iOS University

Models and Networking

Due: Monday, October 8th at 11:59pm

In this lab, we'll explore creating classes for our models (User, Movie, Tweet, Post, etc) and our api clients. In apps that you'll build, you'll almost certainly be working with a private API, and you may additionally leverage 3rd party APIs (Foursquare, Google, Yelp, etc). Generally, you'll have a class for each API.

For this lab, we'll pair program using either person's Movie app from Week 1. We're going to refactor this project to use a Movie model class and an api client class.

Project Setup

1. Add the Week 5 Lab - README Template to the bottom of your previous README.

Milestone 1: Create a movie model

Before we had a movie model, we used a dictionary returned by the API. In order to get the movie poster url, we had code in the view controller that looked like this:

```
let posterPath = movieDictionary["poster_path"] as? String
if let posterPath = posterPath {
    let baseUrlString = "https://image.tmdb.org/t/p/w500"
    let fullPosterPath = baseUrlString + posterPath
    let posterUrl = URL(string: fullPosterPath)
    if posterUrl != nil {
        cell.posterImageView.af_setImage(withURL: posterUrl!)
    }
}
```

That's a lot of code just to get a poster url! Also, if you need to display posters in multiple view controllers, you need this code everywhere. If we were able to move that code to a Movie class, we could replace it with something like:

```
// We still need to check if the movie has a poster url
if movie.posterUrl != nil {
    cell.posterImageView.af_setImage(withURL: movie.posterUrl!)
}
```

1. Create a Movie class

   Click File->New->File. Select Swift File, not Cocoa Touch Class. Cocoa Touch Class expects a base class and inserts boiler plate code that we don't need here. Save the file as `Movie`.

   Add the class declaration:

   ```
   class Movie {

   }
   ```

2. Add properties and initializer

   Add properties for data you want to access from the API. Make sure they are the proper type and decide whether they should be Swift optionals or not. For example, every movie should have a title, so that shouldn't be an optional. However, it may or may not have a poster yet.

   If a Swift class has a non-optional property, then you must also create an initializer that sets that property. For example, since the movie comes in as a dictionary from the API, then it would be convenient if there was an initializer that took a dictionary as its parameter.

```
class Movie {
    var title: String
    var posterUrl: URL?

    init(dictionary: [String: Any]) {
        title = dictionary["title"] as? String ?? "No title"

        // Set the rest of the properties
    }
}
```

In the example above, you shouldn't do `dictionary["title"] as! String` because that could crash. The `??` operator (nil coelescing operator) will use "No title" if the cast fails.

Milestone 2: Using the movie model

1. Create Movie instances

   In the movies view controller, you have a property for an array of movies. Change it from an array of dictionaries to an array of movies.

   ```
   var movies: [Movie] = []
   ```

   In the network call, iterate through the array of dictionaries and create an array of Movies.

   ```
   let dataDictionary = try! JSONSerialization.jsonObject(with: data, options: []) as! [String: Any]
   let movieDictionaries = dataDictionary["results"] as! [[String: Any]]

   self.movies = []
   for dictionary in movieDictionaries {
       let movie = Movie(dictionary: dictionary)
       self.movies.append(movie)
   }
   ```

2. Update functions

   You'll need to update the `tableView(cellForRowAt:)` and `prepare(for segue:)` functions to use the Movie class.

3. Add helper function to Movie

   You'll often need to create an array of Movies from an array of dictionaries. It would be helpful to move this functionality to the Movie class. Add the following function to the Movie class:

   ```
   class func movies(dictionaries: [[String: Any]]) -> [Movie] {
       var movies: [Movie] = []
       for dictionary in dictionaries {
           let movie = Movie(dictionary: dictionary)
           movies.append(movie)
       }

       return movies
   }
   ```

   Refactor the view controller to use this function. **Discuss with your partner why is this a class function instead of an instance function?** If your pair doesn't have an answer, discuss with your pod.

Milestone 3: Property observers

Right now, in your view controller, you probably have a function like this:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: "MovieCell", for: indexPath) as! MovieCell

    let movie = movies[indexPath.row]
    let title = movie["title"] as! String
    let overview = movie["overview"] as! String

    cell.titleLabel.text = title
    cell.overviewLabel.text = overview

    let posterPathString = movie["poster_path"] as! String
    let baseURLString = "https://image.tmdb.org/t/p/w500"
    let posterURL = URL(string: baseURLString + posterPathString)!
    cell.posterImageView.af_setImage(withURL: posterURL)

    return cell
}
```

This is bad encapsulation, which means that the classes (NowPlayingViewController and MovieCell) are not well separated. One test for encapsulation is to see how easy it is to reuse MovieCell in another view controller. You'd have to copy all the code in `tableView(cellForRowAt:)` in the other view controller, which is bad.

1. Instead, add a movie property to the MovieCell.

   ```
   var movie: Movie!
   ```

2. Refactor `tableView(cellForRowAt:)`

   ```
   func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

       let cell = tableView.dequeueReusableCell(withIdentifier: "MovieCell", for: indexPath) as! MovieCell

       cell.movie = movies[indexPath.row]

       return cell
   }
   ```

3. Read about Swift property observers
4. Add a property observer to the movie property in the MovieCell.

   Configure the MovieCell views in the movie property observer.

This has allowed us to move code out of the view controller and makes the MovieCell more reusable. The MovieCell is responsible for configuring its own views, you just need to tell it what movie to display.

Milestone 4: Basic API client

This is an example of a simple API client, read and discuss it. It's the same code that you had in the view controller, but the tricky part is the definition of the completion parameter, which is a closure.

1. Create a new class, MovieApiManager

```
class MovieApiManager {

    static let baseUrl = "https://api.themoviedb.org/3/movie/"
    static let apiKey = "a07e22bc18f5cb106bfe4cc1f83ad8ed"
    var session: URLSession

    init() {
        session = URLSession(configuration: .default, delegate: nil, delegateQueue: OperationQueue.main)
    }

    func nowPlayingMovies(completion: @escaping ([Movie]?, Error?) -> ()) {
        let url = URL(string: MovieApiManager.baseUrl + "now_playing?api_key=\(MovieApiManager.apiKey)")!
        let request = URLRequest(url: url, cachePolicy: .reloadIgnoringLocalCacheData, timeoutInterval: 10)
        let task = session.dataTask(with: request) { (data, response, error) in
            // This will run when the network request returns
            if let data = data {
                let dataDictionary = try! JSONSerialization.jsonObject(with: data, options: []) as! [String:
                let movieDictionaries = dataDictionary["results"] as! [[String: Any]]

                let movies = Movie.movies(dictionaries: movieDictionaries)
                completion(movies, nil)
            } else {
                completion(nil, error)
            }
        }
        task.resume()
    }
}
```

2. Refactor NowPlayingViewController to use the MovieApiManager

```
MovieApiManager().nowPlayingMovies { (movies: [Movie]?, error: Error?) in
    if let movies = movies {
        self.movies = movies
        self.tableView.reloadData()
    }
}
```

3. Add a function for fetching popular movies.