LambdaSchool / **ios-sprint12-challenge**

Sprint challenge for Sprint 12 of Lambda's iOS course

3 commits **1** branch **0** releases **1** contributor

Branch: **master** New pull request Create new file Upload files Find File Clone or download

armadsen Update README.md Latest commit 62ddc78 on Oct 12, 2018

.gitignore	Initial commit	8 months ago
README.md	Update README.md	5 months ago

README.md

Pokedex

Instructions

Please read this entire README to make sure you understand what is expected of you before you begin.

This sprint challenge is designed to ensure that you are competent with the concepts taught throughout Sprint 12, Objective-C Part 2.

Begin by forking this repository. Clone your forked repository to your machine. There is no starter project. Create a project for this challenge in this repository and commit as appropriate while you work. Push your final project to GitHub, then create a pull request back to this original repository.

You will have 3 hours to complete this sprint challenge

If you have any questions about the project requirements or expectations, ask your PM or instructor. Good luck!

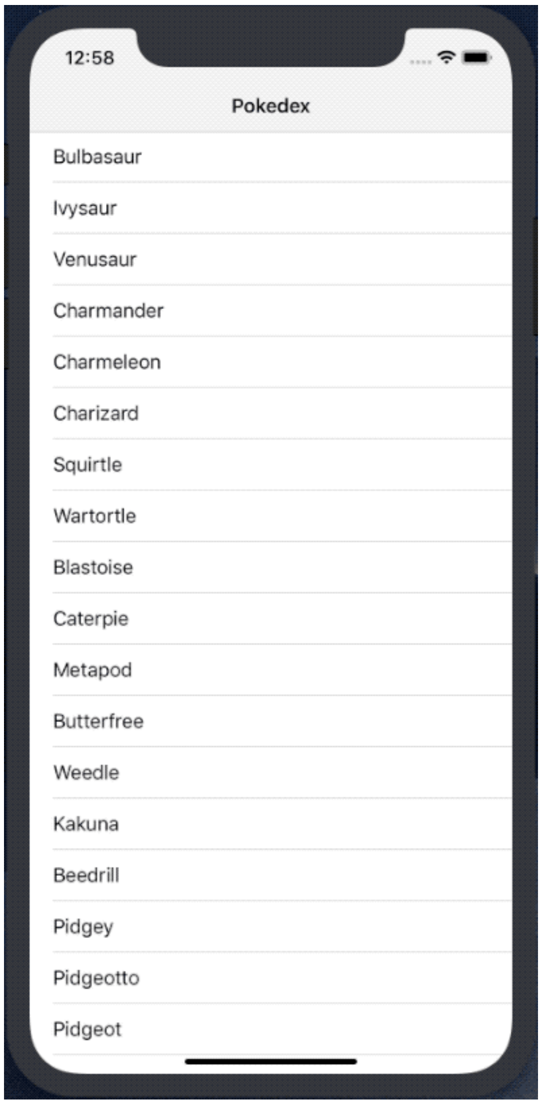
Description

For this project, you will build a viewer app for the Pokedex API. The Pokedex API does not require an API key or any other form of authentication.

The app will consist of two screens, a main table view, and a detail view. You will implement it using a mix of Objective-C and Swift.

Screen Recording

Please view the screen recordings so you will know what your finished project should look like:



(The gif is fairly large in size. It may take a few seconds for it to appear)

Requirements

The goal of this sprint challenge is to create a viewer for the Pokedex Pokemon API.

The requirements for this project are as follows:

1. Your model must support at least: pokemon name, identifier, sprite, and abilities.
2. For abilities, only the name of each ability is needed.
3. Classes must be written in Objective-C except for the networking class. **The detail view controller can be written in either Objective-C or Swift. It's your choice.**
4. All Objective-C classes must be annotated for nullability and generics as appropriate.
5. You must populate the table view with the results of the `https://pokeapi.co/api/v2/pokemon/` (see Hints and Tips below). Only when a Pokemon is tapped to show the detail view controller should you load the rest of its information.
6. Your network controller must have an interface as specified below. Note that the `fillInDetails` method **does not take a completion closure**. Rather, it fills in additional details (identifier, abilities, sprite) by setting `pokemon` 's properties directly.
7. The detail view controller must use KVO to be notified when the network controller has finished filling in details for a Pokemon, triggering a UI update.
8. You must take care to avoid memory mangement issues. (However, to be clear, you should not use Manual Reference

Counting.)

Network class interface:

```
class PokemonAPI: NSObject {

    @objc(sharedController) static let shared: PokemonAPI

    @objc func fetchAllPokemon(completion: @escaping ([LSIPokemon]?, Error?) -> Void)

    @objc func fillInDetails(for pokemon: LSIPokemon)
}
```

Hints and Tips

1. Use the "Try it now!" box on https://pokeapi.co to test URLs and see the JSON returned by the API.
2. A GET request to `https://pokeapi.co/api/v2/pokemon/` will return a list of all Pokemon supported by the API. However, this list only includes each Pokemon's name and a url to get more information about it.
3. A GET requeset to `https://pokeapi.co/api/v2/pokemon/<pokemon_name>` (where `<pokemon_name>` is the name of the Pokemon in question), will return fullly detailed information about a specific Pokemon.
4. If you need to use KVO in Swift, here's a primer:

The KVO observation API in Swift is significantly different from Objective-C. In Swift, start observing a key path on an object by calling the `observe(_ keyPath:, changeHandler:)` method. Here, `keyPath` is a key path, which is **not** just a string in Swift. In Swift, specify a keypath by doing `\ClassName.firstKey.secondKey.etc` (you can usually leave the initial `ClassName` off). `changeHandler` is a closure that takes two arguments: the object that triggered the observation, and an `NSKeyValueObservedChange` object with information about the change that occurred. You usually don't need to worry about these arguments. Finally, the `observe` method returns a `NSKeyValueObservation` object. **You must store this object in a property, and get rid of it when you want to stop observing.** You can find the documentation for KVO in Swift here.