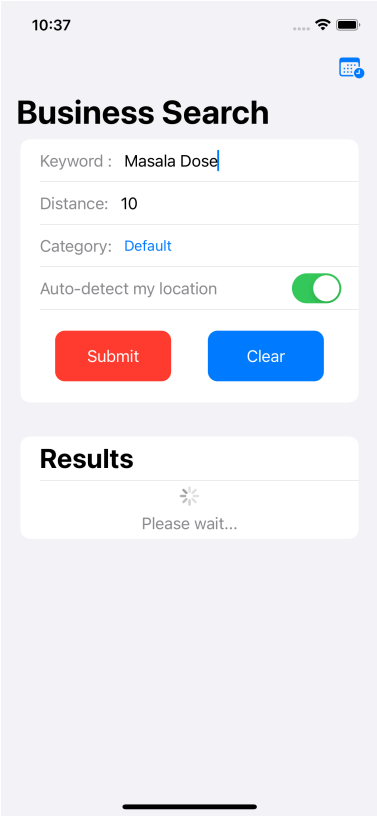


“Submit” button ENABLED

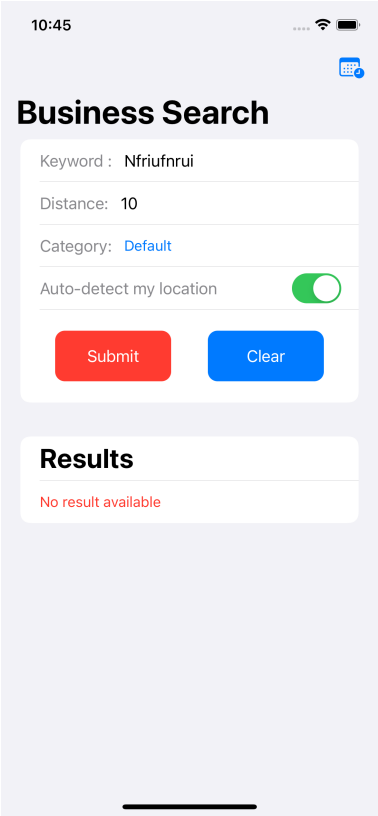
5.2.2 Results Section

Once the user hits the submit button in the Business Search section, the outcome of the search must be displayed in the results section.

A ProgressView with the message “Please wait...” must be displayed while waiting for the backend APIs to respond to the search request. If the backend API does not return any result, then “No result available” must be shown in the Results Section.



ProgressView



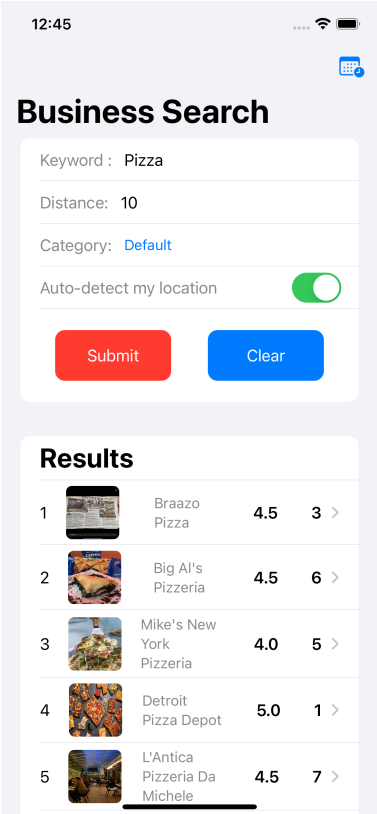
No Result Available

Once the API returns the search results, the following must be displayed:

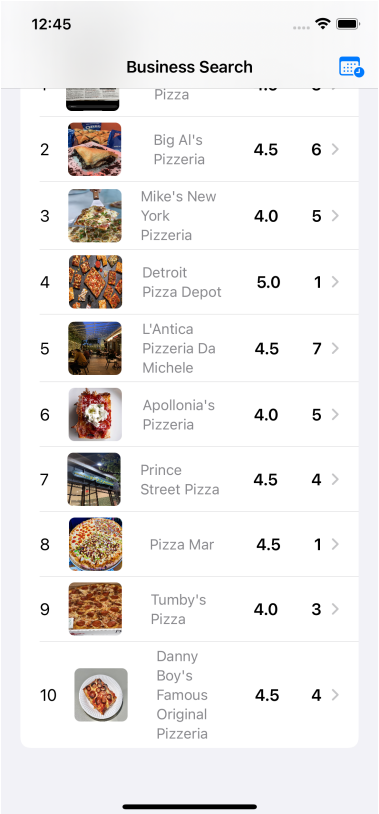
1. Icon: Use Kingfisher library to download and cache the images. Please refer to hints section for more details on Kingfisher library. The icons should maintain the aspect ratio and have a rounded corner.
2. Name: Name of the business
3. Rating
4. Distance

Each search result has an arrow on the right side of the Distance field. This is automatically added by SwiftUI if you embed your ListView inside a NavigationView. When a list item is clicked, the **Business Detail screen** will open for the selected business.

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.**



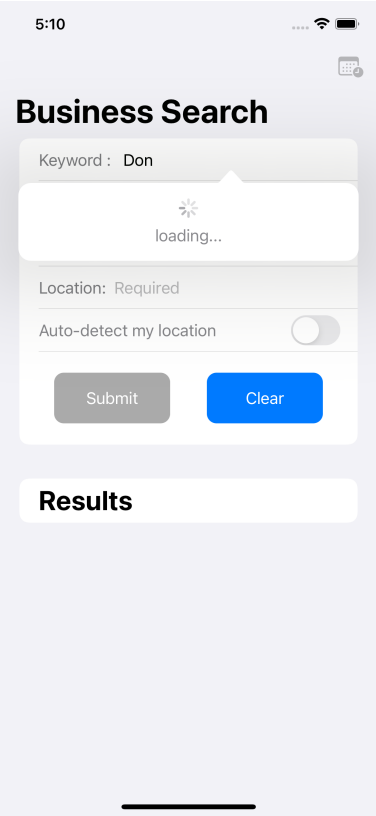
Results Section



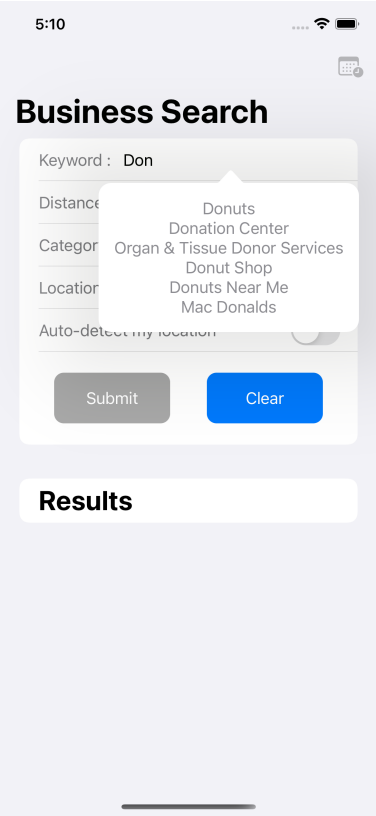
Results Section

5.3 Keyword Search Functionality

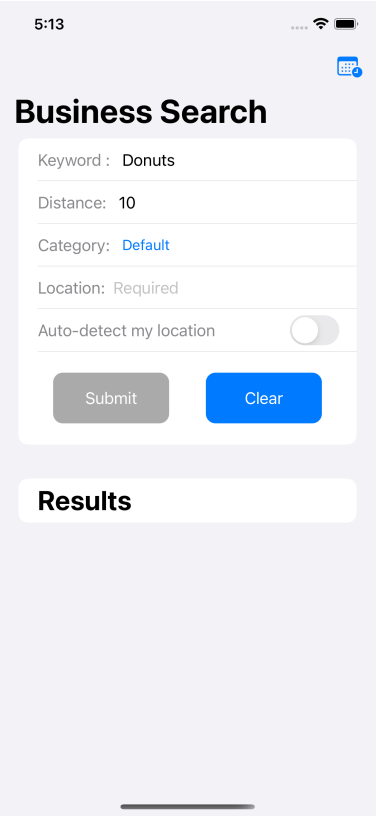
- Users can type in a keyword and hit the enter key.
- This will make an API call to the Node.js backend and fetch the results which are displayed in a popover. Please see the hints section for more on popover.
- A ProgressView must be added to the popover while waiting for the results from the API call.
- Once the user clicks on any of the results displayed from the autocomplete API, the selected business must be displayed in the “Keyword” field.
- Using the starter code given in the hints section for popover WILL NOT result in any penalty.



ProgressView after user hits the enter key



Autocomplete results displayed in the popover



User selection shown in keyword field

5.4 TabView

- Use the TabView to show 3 tabs with the following SF symbols to represent each of the tab labels:

    I. Business Detail: “text.bubble.fill”
    II. Map Location: “location.fill”
    III. Reviews: “message.fill”

5.4.1 Business Details Screen

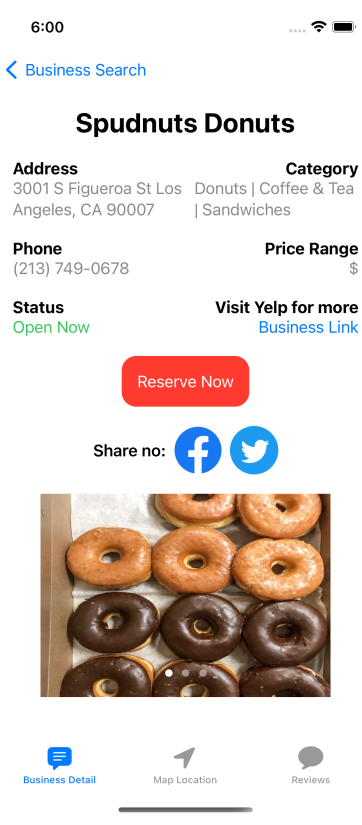
Clicking on any of the Business Search results in the Home Screen should navigate the user to the Business Details Screen.

The NavigationBar should have Business Search 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 Business Search fields populated with values used to fetch the current Business Details Screen). The state of the Home Screen must be preserved.

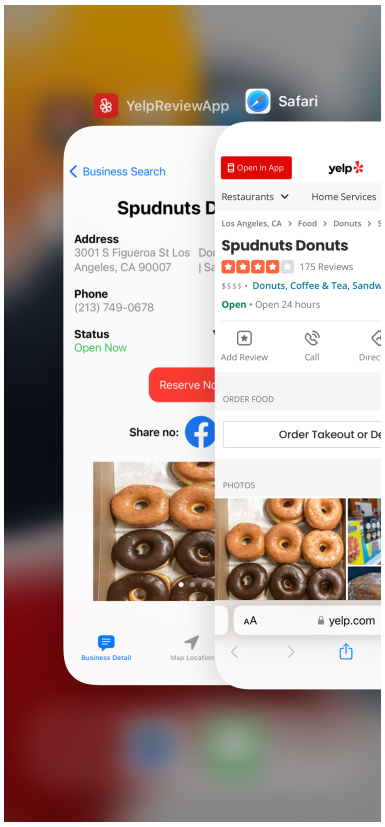
Below the NavigationBar, there should be:

- Name of the business
- Address of the Business
- Category
- Phone
- Price Range
- Status: This should have the value:

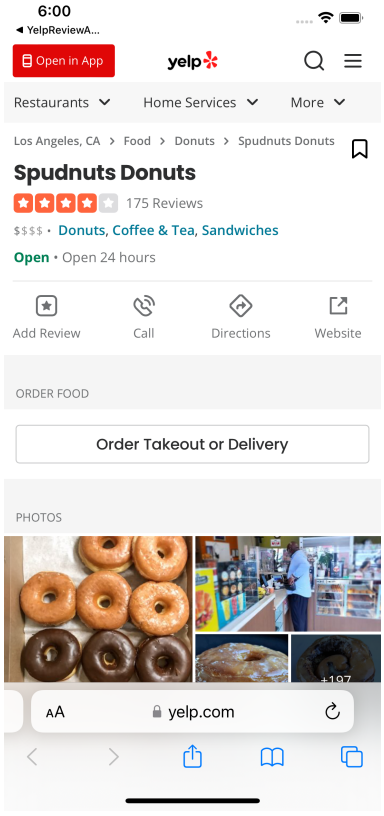
  “Open Now” in GREEN if the business is open at the time of the search.
  “Closed” in RED if the business is closed at the time of the search.
- Business Link: This link should open in Safari browser when clicked.



Business Details Screen



User clicks on Business Link



Business Link opens in Safari

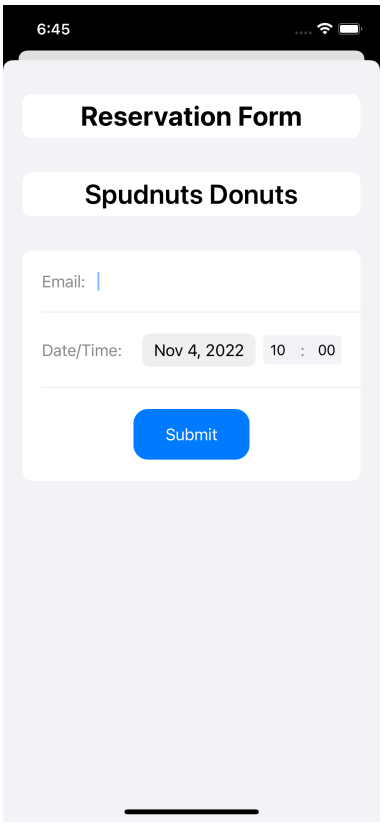
There is a “Reserve Now” button in red. Clicking on this button opens a reservation sheet. Please refer to hints section to know more about sheets. The sheet is titled “Reservation Form” and should display the name of the business as well. The user must be able to make a reservation at the business by entering his/her email address.

- E-mail addresses must be **validated**.
- User should be able to pick date and time of the reservation.
- User should be able to pick a date using “DatePicker”. Only the date from the **present day to the future** must be shown. The dates in the past must be greyed out and the user SHOULD NOT be able to select it.

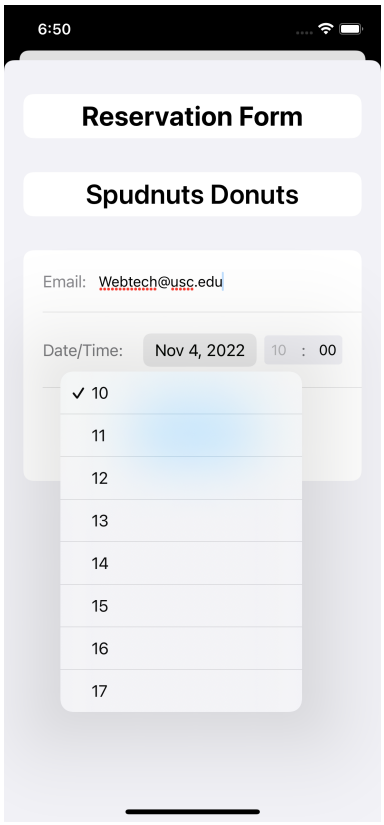
- User should be able to select a time only between 10am-5pm for the reservation. Limit the minutes selection to multiples of 15 as shown in the figure.
- When the user hits “Submit” button:

    I. A toast message should be displayed to the user who does not enter a valid e-mail address. Read more about toasts in the hints section.

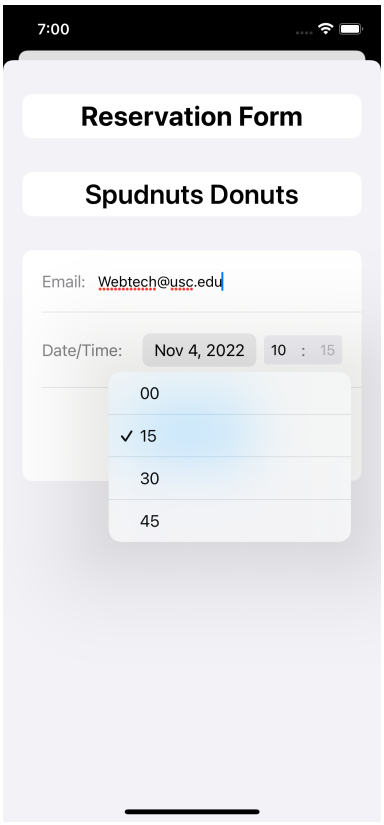
    II. If the user has a valid e-mail address, then a reservation confirmation sheet should be shown to the user. The name of the business must be included in the confirmation sheet as shown in the figure. The confirmation sheet should automatically close after 2 seconds or when the user clicks on the “Done” button at the bottom of the sheet. The reservation details should be stored in Local storage. Look at the hints section for more details on how to use UserDefaults.



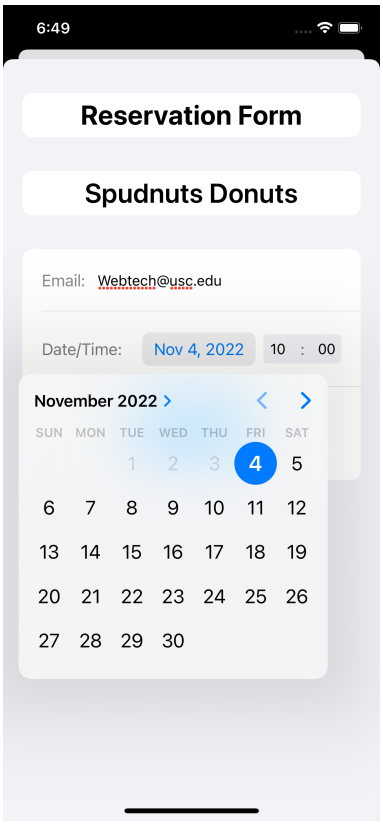
Reservation Sheet



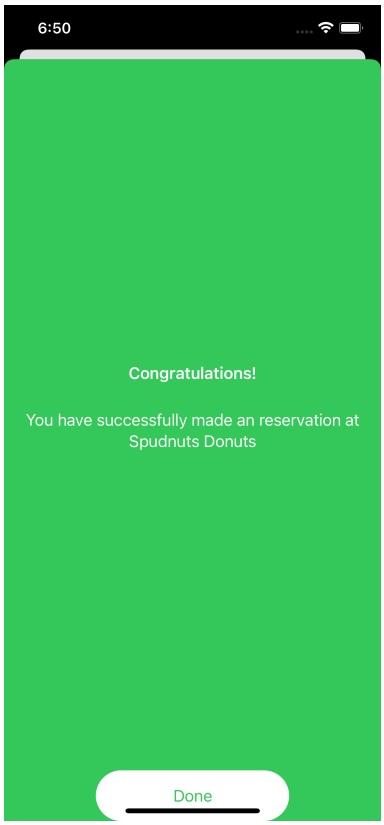
Hour picker



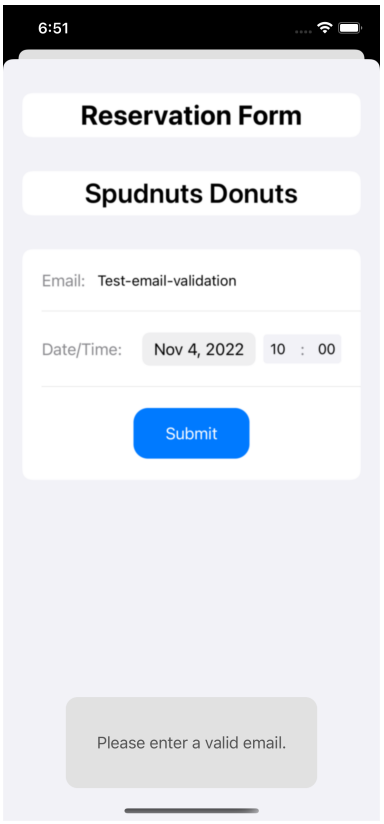
Minutes picker



Date picker

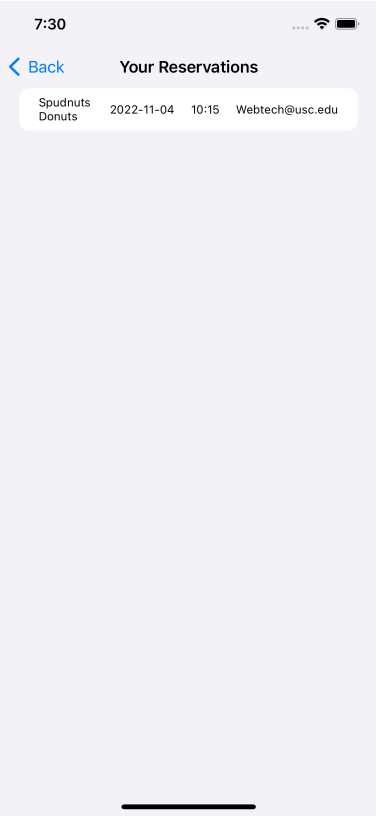


Reservation Confirmation

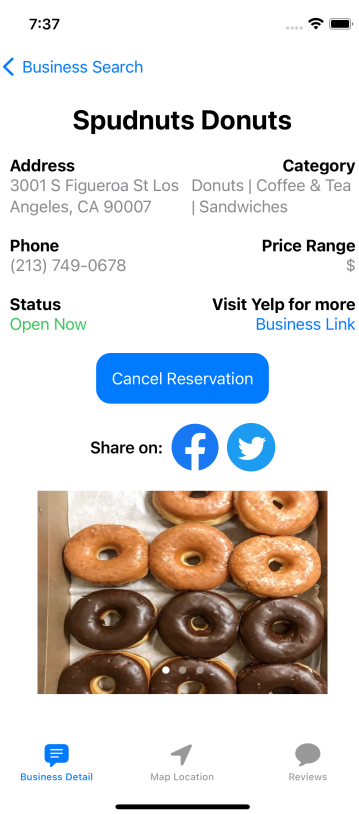


Invalid e-mail Toast

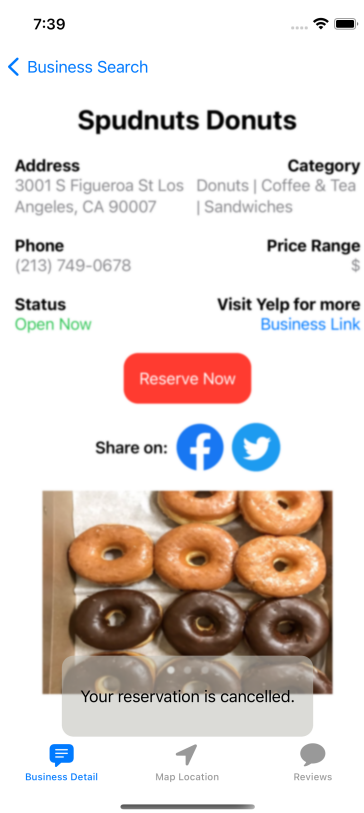
- Once a reservation has been made at a business, the “Reserve Now” button must change to “Cancel Reservation”.
- If the user clicks on “Cancel Reservation”, an appropriate toast message should be displayed saying the reservation has been cancelled and the “Reserve Now” button should be shown again.



Reservations section showing the reservation just made by the user



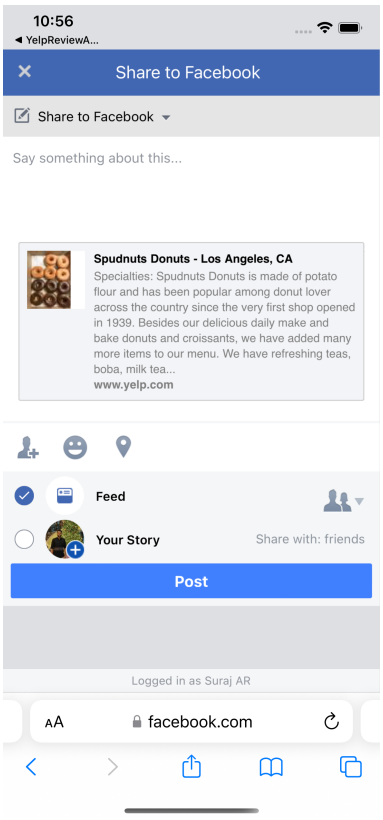
Cancel Reservation



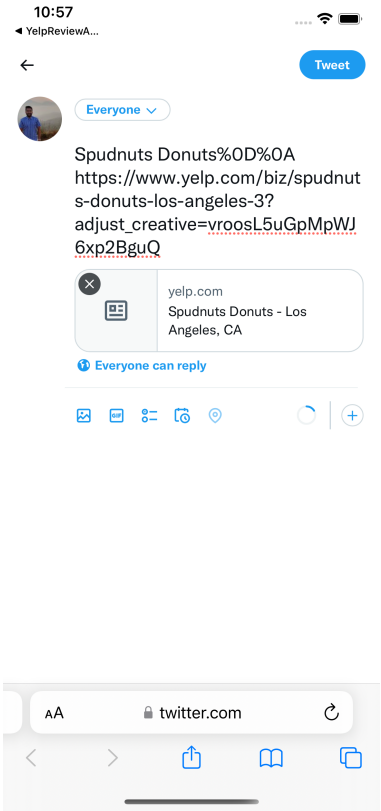
Toast after cancellation

5.4.1.1 Business Sharing

- **Facebook Share button**: Clicking on this button should share name and URL of the business as a post on Facebook. Facebook should be opened on Safari browser.
- **Twitter Share button**: Clicking on this button should share the name and URL of the business as a tweet. Twitter should be opened on Safari browser.



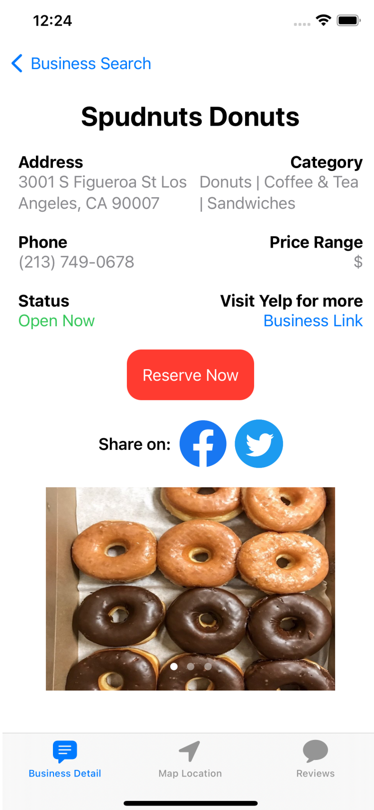
Facebook Share

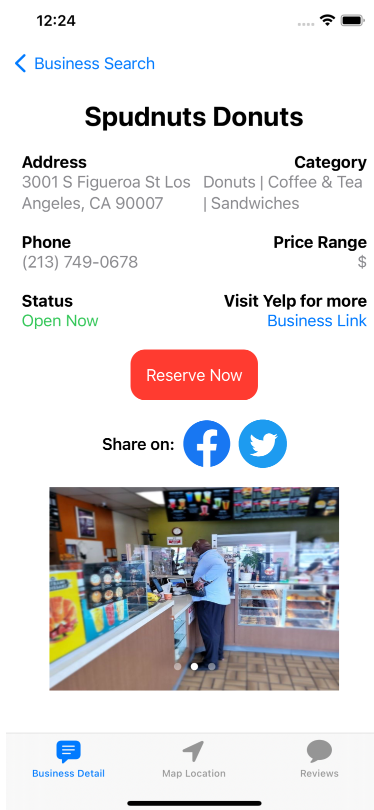


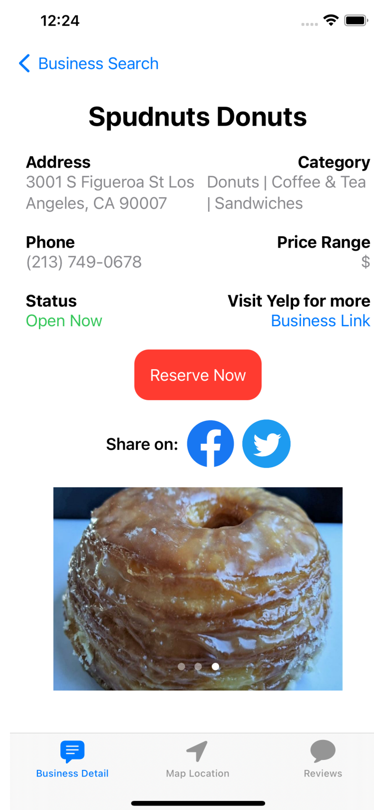
Twitter Share

5.4.1.2 Business Images

- Use a TabView to display 3 Business Images. Use KingFisher library to download and cache images. Refer to the hints section for more on KingFisher library.

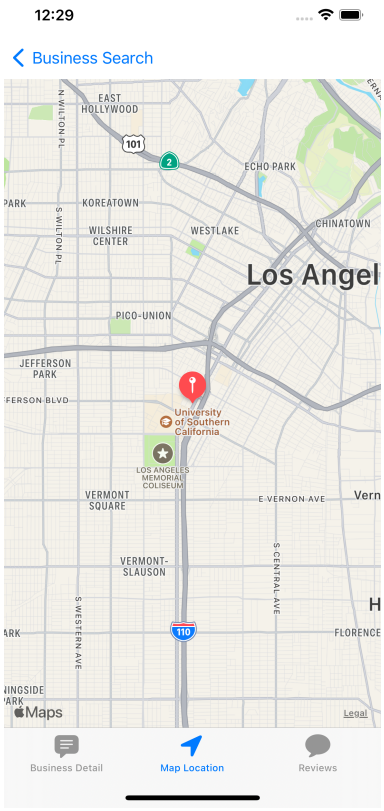






5.4.2 Map Location Screen

- Use Apple Maps and drop a pin at the location where the business is located.

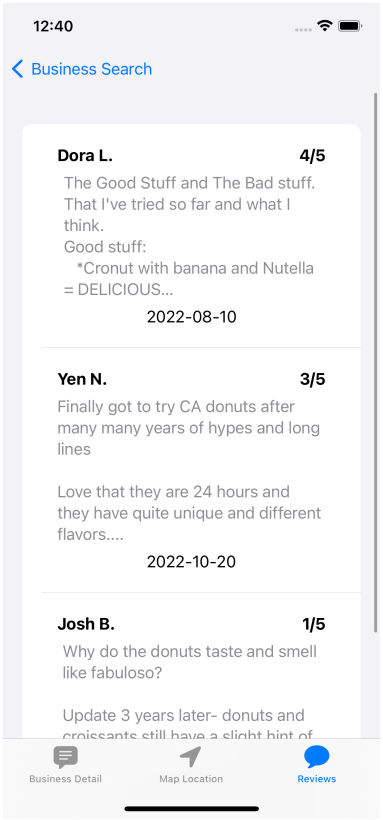


Pin Dropped at business location

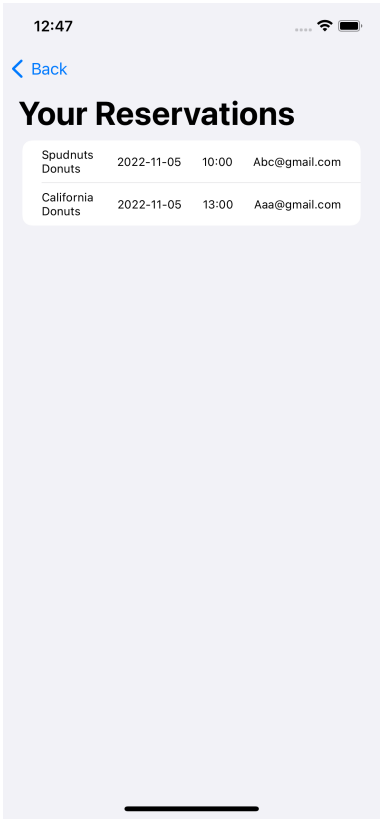
5.4.3 Reviews Section

- A ProgressView should be added with a Circular Progress spinner while the business review data is being fetched.
- Create a list of 3 reviews.
- For each review, the following aspects must be included:

    I. The reviewer's name
    II. The rating given by the reviewer for that particular business
    III. The review itself
    IV. Date when the review was given



Business Reviews Tab



Reservations Screen

5.5 Reservations Section

- All reservations made by a user must be shown in the reservations section which can be accessed by clicking on the calender icon present on the NavBar.
- Each reservation should contain Name of the Business, the date when the reservation was made, the time of the reservation and the e-mail ID of the user who made the reservation as shown in the picture above.
- Use Local Storage (Refer to Implementation Hints) to store the list of reservations made.
- You have to implement a “Swipe to delete” functionality for each reservation. When a reservation is swiped left, the reservation must be deleted and not shown in the reservation screen anymore.

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, Business Detail screen** and before fetching the reviews. 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 https://developer.apple.com/documentation/swift/optional

<u>Note:</u> Based on your implementation, you might need to put progress bars on different places.

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 the styling matches the video/screenshots.
5. All API calls are to be made using Node.js backend, like what you have developed for homework 8.

5.8 Additional Info

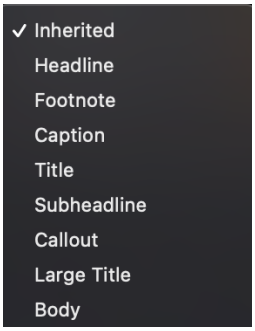
For things not specified in the document, grading guideline, the video, or in Piazza, you can make your own decisions. But keep in mind about the following points:

- 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 (Refer to Hints) 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..



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, when the business search results are embedded in a NavigationLink and the user clicks on a search result, Swift provides an animation with a left sliding behaviour to transition into the Business Details screen.

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:
https://developer.apple.com/videos/play/wwdc2020/10119/

SwiftUI Tutorials: https://developer.apple.com/tutorials/swiftui

SwiftUI MVVM Programming with ObservableObject @Published @ObservedObject:
https://www.youtube.com/watch?v=1IlUBHvgY8Q

View Modifiers: https://developer.apple.com/documentation/swiftui/viewmodifier

SwiftUI cheat sheet: https://jinxiansen.github.io/SwiftUI/

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

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: https://stackoverflow.com/questions/31661575/how-to-create-objects-from-swiftyjson second answer.

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

https://www.avanderlee.com/xcode/launch-screen/

6.8 Working with NavigationBar and App Navigation

Adding a navigation bar - a free Hacking with iOS: SwiftUI Edition tutorial:
https://www.hackingwithswift.com/books/ios-swiftui/adding-a-navigation-bar

The Complete Guide to NavigationView in SwiftUI:
https://www.youtube.com/watch?v=nA6Jo6YnL9g

6.9 Working with TabView

Adding TabView and tabItem(): https://www.hackingwithswift.com/quick-start/swiftui/adding-tabview-and-tabitem

TabView: https://developer.apple.com/documentation/swiftui/tabview

6.10 Local Storage

You should save the user reservation information to local storage. When the app is opened, you should load them into the Reservation Section of the App.

The old way of dealing with small amounts of data storage is using UserDefaults: https://learnappmaking.com/userdefaults-swift-setting-getting-data-how-to/ . In SwiftUI, a new @AppStorage: https://developer.apple.com/documentation/swiftui/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 https://stackoverflow.com/questions/29215825/how-to-store-array-list-of-custom-objects-to-nsuserdefaults teach you how to do that, do your searches.

6.11 Adding Toasts

SwiftUI: Global Overlay That Can Be Triggered From Any View:
https://stackoverflow.com/questions/56550135/swiftui-global-overlay-that-can-be-triggered-from-any-view second answer

6.12 Popover

SwiftUI Popover implementation details can be found here:
https://pspdfkit.com/blog/2022/presenting-popovers-on-iphone-with-swiftui/

6.13 Open Link in browser

How to open web links in Safari - a free SwiftUI by Example tutorial:
https://www.hackingwithswift.com/quick-start/swiftui/how-to-open-web-links-in-safari

6.14 Image related

How to disable the overlay color for images inside Button and NavigationLink:
https://www.hackingwithswift.com/quick-start/swiftui/how-to-disable-the-overlay-color-for-images-inside-button-and-navigationlink

SwiftUI Image clipped to shape has transparent padding in context menu:
https://stackoverflow.com/questions/62687224/swiftui-image-clipped-to-shape-has-transparent-padding-in-context-menu

6.15 Adding a Sheet for Reservations

How to present a new view using sheets - a free SwiftUI by Example tutorial:
https://www.hackingwithswift.com/quick-start/swiftui/how-to-present-a-new-view-using-sheets

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 to the DEN D2L Dropbox folder.

You will also have to submit a video of your assignment demo to the same DEN D2L Dropbox folder. You must demo your submission using Zoom. Details for that how to create the video are on DEN D2L.

Demo is done on a MacBook using the simulator, and not on a physical mobile device.