
# Homework 9
# Event Finder App

Prof. Marco Papa

Developed and Designed by
Dr. Marco Papa, Megha Bagri and Prudhvi Raj

# Homework 9: Event Search iOS App

## 1. Objectives

- Become familiar with Swift language, Xcode and iOS App development.
- Practice the Model-View-Controller design pattern.
- Build a good-looking IOS app.
- Learn to use Google Maps APIs and iOS SDK
- Manage and use third-party libraries by CocoaPods

## 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 and iOS. First released in 2003, the latest stable release is version 14.2 and is available via the Mac App Store free of charge.

Features:

- Swift 5 support
- Playgrounds
- Interface Builder
- 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 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 at 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/

The complete guide to SF Symbols is available at:

https://www.hackingwithswift.com/articles/237/complete-guide-to-sf-symbols

### 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/

# 3. Prerequisites

This homework requires the use of the following components:

### 3.1 Download and install the latest version of Xcode

To develop iOS apps using the latest technologies described in these lessons, you need a Mac computer (macOS Sierra 10.12.4 or later) running the latest version of Xcode. Xcode includes all the features you need to design, develop, and debug an app. Xcode also contains the iOS SDK, which extends Xcode to include the tools, compilers, and frameworks you need specifically for iOS development.

Download the latest version of Xcode on your Mac free from the App Store.

To download the latest version of Xcode

- Open the App Store app on your Mac (by default it's in the Dock).
- In the search field in the top-right corner, type Xcode and press the Return key.
- The Xcode app shows up as the first search result.
- Click Get and then click Install App.
- Enter your Apple ID and password when prompted.
- Xcode is downloaded into your /Applications directory.

You may use any other IDE other than Xcode, but you will be on your own if problems come up.

### 3.2 Add your account to Xcode

When you add your Apple ID to the Xcode Accounts preferences, Xcode displays all the teams you belong to. Xcode also shows your role on the team and details about your signing identities and provisioning profiles that you'll create later in this document. If you don't belong to the Apple Developer Program, a personal team appears.

Here is detailed documentation:

**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, failing to compile or run SwiftUI code, your best option is to install the latest MacOS.

**IMPORTANT**

This homework is developed with SwiftUI. 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 14.2. Xcode 14.2 provides the crucial functionalities you will need to develop SwiftUI apps. You can download Xcode14.2 by searching "Xcode" in your mac's app store.

- Download and install SF Symbols 4, 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!

*Note that with newer versions of XCode we use the Swift Package Manager. CocoaPods is only used as a dependency manager for Swift and Objective-C Cocoa projects.*

## 4. High Level Design

This homework is a mobile app version of Homework 8. In this exercise, you will develop an iOS Mobile application, which allows users to search for event information, ticketing information, save events as favorites, and post on Twitter. You should reuse the backend service (node.js script) you developed in HW8 and follow the same API call requirements.

The main scene of this app is like that in Figure 1. All the implementation details and requirements will be explained in the following sections.

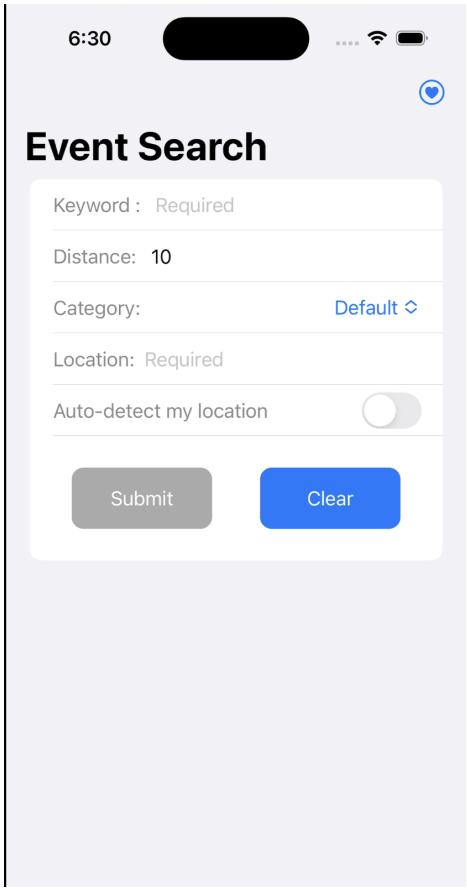


**Figure 1. The Event Search App**

## 5. Implementation

### 5.1 Search Form

You must replicate the Search Form, as shown in Figure 1. Note that all the components in the search form are the same as assignment 8 (validations and defaults etc.)

The interface consists of the following:

- **Keyword**: It provides the autocomplete function as shown in Figure 3. Make sure you use the same API as Homework 8.
  **NOTE:** You need to use a debouncer to avoid unnecessary API hits while searching keywords.
- **Category**: When the user taps on this field, a dropdown view should display for selecting a category, as shown in Figure 2. Make sure you include all the categories in homework 8.
- **Distance**: To enter the distance (in miles).
- **Location**: To enter the location.

- **Auto-detect my location**: There is a toggle to Auto-detect my location. When toggled the Location section collapses.
- A **SUBMIT button** to get the input information of each field, after validation. If the validation is successful, then the events would be fetched from the server. However, if the validations are unsuccessful, the submit button should remain disabled. and no further requests would be made to the server.
- A **CLEAR button** to clear the input fields and set them to default values if applicable. It should remove all error messages (if present). Also in case the user has some search results already, they should also disappear upon clearing.

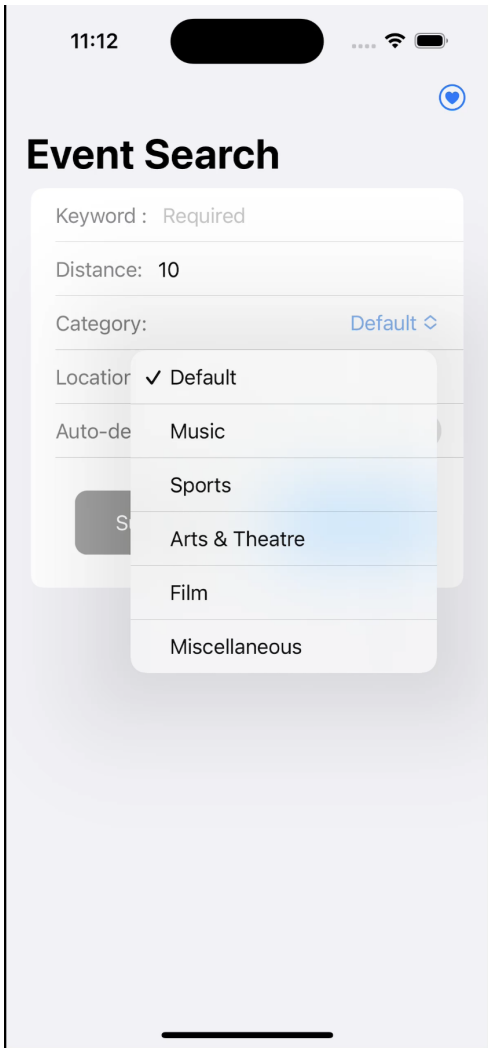


**Figure 2: Choose category from Drop down**

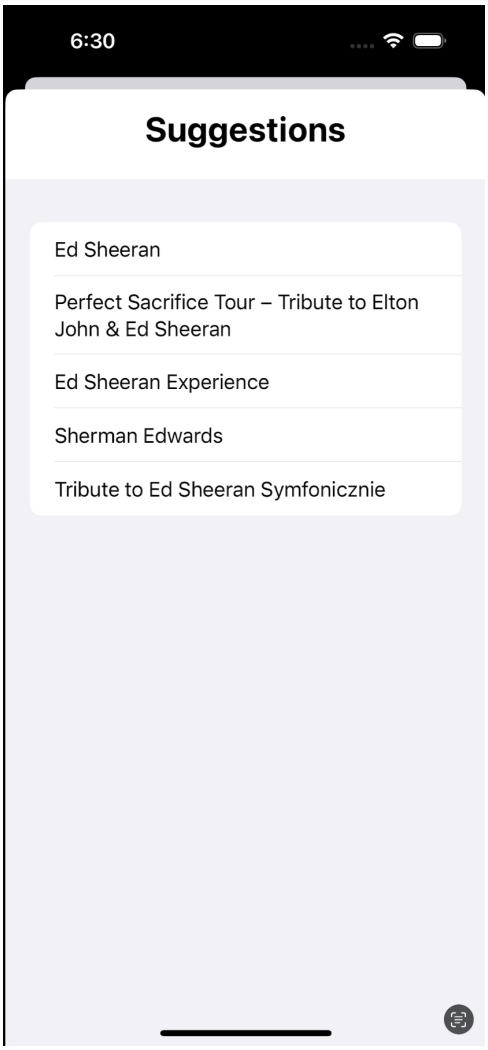


**Figure 3: Autocomplete for keywords**

The validation for empty keyword and empty location needs to be implemented. If the user does not enter anything in the 'TextField' or just enters some empty spaces, the submit button will still be inactive as the input would not be considered as valid input.

Once the validation is successful, you should execute an HTTP request to the Node.js script located in the GAE/AWS/Azure, which you have finished in Homework 8, and then display the search results just below the Search Form.

### 5.2 Search Results

When the user taps the "SUBMIT" button, your app should display a spinner before it's ready to show the results sections, as shown in Figure 4. Then after it gets data from your backend, hide the spinner and display the results in the results section as a list, as shown in Figure 5.

Each of the list item should have the following:

- Category image
- Name of the event
- Name of the venue
- Data and time of the event

**See homework 8 for more details about these fields.**

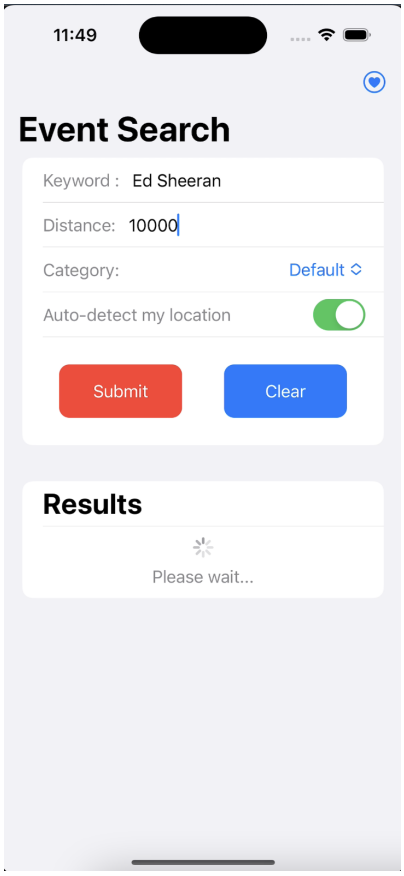


Figure 4: Display a spinner while searching

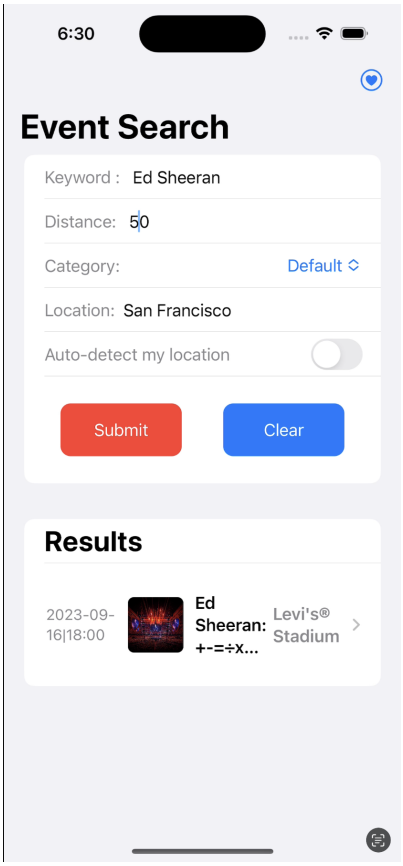


Figure 5: List of search results

### 5.3 Event Details

Tapping on a NavigationLink in the results table should show details of that event with three tabs: Events, Artist/Team, Venue. Note that a spinner should be shown before you are ready to display information in each tab as in Figure 6.

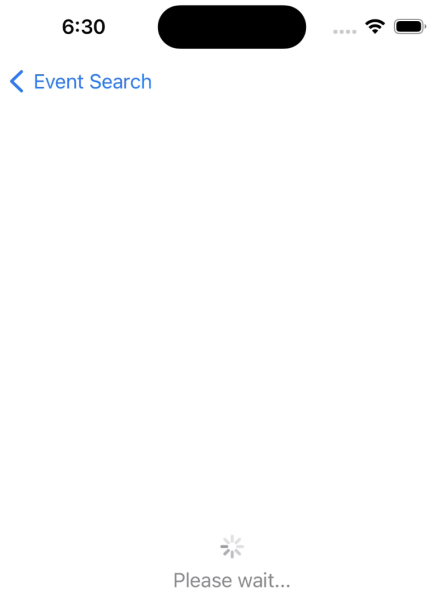


**Figure 6: Display a spinner while search happens.**

All the three tabs share the same navigation bar on the top. The navigation bar should include the following elements:

- **Back button** which navigates back to the search result list

NOTE: All the texts in the app should be a scrolling text i.e. whenever a text is encountered which does not fit in the given space (e.g. Event Name or Artists/Team section) it should scroll automatically.

### 5.3.1 Event Info Tab

Show the fields listed in Figure 7 in the Events tab: Artist, Title, Artist/Team, Venue, Genre, Price Range, Ticket Status, Ticketmaster Ticket Link and Seat Map. You can use Kingfisher.KFImage to show the images.To learn more about Kingfisher, navigate to the Hints section 6.5.3. Clicking on the ticketmaster URL will open the Ticketmaster URL in a Safari window. **See homework 8 for more details about each field.**

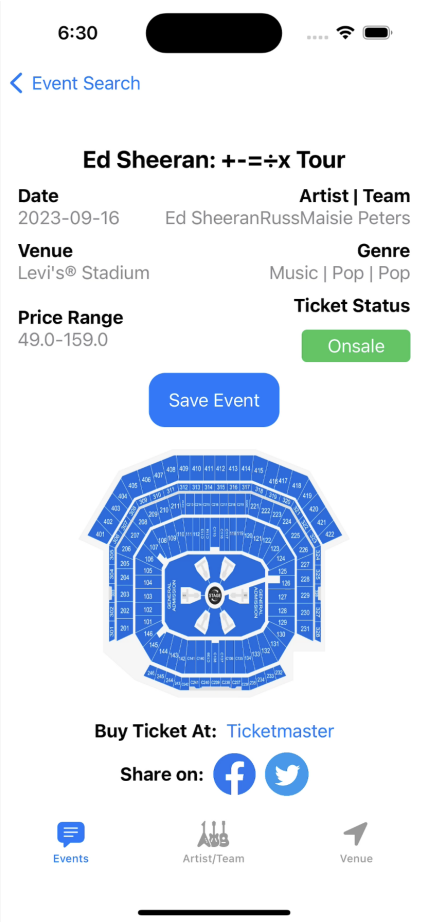


**Figure 7: Event details and event tab**

There is a save event button which adds it to the favorites. If it is already added to favorites, display 'Remove from favorites'. See Figure 8. Also display appropriate messages below when added/removed from the favorites.

To learn more about how to make a toast in SwiftUI follow this link: https://stackoverflow.com/questions/56550135/swiftui-global-overlay-that-can-be-triggered-from-any-view

Feel comfortable to use anything else as well.

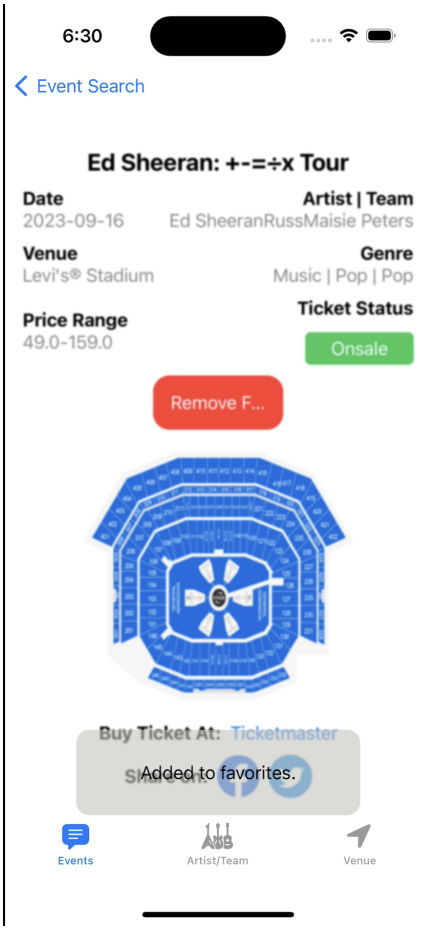


**Figure 8: Save event is clicked**

You also have a Share section (Twitter icon and Facebook Icon) to share the event details on Twitter/Facebook. Once the button is tapped, respective web pages should be opened to allow the user to share the event information on Twitter, as shown in Figure 9a and Figure 9b. See homework 8 for how it works and the format of the shared content.

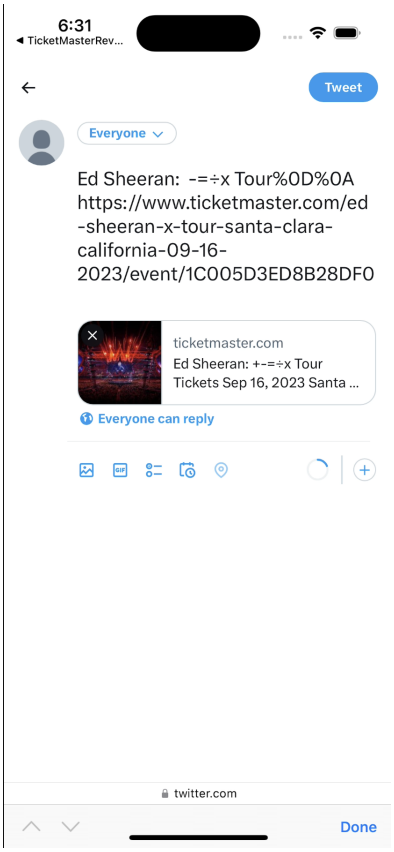


**Figure 9a: Share event on twitter**

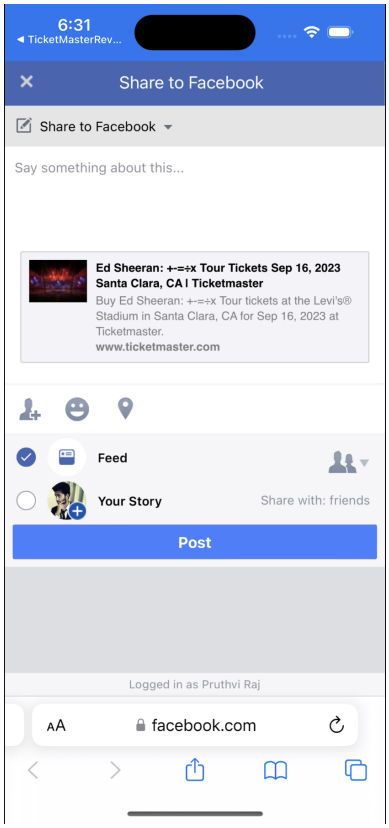


**Figure 9b: Share event on Facebook**

### 5.3.2 Artist(s) Tab

Same as in homework 8. For each artist show

- Artist Name
- Followers - followers count should be displayed in M(Millions) or K(Thousands) depending on the value
- Popularity – with a red progress ring around the popularity number
- Link to the Artist's Spotify page
- Top 3 popular albums' cover

For the events that are not related to music category, the Artist tab will not display artist info. Rather it should display a message "Artist/Music data unavailable.". Figure 10, 11 and 12

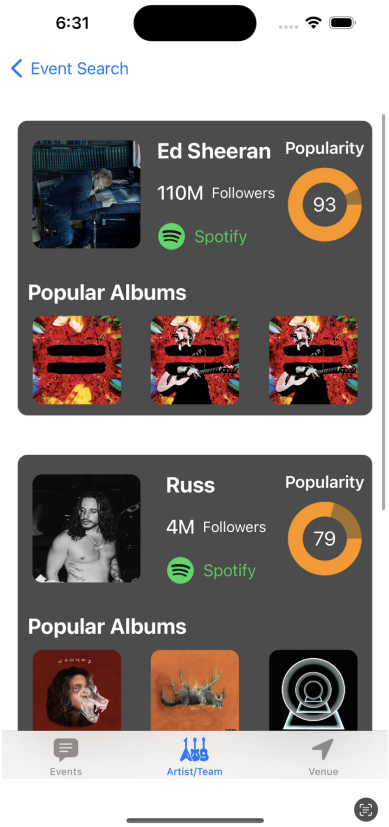


**Figure 10:** Artists Tab with Music Artist

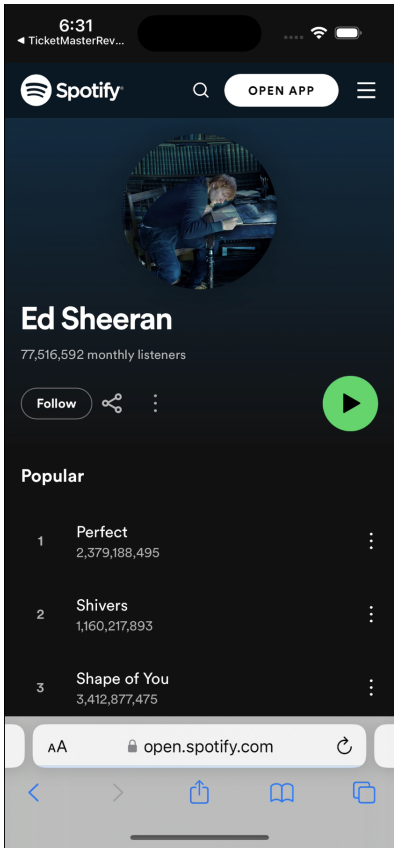


**Figure 11:** Artist's Spotify page

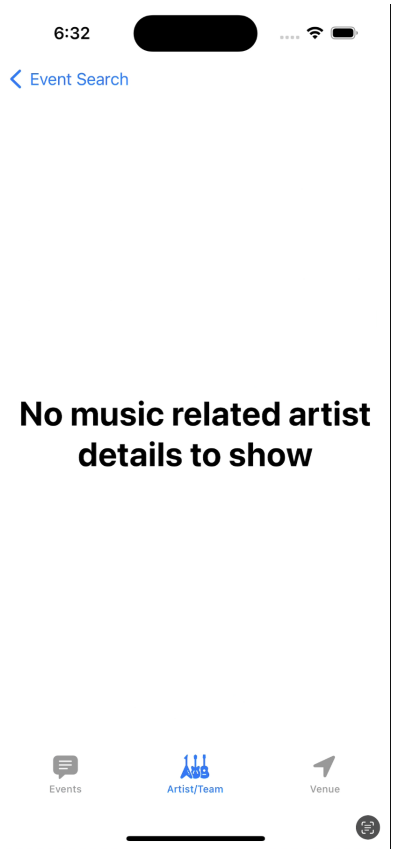


**Figure 12:** Artists/Teams tab for non-music event

### 5.3.3 Venue Tab

As shown in Figure 13, 14 and 15:

- Same as homework 8 with below mentioned fields.
    - Name
    - Address
    - Phone No
    - Open Hours
    - General Rule
    - Child Rule

This view should be scrollable since the details of the venue table may be too long.

You should have a ‘Show venue on maps’ button which should open Apple Maps and display the location.

To learn more about using Apple maps, follow this link:
https://kristaps.me/blog/swiftui-mapview/

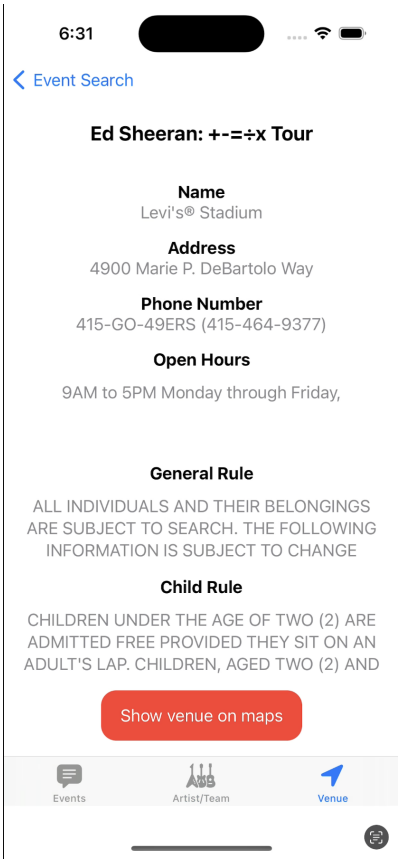


**Figure 13: Venue Tab**

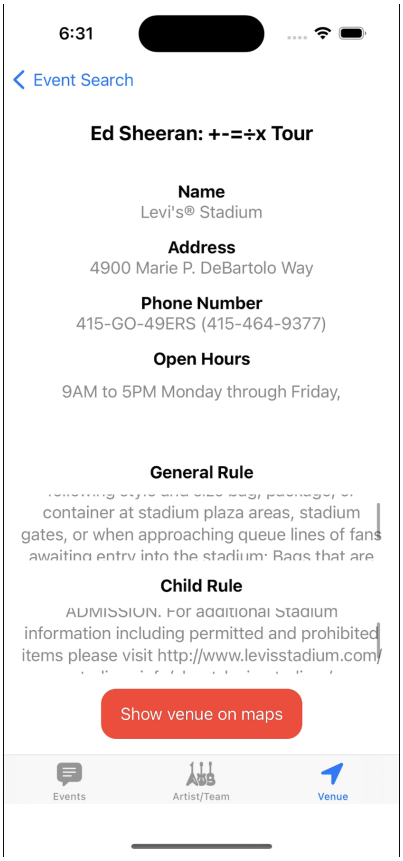


**Figure 14: Venue Fields Scrollable**

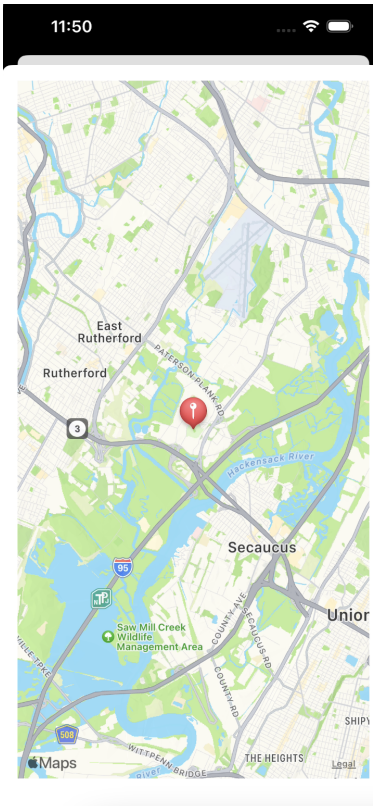


**Figure 15: Venue Map**

### 5.4 Favorites list

Use a control button in the main screen to switch between the search page and the favorites page. The favorite events should be displayed in a list. Each of the list components has the exact same structure as the search results cell, but has **NO favorite button** as shown in Figure 16. If there are no favorites, a “No Favorites” should be displayed at the center of the screen, as shown in Figure 17.

Tapping on a cell should have the same behavior as tapping on a cell in the search results page.

The user should also be able to remove an event by right-swiping a cell, and clicking the delete option as shown in Figure 18. Upon deletion from here, the favorite should be removed from UserDefaults and the Button on the details page of that event should be 'Save Event'.

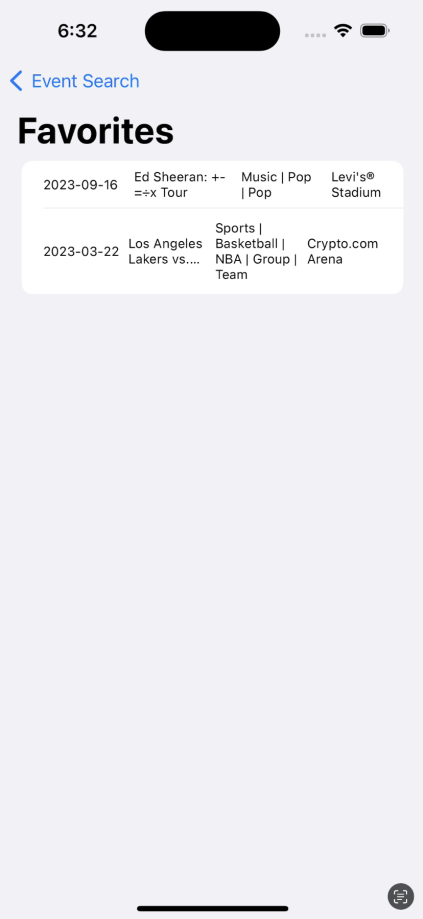


**Figure 16: Favorite list**

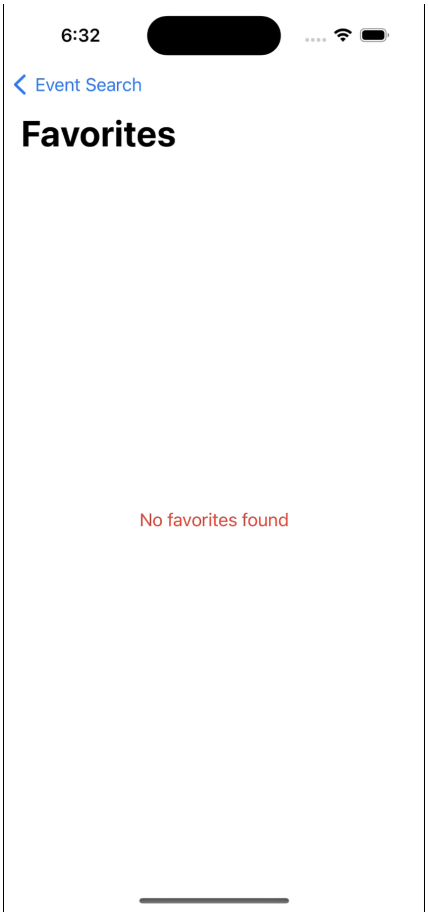


**Figure 17: No Favorites**

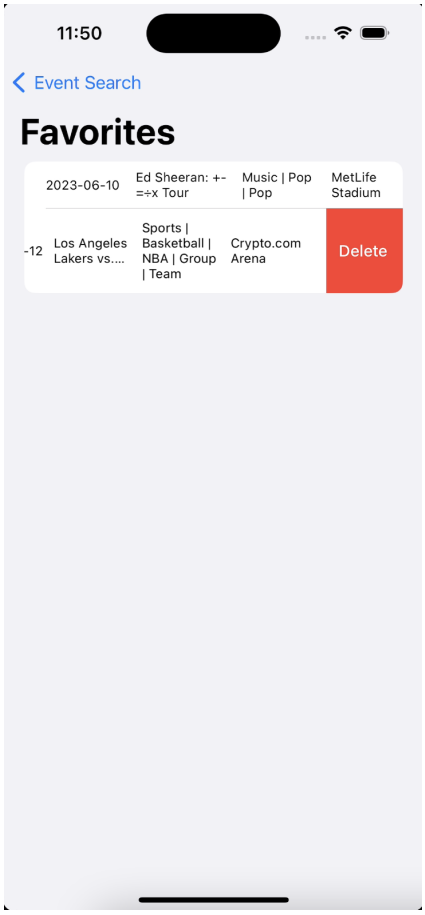


**Figure 18: Swipe to remove an event**

## 5.5 Error handling

If no events are found for a given keyword, a “no results” should be displayed, as shown in Figure 19.

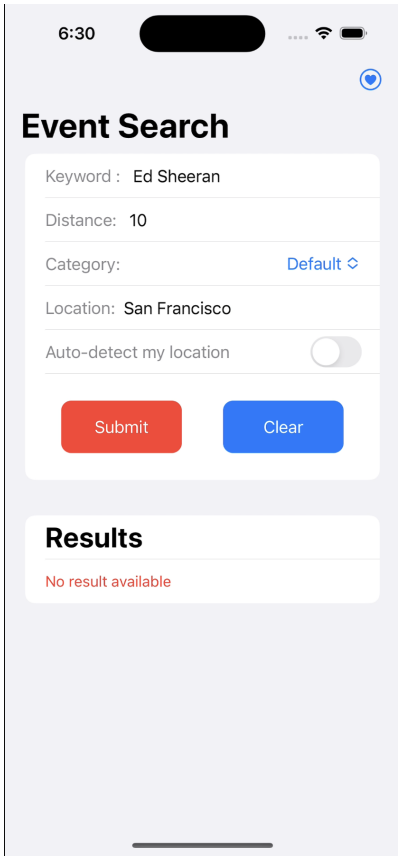


**Figure 19: No result available page**

If for any reason (no network, API failure, cannot get location etc.) an error occurs, appropriate error messages should be displayed at the bottom of the screen.

### 5.6 Additional

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 occurs.
- You can only make HTTP requests to your backend (Node.js script on AWS/GAE/Azure) or use Apple Map SDK for iOS.
- All HTTP requests should be asynchronous.

**NOTE:** Alamofire makes all its API calls asynchronously and if students are using Alamofire they don't have to worry about it

## 6. Implementation Hints

### 6.1 Images

The images used in this homework are available in the zip file in the D2L dropbox folder and will be linked in Piazza post for HW 9. The videos also will be uploaded on D2L and the youtube video linked on Piazza post.

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.

### 6.2 Get current location and favorites storage

Use ipInfo to get the current location. You should make a call to ipInfo similar to how it was done in assignment 8. Since sometimes it fails to get the current location in the iPhone simulator, you might have to set a custom location by setting "Debug – Location – custom location…" in the simulator.

Use "UserDefaults" to store favorite data.

### 6.3 Good Starting Point

There will be a Piazza with links to great tutorials, here are 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/

### 6.4 Third party libraries

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

#### 6.4.1 Alamofire

Alamofire is an HTTP networking library written in Swift.

https://github.com/Alamofire/Alamofire

#### 6.4.2 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

### 6.5 Displaying ProgressView

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

### 6.6 Implementing Splash Screen

https://kristaps.me/blog/swiftui-launch-screen/

### 6.7 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

### 6.8 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

### 6.9 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.

**6.10 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.

**6.11 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

**6.12 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

**6.13 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 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.**

****IMPORTANT**:**

All videos are part of the homework description. All discussions and explanations in 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.