CSE 438: Mobile Application Development

Lab 4: Movie Search App

Overview

In this lab, you will be implementing common iOS view structures to create something that feels like a "real app". Navigation bars, tab bars, and collection views are built-in ways to structure your app's content. As for the content itself, you will be hitting a remote API for JSON data to parse into your application. This is a common structure which is used by many popular apps on the App Store.

Please **read this entire PDF** prior to starting work on your lab. Every section contains information relevant to how you will be graded, not just the Requirements section.

Please test opening your zip file on another computer to make sure the files are included properly. TA's are not liable for broken submissions and the late penalty will apply.

Details

Due date: Wednesday, July 17th, 11:59pm
Grading: This lab is out of 150 points total. The exact point distribution is described in the "Requirements" section below.
Submission: Zip the entire project folder and email it to cse438ta@gmail.com. Please name the file "FirstNameLastName-Lab4.zip" and include a brief summary of your creative portion in the email body the same way you've done before.

Description

This lab requires you to create your own app from scratch, and **will take a considerable amount of time to complete**. Begin early and get help as you run into issues.

The goal of this project is to create a movie-searching app, which allows users to find information about movies. Data will be pulled from The Movie Database's (TMDb) API (http://www.themoviedb.org). Users should be able to search for movies, see the results populated in a collection view, and select a movie to view more details. They should also be able to favorite a movie, and view all of their saved favorites on a different tab.

Requirements

[10 points] The app implements a tab bar with at least two tabs: one to search for movies, and the other for their saved favorites. Each tab should have a custom image (no built-in images).

[20 points] Data is pulled from the API and populated in a collection view with multiple columns. Each search should return at least 20 movies (less if there aren't 20 results). JSON results are processed using the Codable protocol.

[10 points] Users can see the title and an image for each movie.

[10 points] Images are cached to allow for smooth scrolling. This means don't make a network request for an image every time a new cell appears on screen. Reuse the cells instead of creating new ones.

[20 points] Selecting a movie pushes a new view controller onto the navigation stack with a larger (higher resolution) image and at least 3 additional pieces of information about the movie (hint: you might have to make an additional API request to get this data).

[10 points] The user can change the search query by editing a text field.

[10 points] A spinner is shown when the data is being pulled from the API, and the request is done in the background, not locking up the user interface.

[10 points] Users can save a movie to their favorites.

[5 points] Users can delete a movie from their favorites.

[4 points] The layout is clean and easy to use.

[1 point] Properly attribute TMDb as the source of the movie data.

[10 points] Favorites are maintained between app launches (data saved locally).

[30 points] Creative portion: Add 3 additional additional features. Be creative!

Helpful Advice and Code Snippets

The Codable protocol is a great way to parse the JSON data from TMDb. Create two different structs that conform to this protocol. Examples of the two structs are listed below. Feel free to modify either to include additional information from TMD's API.

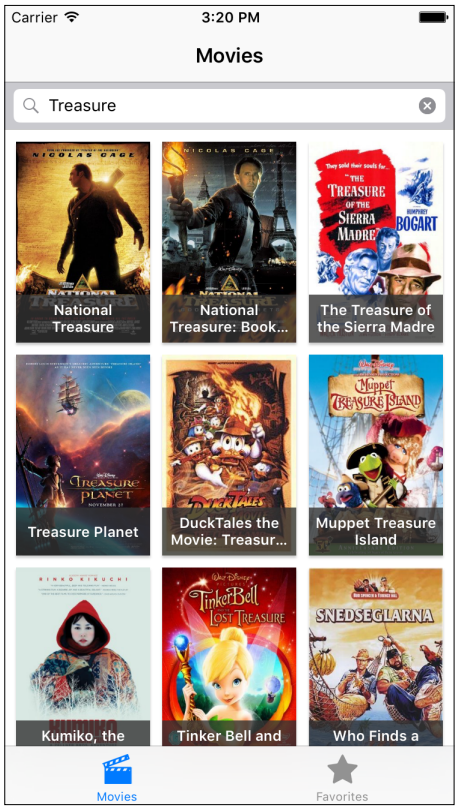
```
struct APIResults:Decodable {
    let page: Int
    let total_results: Int
    let total_pages: Int
    let results: [Movie]
}
struct Movie: Decodable {
    let id: Int!
    let poster_path: String?
    let title: String
    let release_date: String
    let vote_average: Double
    let overview: String
    let vote_count:Int!
}
```

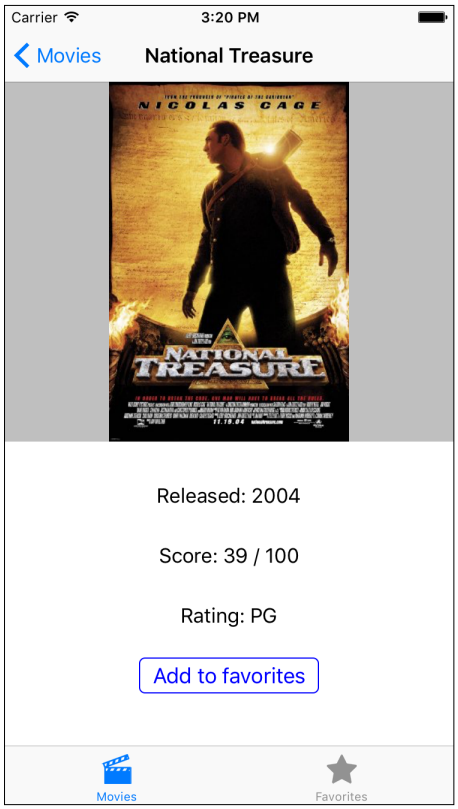
Collection views can be tricky to set up and get to work as you desire. Make sure to declare the delegate and data source, implement the protocol functions, and add the collection view to the view hierarchy. Start early and add functionality incrementally! It might be smart to test your collection view with hardcoded data before proceeding to ensure it works properly.

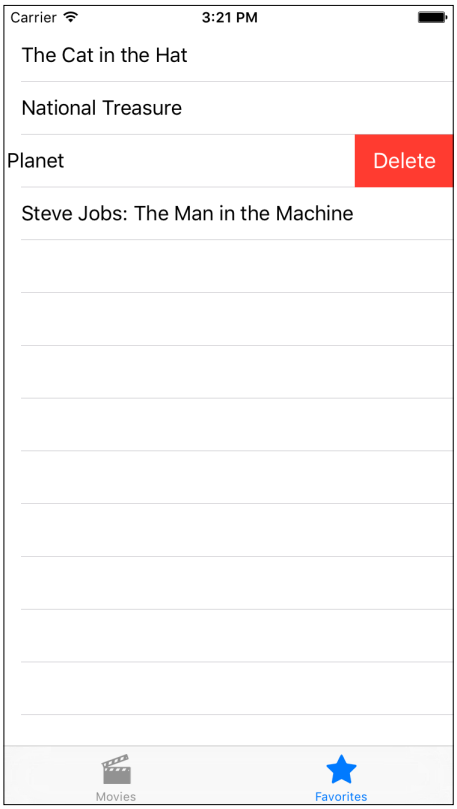
Because this app has to keep track of lots of data, one way to keep things organized is to create a struct that holds all the movie-related data. This struct could have variables that store the movie name, poster image, description, etc. You could create an array of this new type, and when the user selects a cell, simply pass the object to the new view controller so it can be populated with the correct data.

To save the user's favorite to their device, there are many options. The simplest way would be to save the movie data to UserDefaults. This has many limitations, and is only meant to store simple settings in an application. Alternative methods include plist files, a local SQLite database, or Core Data. These might be good options to explore if you are considering creating a data-heavy final project. **Make sure to test your storage method by closing the app in simulator completely (double clicking home button and swiping it away) before reopening the application.**

Also, please download and test your app on a different computer! The favorite saving functionality is a common point of failure for submissions, largely due to the new install missing the array.







Example screenshots from a working movie search app. Feel free to design the app in any way you choose that still matches the requirements.