Assignment IV: SPoT

Objective

In this series of assignments, you will create an application that lets users tour Stanford through photos. This first assignment is to create a navigation-based application to let users browse different categories of spots at Stanford, then click on any they are interested in to see a photo of it.

All the information you need will be downloaded from Flickr.com using Flickr's API.

This assignment is due by the start of lecture next Tuesday.

Be sure to check out the **Hints** section below!

Materials

- This is a completely new application, so you will not need anything (but the knowledge you have gained) from your previous homework assignments.
- You will need the `FlickrFetcher` helper code (download from the same place you found this document).
- 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. You will be using the method `stanfordPhotos` in the provided `FlickrFetcher` helper code to get an `NSArray` of Flickr photos of various sites on the Stanford campus. Each entry in the `NSArray` is an `NSDictionary` object with numerous entries which describe the photo (more on this below). `FlickrFetcher.h` has `#define`s for the keys.
2. Your user-interface should present a `UITabBarController`-based UI with two tabs, each of which contains its own navigation (i.e. `UINavigationController`-based) stack. One tab lets you browse (in the manner described below) all of the photos returned in the above query and another which maintains a list of the photos the user has most recently viewed in your application.
3. The main (non-Recents) tab must present a list containing all of the tags (`FLICKR_TAGS`) on all of the photos queried except the tags "cs193pspot," "portrait" & "landscape" (note that Flickr tags come back all lower case with no special characters). The subtitle for each cell in this list should display how many photos have the corresponding tag. Capitalize the tags so they look a bit nicer in the list.
4. When the user chooses one of the tags in the list, navigate to a new list of the titles of all the photos (in the data you queried originally with `stanfordPhotos`) that have that tag. The subtitles in this list should be the photo's description (if it has one).
5. When the user then chooses a particular photo from the list, display it inside a `UIScrollView` that allows the user to pan and zoom.
6. When a photo first appears, and as the `bounds` of the `UIScrollView` showing a photo change (due to, for example, autorotation), you must adjust the zooming to show as much of the photo as possible with no extra, unused space. Once the user starts pinching to zoom on a given photo, you can stop doing this automatic zooming while that photo continues to be visible.
7. All lists should each be displayed in a `UITableView`.
8. The recents tab must show a list of the most recently view photos in order, with the most recent at the top and no duplicates in the list. The list of recents must also persist across application termination and relaunch. When a recent photo in the list is chosen, it should navigate to a view of the photo in the same way as other table views in the application do. Limit the size of the list to a reasonable number.
9. Since all the lists you display allow further navigation, they should display a disclosure indicator in every cell (Xcode will automatically add this for you when you create a segue from a table view cell).
10. This application must work properly in both portrait and landscape modes on the iPhone 4 and iPhone 5. It is extra credit to also do the iPad, but the iPhone version is required.

Hints

These hints are not required tasks. They are completely optional. Following them may make the assignment a little easier (no guarantees though!).

1. After you copy in the `FlickrFetcher` helper files, put your own Flickr API key into `FlickrAPIKey.h` or your queries will not work.
2. 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 the query and then `NSLog()` the results. That way you can see the format of the fetched Flickr results (i.e., the `NSArray` of `NSDictionary` objects).
3. You'll notice that the `tags` that are returned by Flickr are separated by spaces (e.g., "cs193pspot portrait classroom engineering"). The `NSString` method `componentsSeparatedByString:` might come in handy.
4. You'll also notice that the value for the key `description` 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.
5. The key `id` (in the dictionary returned from Flickr) is a unique identifier for the photo.
6. This may be obvious, but there are no static table views in this assignment (e.g. all of the table views will use dynamic prototype cells).
7. Do not communicate data between view controllers on your navigation stack using global data! Each view controller should be set up with the information it needs before it is pushed and allowed to go do its thing with no further intervention. Be certain to reuse your table view code as much as possible (e.g., the implementation of the Recents tab is **very** similar to the implementation of the browsing tab!).
8. It is perfectly fine to use a `UITabBarItemSystemItem` as the `tabBarItem` for either or both view controllers that need one in this application. It would be poor implementation for either of your tabs not to have both an icon and a title, so this should make it easy to make it look nice.
9. The best size picture to use is probably `FlickrFetcherPhotoFormatLarge` (that's 1024x768). On the iPad (if you're doing that extra credit), you might want to move up to `FlickrFetcherPhotoFormatOriginal`. See the Flickr doc about sizes if you're interested.
10. The Required Task about auto photo zooming requires some calculations involving the `UIScrollView`'s `bounds` (i.e. the scroll view's visible area in its own coordinate system) and the size of the photo. Recall from the update in lecture about when in the View Controller's lifecycle you can do geometry calculations with Autolayout on.
11. 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 this week.

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.

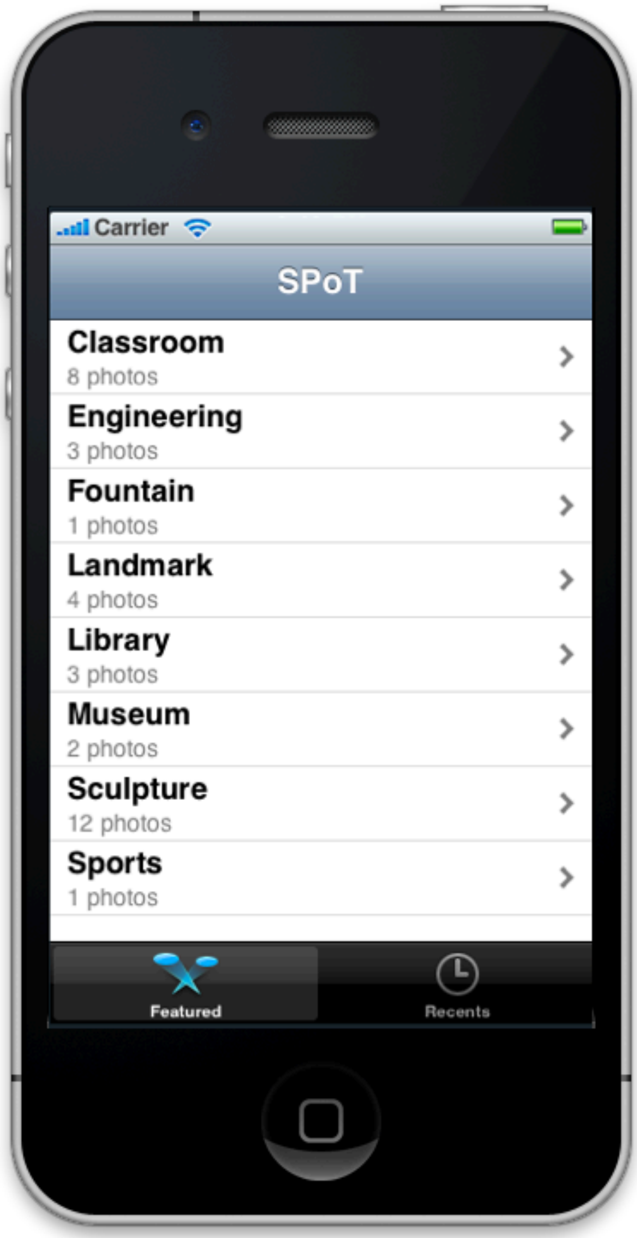
Here are the most common reasons assignments are marked down:

- 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.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, etc.
- UI is a mess. Things should be lined up and appropriately spaced to "look nice." Xcode gives you those dashed blue guidelines so there should be no excuse for things not being lined up, etc. Get in the habit of building aesthetically balanced UIs from the start of this course.
- Assignment was turned in late (you get 3 late days per quarter, so use them wisely).
- Incorrect or poor use of object-oriented design principles. For example, code should not be duplicated if it can be reused via inheritance or other object-oriented design methodologies.

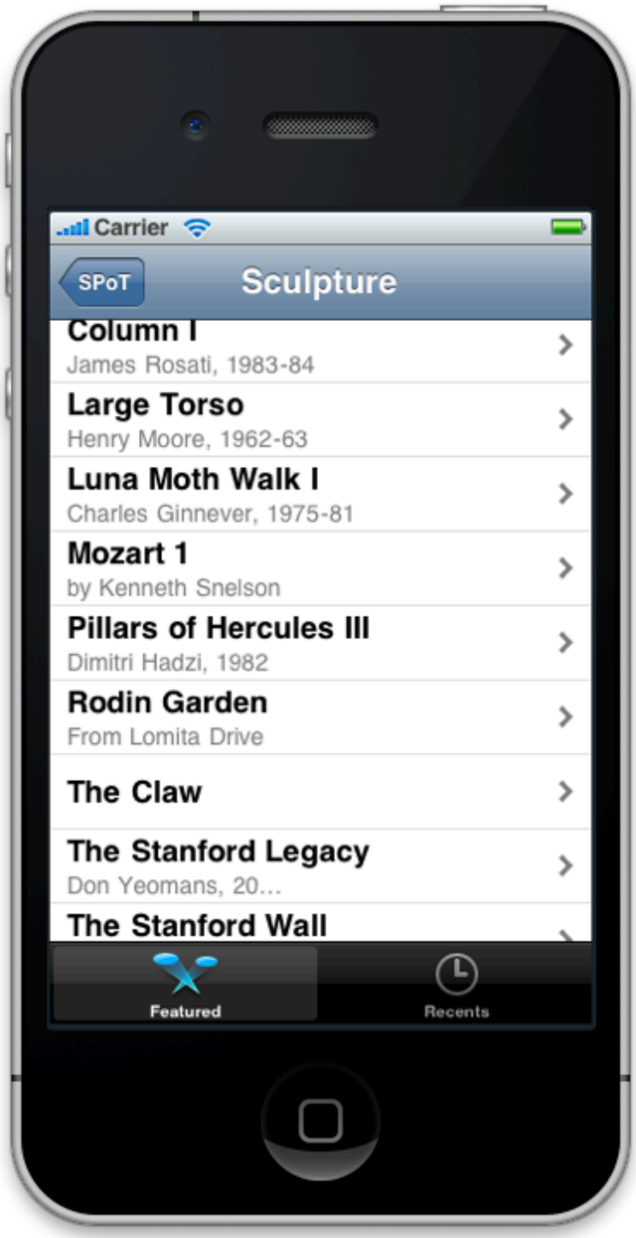
Often students ask "how much commenting of my code do I need to do?" The answer is that your code must be easily and completely understandable by anyone reading it. You can assume that the reader knows the SDK, but should not assume that they already know the (or a) solution to the problem.

Screen Shots

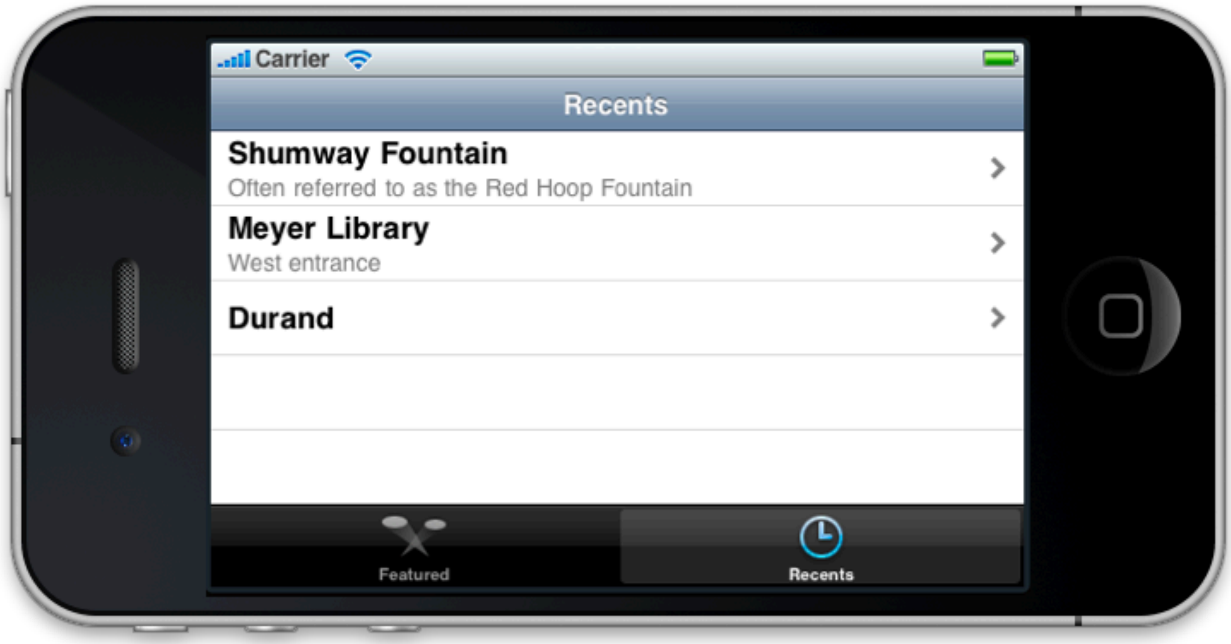
These screen shots are only meant to be examples. Your actual implementation may vary.



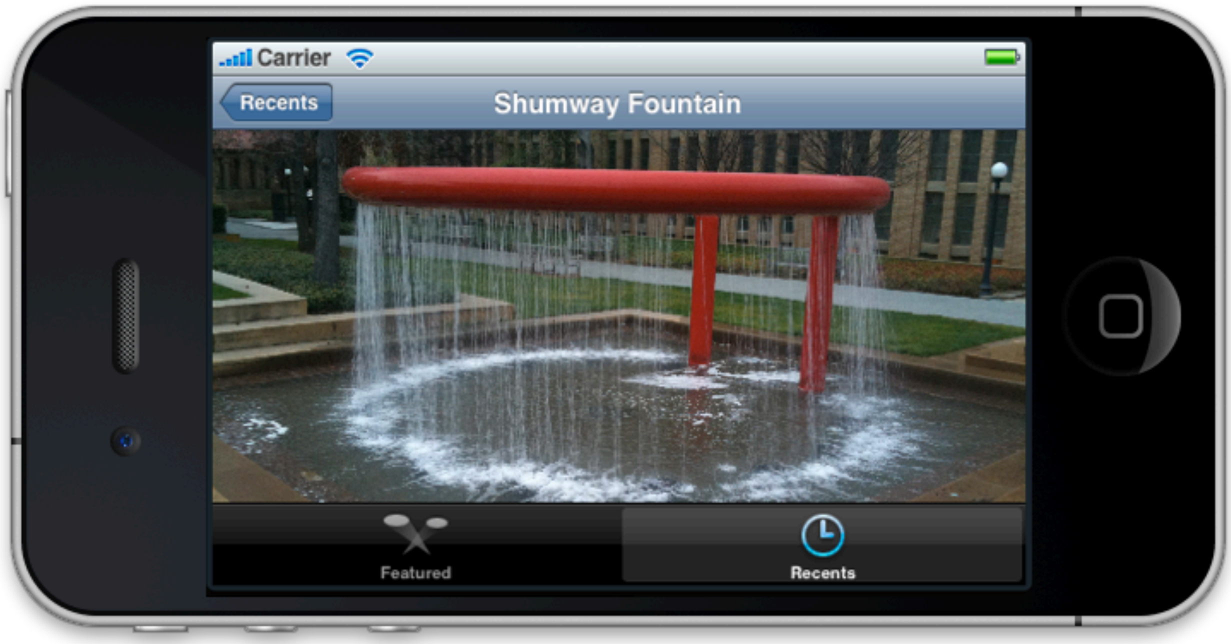
Classroom
8 photos
Engineering
3 photos
Fountain
1 photos
Landmark
4 photos
Library
3 photos
Museum
2 photos
Sculpture
12 photos
Sports
1 photos



Column I
James Rosati, 1983-84
Large Torso
Henry Moore, 1962-63
Luna Moth Walk I
Charles Ginnever, 1975-81
Mozart 1
by Kenneth Snelson
Pillars of Hercules III
Dimitri Hadzi, 1982
Rodin Garden
From Lomita Drive
The Claw
The Stanford Legacy
Don Yeomans, 20…
The Stanford Wall



Shumway Fountain
Often referred to as the Red Hoop Fountain
Meyer Library
West entrance
Durand



Extra Credit

1. Show your lists sorted alphabetically. There are two methods that might be helpful in this: `sortedArrayUsingSelector:` and `sortedArrayUsingDescriptors:`. The former is good when you have an array of `NSString` objects. The latter is good when you have an array of `NSDictionary` objects (and you want to sort by one of the values in the dictionaries).
2. Make your application work on the iPad as well with appropriate user-interface idioms on that platform. This will require you to have absorbed the lecture immediately after this was assigned. But you'll be asked to do this next week anyway, so you'll be ahead of the game if you do it this week (and get a little bit of extra credit for doing it early).