Assignment IV:

Top Places

Objective

In this series of assignments, you will create an application that presents a list of popular Flickr photo spots. This first assignment is to create a navigation-based application to let users browse the most popular places on Flickr, click on any they are interested in to see some photos taken in that place.

The primary work to be done in this assignment is to create build a tab-based user-interface with two tabs: Top Places and Recents. The first two will show the names of places and the second a list of the most recently-viewed photos.

The goals are to get familiar with table views, tab bar controller, scroll views, image views and to get experience building yet more MVCs in your application.

All the data you need will be downloaded from Flickr.com using Flickr's API. Code will be provided for the Flickr queries you need for this assignment.

Be sure to check out the Hints section below!

Also, check out the latest in the Evaluation section to make sure you understand what you are going to be evaluated on with this assignment.

Materials

- This is a completely new application, so you will not need anything (but the knowledge you gained) from your first three homework assignments.
- You will need to obtain a Flickr API key. A free Flickr account is just fine (you won't be posting photos, just querying them).

Required Tasks

1. Use the provided `FlickrFetcher` class method `topPlaces` to get an array of the most popular Flickr photo spots in the last day or so. It is an array of `NSDictionary`s which contain information about each place.
2. Create a `UITabBarController`-based user-interface with two tabs. The first shows a `UITableView` with the list of places (in alphabetical order) obtained in Required Task #1. The second shows a `UITableView` with a list of the 20 most recently viewed photos.
3. Anywhere a place appears in a table view in your application, the most detailed part of the location (e.g. the city name) should be the `title` of the table view's cell and the rest of the name of the location (e.g. state, province, country, etc.) should appear as the `subtitle` of the table view cell.
4. When the user chooses a place from the list obtained in Required Task #1, you must query Flickr again to get an array of 50 recent photos from that place and display them in a list. Do this using the `FlickrFetcher` method `photosInPlace:maxResults:` (it returns an array of dictionaries, each of which contains info about a photo).
5. Any list of photos should display the photo's title as the table view cell's title and its description as the table view cell's subtitle. If the photo has no title, use its description as the title. If it has no title or description, use "Unknown" as the title.
6. When the user chooses a photo from any list, display its image inside a scrolling view that allows the user to pan and zoom (a reasonable amount). You obtain the URL for a Flickr photo's image using `FlickrFetcher`'s `urlForPhoto:format:` (use `Large`).
7. Make sure the photo's title is somewhere on screen whenever you are showing the photo image to the user.
8. Whenever a photo's image appears on screen, it should initially be zoomed to show as much of the photo as possible with no extra, unused space. It is not necessary to continue to do this as the user rotates the device or zooms in and out on the photo by pinching.
9. Your application must work in both portrait and landscape orientations on the iPhone. Support for the iPad is optional (though it will be required next week, so you can save time later by implementing it now). Use appropriate platform-specific UI idioms (e.g., you must use `UINavigationController`s to present the information on the iPhone).
10. The recents tab must show the list of most recently view photos in chronological order of viewing with the most recent at the top, and no duplicates in the list. It is sufficient to only update the list each time it (re)appears on screen (i.e. in `viewWillAppear:`). A photo can be uniquely identified by its "id" entry in its dictionary.
11. The list of recents photos should be saved in `NSUserDefaults`. The arrays you get back from the `FlickrFetcher` methods are all property lists.

Hints

1. Put your own Flickr API key into `FlickrAPIKey.h` or your queries will not work.
2. It is possible to start off this assignment with the Tabbed Application template (or even the Master-Detail Application template) and you are welcome to play with doing so. However, it is also fine to just start with the Single View Application template and drag in the `UITableViewController`s you need and use the Embed menu item (for `UINavigationController`s, `UITabBarController`s and `UIScrollView`s) as needed and then ctrl-drag to set up Relationships (like `rootViewController` or `viewControllers`) and Segues. It is an important part of this assignment to reinforce your understanding of how these storyboard-construction objects all relate to each other.
3. The very first thing you're probably going to want to do once you have copied the `FlickrFetcher` code into your application (and set your API key) is to do a `topPlaces` query and then `NSLog()` the results. That way you can see the format of the fetched Flickr results (it's an `NSArray` of `NSDictionary` objects). Ditto when you query Flickr for the list of photos at a given place.
4. If you look carefully, you'll notice that the value for the key `description` in a dictionary of photo information from Flickr is not actually the photo's description. Instead, it's another `NSDictionary` that has a key `_content` in it. That's where the actual description is. The method `valueForKeyPath:` can be sent to an `NSDictionary` with a key with dots in it, e.g., "`description._content`" to access sub-dictionaries.
5. The key `id` (in a photo's dictionary of info) is a unique, persistent photo identifier.
6. To create a table-view-based MVC, drag a Table View Controller out of the Object Library into your storyboard and change its class to be a custom subclass of `UITableViewController` (don't forget to change the superclass to `UITableViewController` in the dialog that New File ... brings up). You will likely want a number of different `UITableViewController` subclasses for this assignment.
7. Each MVC should be set up with the information it needs before it is pushed and then allowed to go do its thing. And use your awesome object-oriented programming design skills to be certain to reuse as much code as possible. Many of the MVC's in this application are very similar. It is perfectly fine to create a subclass of `UITableViewController` to do something, then create a subclass of that class to do something slightly more refined.
8. Note that all the `UITableViewCell`s in this assignment require subtitles, so you must set that as the type of the cell in Xcode for your dynamic prototypes. It's probably also a good idea to set the "backup" cell creation code in `tableView:cellForRowAtIndexPath:` to be the `Subtitle` type as well (just for consistency).
9. Don't forget that the `UITableViewCell` reuse identifiers that you set in Xcode for dynamic prototype cells must match what is in your

`tableView:cellForRowAtIndexPath:` methods. This can be a little confusing if you choose to have subclasses of subclasses of `UITableViewController` (since you are then inheriting `tableView:cellForRowAtIndexPath:`), so pick good reuse identifier names (that succinctly describe what the cell is displaying).

10. Turning a URL on the internet into a `UIImage` is easy. Just create an `NSData` with the contents of that URL (`[NSData dataWithContentsOfURL:theURL]`), then create a `UIImage` using that `NSData` (`[UIImage imageWithData:imageData]`).
11. Required Task #8 (initial photo zooming) requires some calculations involving the `UIScrollView`'s `bounds` and the size of the photo. Don't forget from lecture where (in the View Controller Lifecycle) geometry calculations for a view have to occur.
12. If you support iPad this week, don't forget you'll need some view controller to serve as your `UISplitViewControllerDelegate`. Any of them is probably okay, but if you choose your detail view controller, you will be able to do the bar button dance a little more easily (since that's the view controller that places the button in the UI).
13. If you want to update the detail view controller in a split view on the iPad from a master view controller which is a table view controller, you'll probably want to implement the `tableView:didSelectRowAtIndexPath:` method in the master (it's sort of the "target/action" method of a table view) rather than segueing. You'll want to do the same things in that method that you do in `prepareForSegue:sender:` (i.e. set the Model (and any "how to display this" properties) of the destination view controller).
14. If you are resetting the image of your image-displaying MVC (e.g. it's the detail view controller in a split view), be careful to reset your `UIScrollView`'s `zoomScale` back to `1` before you reset the `contentSize` for a new image. The `zoomScale` affects the `contentSize` (e.g., when you zoom in the `contentSize` is automatically be adjust to be larger and when you zoom out, it gets smaller), so if you have a `zoomScale` other than `1` and you start mucking with the `contentSize`, you'll get results you're probably not anticipating.
15. The method `mutableCopy` in `NSArray` might come in handy when you want to add something to a data structure already stored (immutably) in `NSUserDefaults`.
16. You're going to notice that your application is not very responsive. Whenever it goes off to query Flickr, there'll be a big pause. This is very bad, but we will be learning how to solve this next week, so don't waste your time trying to fix it now.
17. As always, the amount of code required to implement this application is not huge (under 100 lines of code, if you only count the ones added between curly braces). If you find yourself needing dozens of lines of code for any one feature, there's probably a better way to go about it. In general, "brawn over brains" solutions (i.e. "just keep typing in code until it works") are bug-prone and a pain to maintain, so avoid them like the plague! We use object-oriented programming for a reason. Use its mechanisms to the fullest.

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Code is sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Assignment was turned in late (you get 3 late days per quarter, so use them wisely).
- Code is too lightly or too heavily commented.
- Code crashes.

Extra Credit

If you do any Extra Credit items, don't forget to note what you did in your submission `README`.

1. Divide your list of top places into sections (in the table view sense) by country. This will require a little bit different data structure in that MVC.