Paparazzi - Part 3

Due Date

This assignment is due by **11:59 PM, February 17.**

Assignment

Last week, we continued making progress on our Paparazzi application, adding table views and storing data in a database with Core Data. However, we still only have static data that we're loading from a plist stored locally in the application. That's a great way to try out some basic interface ideas, but in the end we want to get real data off of a real web service.

This week we'll be fetching data from Flickr and plotting the locations of the photos we're downloading on a Map using the MapKit framework.

Here are the requirements for Part 3:

1. **Load images from Flickr and store them locally in the Core Data database**. We've added methods to the FlickrFetcher class that wrap some of the Flickr API to download images. Use them to download recent photos that have been tagged with location data and recent photos from a specific user.
2. **Use a modal view controller to ask our user for a Flickr user name for which to download photos.** We talked about using modal view controllers to request additional information from the user, so now we'll create a new view controller subclass and present it modally to ask our user for the Flickr user name for which to download some photos.
3. **Plot the locations of photos that include location information on a map using MapKit**. If the Flickr photo includes latitude and longitude we want to add a pin to a map to allow the user to browse photos by location. We'll need a new UIViewController subclass to manage this MKMapView, and we'll want to add it to our UITabBarController in our main xib.
4. When one of the pins is tapped on the map, **a callout should pop up to show the user the name of the photo represented by the pin** along with the name of the user that took the photo.
5. When a button is tapped in the callout bar associated with a map pin, **set up and push a PhotoViewController to display that photo**.

There is an archive accompanying this assignment titled **Paparazzi3Files.zip** which includes an expanded version of the FlickrFetcher class from Paparazzi 2 which now facilitates interacting with the Flickr API.

Testing

In most assignments testing of the resulting application is the primary objective. In this case, testing/grading will be done both on the behavior of the application, and also on the code.

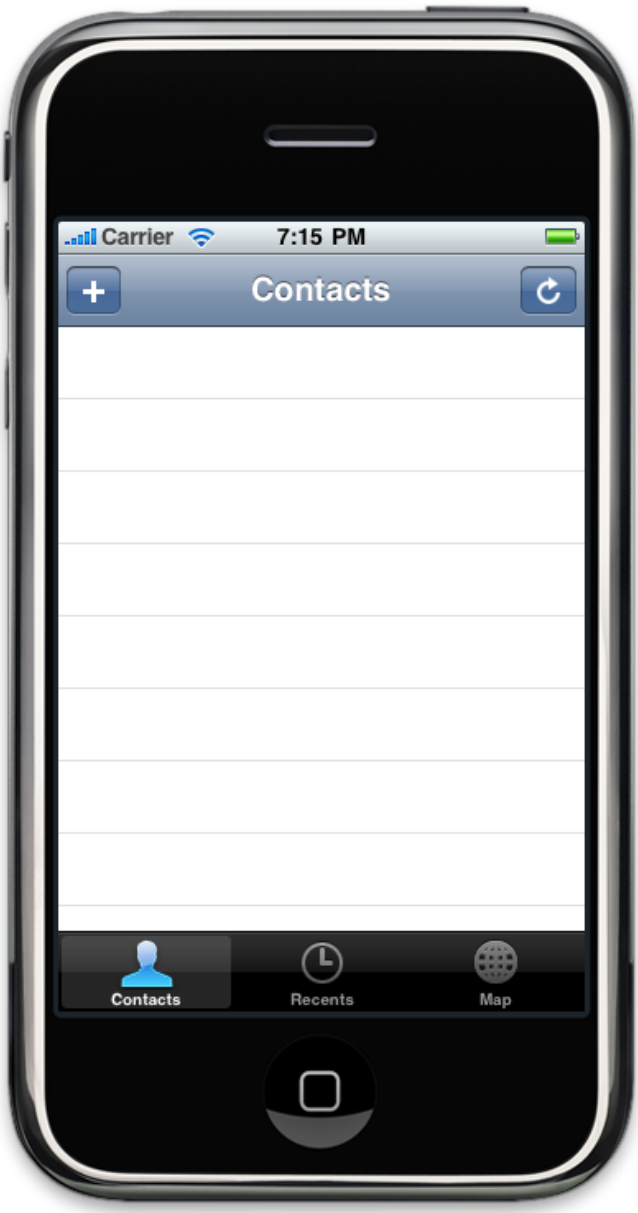
We will be looking at the following:

1. Your project should build without errors or warnings and run without crashing.
2. Each view controller should be the File's Owner of its own Interface Builder document. **Do not put your entire application into a single Interface Builder document!**
3. You should be using retain, release and autorelease correctly at this point. **Don't leak memory or over-release**.
4. Since the project is getting more complex, **readability is important**. Add comments as appropriate, use descriptive variable and method names, and decompose your code.
5. Your program should behave as described above, loading photos from Flickr without blocking the main thread, displaying photos on the map, and allowing the user to view photos from the map.

Walkthrough

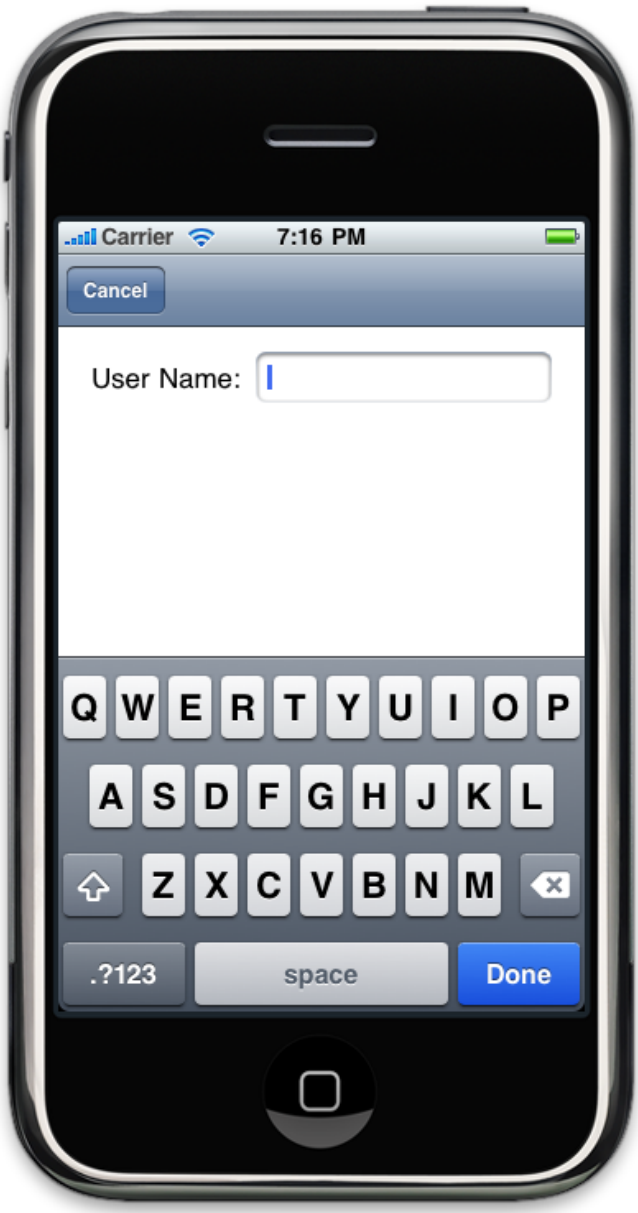
Determining what photos to load

The FlickrFetcher class now provides conveniece functions for downloading photos for a specific user and for downloading recent photos from any user as long as they are geo-tagged. We'd like to be able to use both, but to load photos for a specific user we'll need to ask our user to enter a username. To do this we'll create a new UIViewController subclass that we'll present modally when the user taps on a button in the navigation bar of our Contacts view controller. We'll need a new UIBarButtonItem to represent this action, and while we're at it we'll add one that we'll use to load recent photos, so we should have something like this:



When the user taps the add button, we'll want to present our new view controller modally to request a username. Keep in mind the techniques we discussed for passing data between view controllers. You shouldn't be using globals to pass the value entered by the user back to the calling view controller, so consider the delegation pattern we've discussed.

When it's presented, we should have something like this (though feel free to fancy it up):



Loading photos from Flickr
Take a minute to look over the new methods that have been added to the FlickrFetcher class. These methods are fairly thin wrappers around the http-based Flickr API. They simply construct a URL, use a synchronous (blocking) URL request to download the results, and parse them with the open-source JSON parser we discussed in class. For now we'll be calling these from the main thread, but our UI will remain unresponsive while the network request is being processed. We'll address that next week in Paparazzi 4 using some of the techniques we've discussed in class, so for now blocking the user interface will suffice.

You can insert the users and photos downloaded using the new FlickrFetcher methods the same way you had previously been inserting the contents of the FakeData.plist. It should be pretty straightforward to modify your existing insertion code to work with the results of the Flickr fetch request instead of the fake data. Keep in mind that you may need to modify your model to store additional information, for instance the latitude and longitude. You might also consider storing the downloaded photo data directly within the Core Data database as an NSData rather than in separate files, though either approach is fine.

FlickrFetcher.h defines an enumerated type that allows you to request photos in various formats, and you may want to download more than one for display in different situations.

Plotting photos on a map
Now that we have some geo-tagged photos downloaded and displaying in our table views we can plot them using the MapKit framework. Keep in mind what you had to do to your project to make use of Core Data before you begin calling MapKit functions. You'll want to modify your existing main nib to include a new UIViewController subclass in the tab bar in which we can place our MKMapView. You may also want to embed your new map view controller within a UINavigationController so the user can view photos from the map.

Make use of MKPlacemark, MKMapViewDelegate and UIButton to add annotations to your MKMapView at the locations of each geo-tagged photo. The MapKit API allows for a lot of customization, but you can get all of the expected functionality using just the following:

- MKMapView
- MKMapViewDelegate
- MKPlacemark

You can provide a much more custom interface using custom MKAnnotationViews, but that's not necessary to get the basic level of functionality.

Once this is done you should have something that looks like the image on the next page. You'll notice that the callout bar has a detail disclosure button on the right. When tapped this button should cause a PhotoDetailViewController to be pushed to display the full image. You may wonder how that's possible given that there's no navigation bar visible in my screenshot, but if you look through the methods available on UINavigationController it should be pretty obvious how that might be done.


Carrier
7:36 PM
Christmas lights at the dome
jonomillin
North America
Atlantic Ocean
Europe
Africa
South America
Google
Contacts
Recents
Map

Extra Credit

Here are some suggestions for enhancing the third version of Paparazzi. Some of these require additions to the provided Flickr wrappers. The entire Flickr API is available at <http://www.flickr.com/services/api/>

- Provide a way to download additional photos for a particular Flickr user from the user's photo list. This would require some additions to the existing Flickr API wrappers to specify a different range of photos to download.
- Display photo thumbnails directly on the map using a custom MKAnnotationView.
- Add the ability to download and display photo comments from Flickr.

If you undertake any extra credit, please let us know in your submission notes or otherwise. If you don't, we might not know to look for it. And be sure that the core functionality of your application is solid before working on any of this!