Paparazzi - Part 1

Due Date

This assignment is due by **11:59 PM, February 3.**

Assignment

Over the next four weeks, we'll be building an iPhone application for viewing online photos, also known as "paparazzi", for a list of friends. Just so you know what you're getting yourself into, the evolution of the application will be as follows:

Part 1: Build a basic application, displaying static data, using view controllers. Allow the user to navigate to see additional detail.

Part 2: Use table views and Core Data to display large dynamic data sets.

Part 3: Fetch photos from the Internet, populating the Core Data database with real data. Plot geo-tagged photos' locations on a map using MapKit.

Part 4: Improve the performance and responsiveness of your application with caching and threading. Add support for pinching to zoom in and out on photos. Anything else you can think of! This is your opportunity to try out unfamiliar API, polish your interface and experiment a bit.

Now that you have some idea where we're headed, here's what we're expecting for Part 1:

- Create an application that utilizes UINavigationController and UITabBarController. The navigation and tab bar controllers may be instantiated in your MainWindow XIB or programmatically in your application delegate.
- **Create a view controller that will manage a list of people** (a good name might be PersonListViewController). Use Interface Builder to lay out a view and make connections between the view controller and user interface elements. The list should display a photo, a text label with the photographer's name and a "View" button next to each person (show at least two).
- **Create another view controller that will manage a list of photos** (a good name might be PhotoListViewController). The PhotoListViewController should expose a property for setting an array of photos it will display. The list should display each photo, a text label with the photo's name, a text label with the photographer's name, and a "View" button next to each photo. Again, use Interface Builder to lay out the view and make connections.
- **Create one more view controller that will display a specific photo** (a good name might be PhotoDetailViewController). The PhotoDetailViewController should expose a property for setting the photo. Once more, use Interface Builder to lay out the view and make connections.
- When your application launches it should create a tab bar controller and two navigation controllers, and then add the navigation controllers to the tab bar controller. The first navigation controller will be used for a list of contacts, and the second for a list of recently viewed images. It should then create an instance of the PersonListViewController and push it onto the first navigation controller's stack, and create an instance of the PhotoListViewController and push it on the second navigation controller's stack. An appropriate title should display in both navigation bars.
- When the user presses one of the "View" buttons in the list, create a PhotoListViewController or PhotoDetailViewController instance as appropriate, set its display properties to reflect the data that's being displayed, and push it onto the navigation stack. Again, an appropriate title should display in the navigation bar.

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.

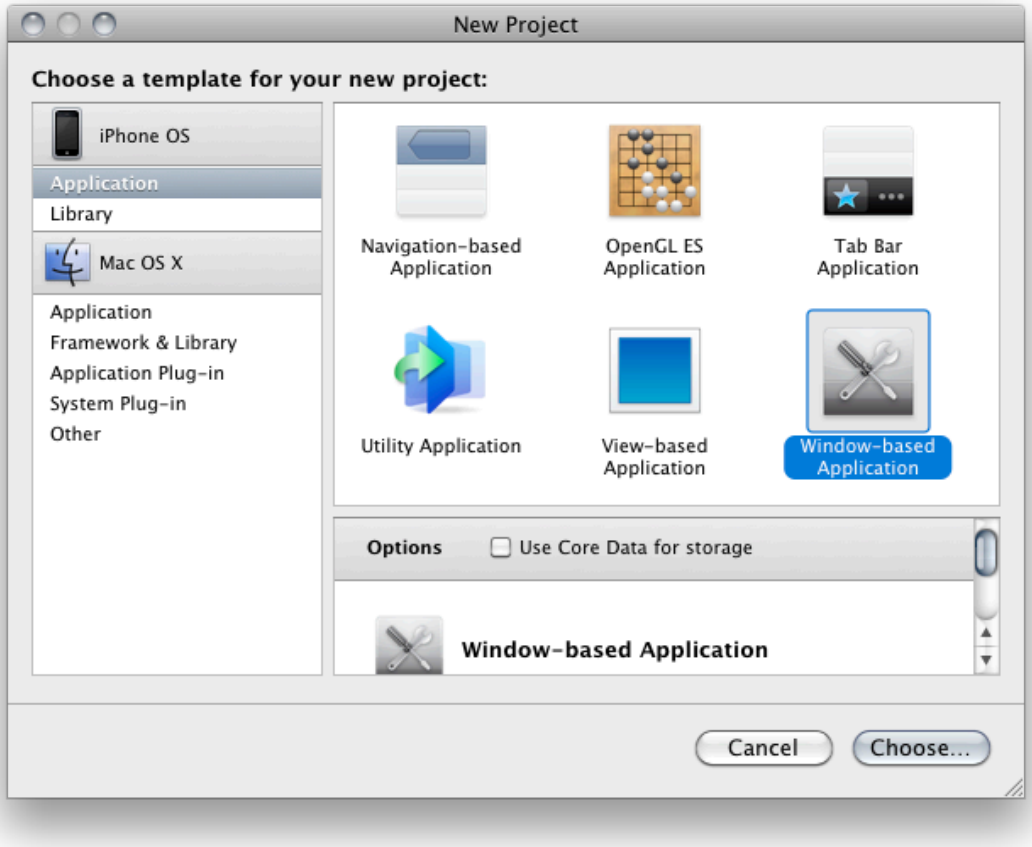
We will be looking at the following:

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!** It's bad for performance as well as application maintainability.
3. Your program should behave as described above, presenting a tab bar with two navigation hierarchies. The first navigation hierarchy should be a list of people that leads to a list of photos and ends with a photo detail view. The second should be a list of photos that leads to a photo detail view.

Walkthrough

Creating your project in Xcode

To begin with, create a new project using the **"Window-Based Application" template** in Xcode. There is a "Navigation-Based Application" template which may look tempting, but it has a lot of code already written (particularly for incorporating a UITableView) which we don't want for this assignment.



Using this template will create a project that has an application delegate class and a MainWindow.xib Interface Builder document.

Creating your tab bar and navigation controllers

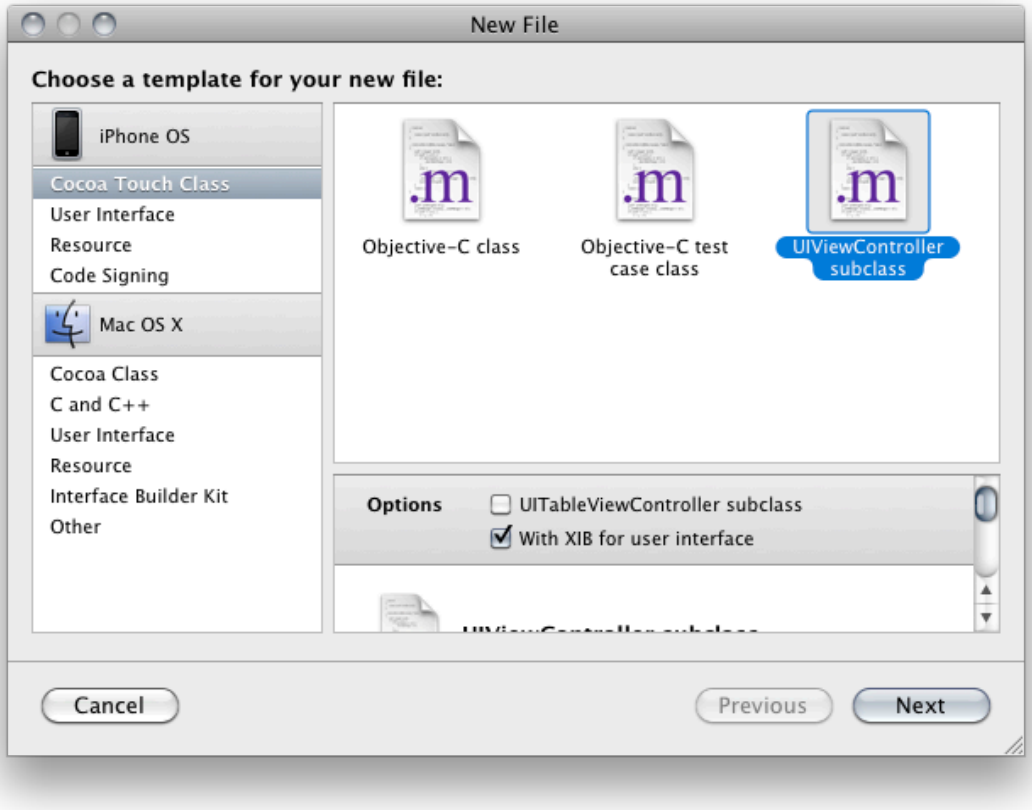
You can create a UITabBarController and two UINavigationControllers in your MainWindow NIB or in code using the -init method. Either way, your application delegate should probably have instance variables referencing them (don't forget to release them in the appropriate place!).

With the tab bar controller created, you'll need to add its view to the window. Remember, UITabBarController is a UIViewController subclass, so it has a view property that you can access.

Try building & running your application now & see what happens.

Creating your first view controller subclass

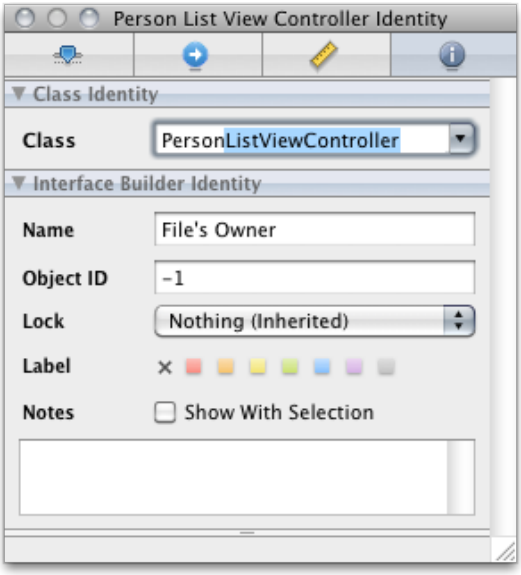
First, we need to create the header and implementation files for our new UIViewController subclass. Select "New File…" and use the **UIViewController subclass** file template. Check the "With XIB for user interface" checkbox to get an XIB created for you. If you don't check it now you can always create one later from the User Interface tab on the left.



Using this template will create a UIViewController subclass with quite a few methods already filled in. You can peruse this if you're curious, but then **delete all of the implemented methods** from the .m file. We don't need any of that right now.

Lay out your static list of people using labels, buttons and image views. In addition, if you didn't create the XIB with the "With XIB for user interface" checkbox, **we need our view controller to manage the XIB**. There are two steps to make this happen.

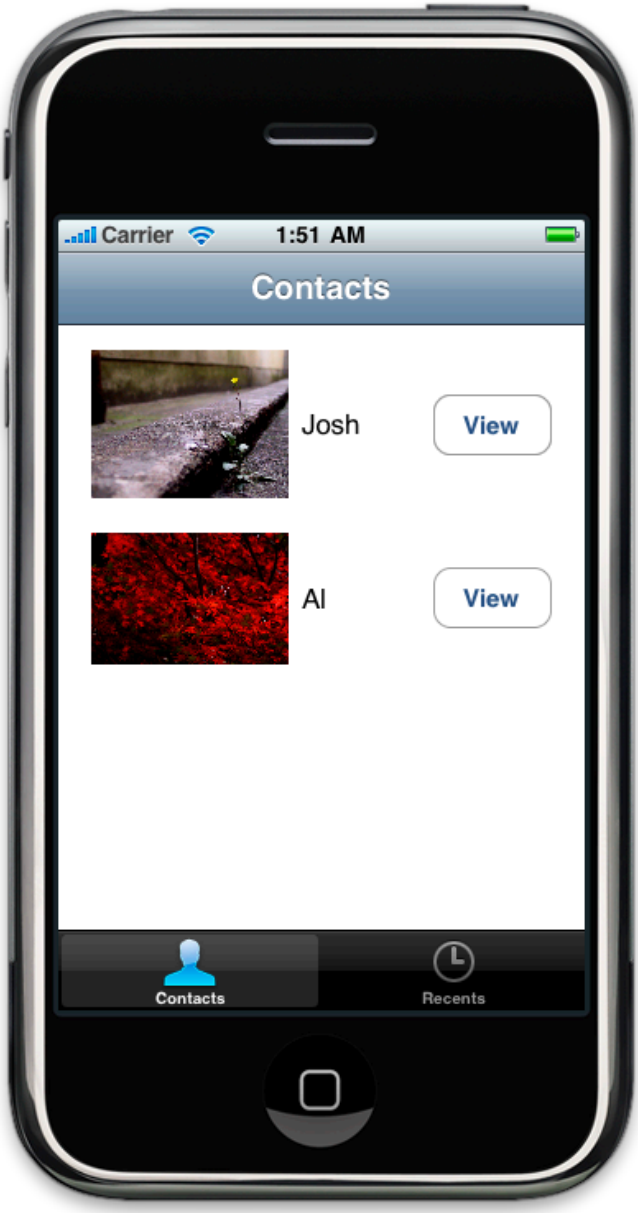
First, we need to **set the view controller as the custom class for the File's Owner proxy object**. Select the File's Owner object in your xib, then navigate to the Identity Inspector in the Tools menu. Type in the name of your view controller subclass in the text field.



Now that the XIB knows that the File's Owner is an instance of your view controller subclass, we can **make the connection from the "view" outlet to the view**. You already know how to do this.

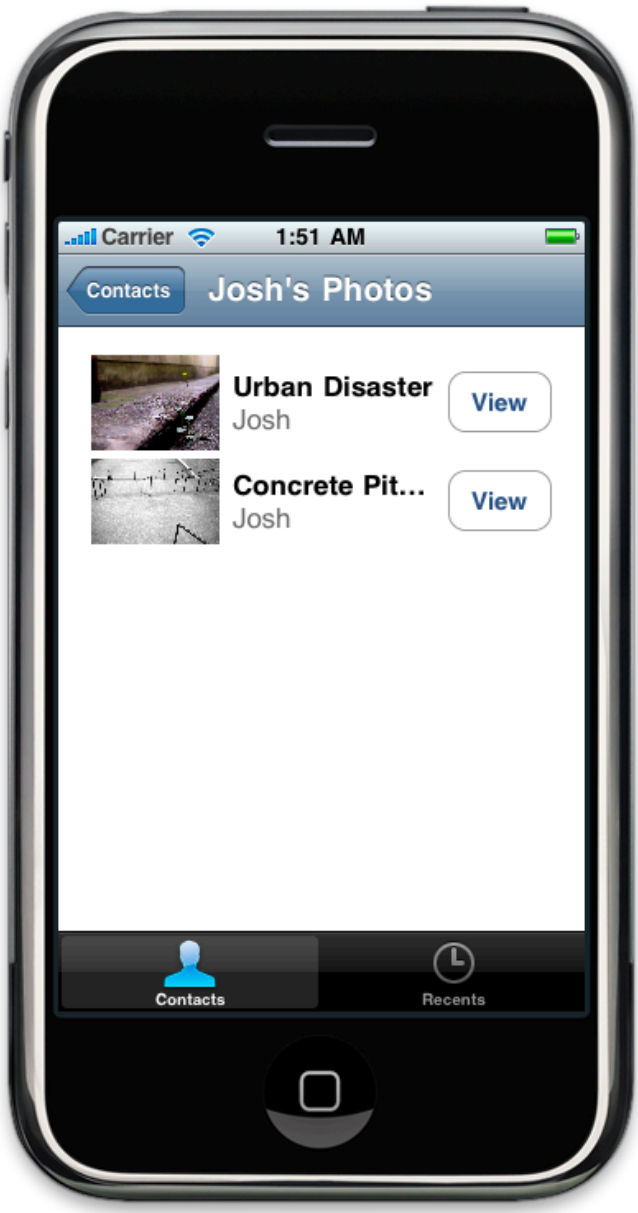
Next, we want to **create a PersonListViewController instance** and display it in our navigation stack. This code should go in your application delegate in the same place you create your navigation controller. Using the -initWithNibName:bundle: method, pass the name of your Interface Builder document as the first argument (you can leave out the file extension), and [NSBundle mainBundle] as the second argument.

Finally, we need to **push the view controller onto the navigation controller's stack** so that it's displayed. Now, you should be able to build & run from Xcode and see your list. At this point, it should look something like this:



The rest of the assignment, in brief...

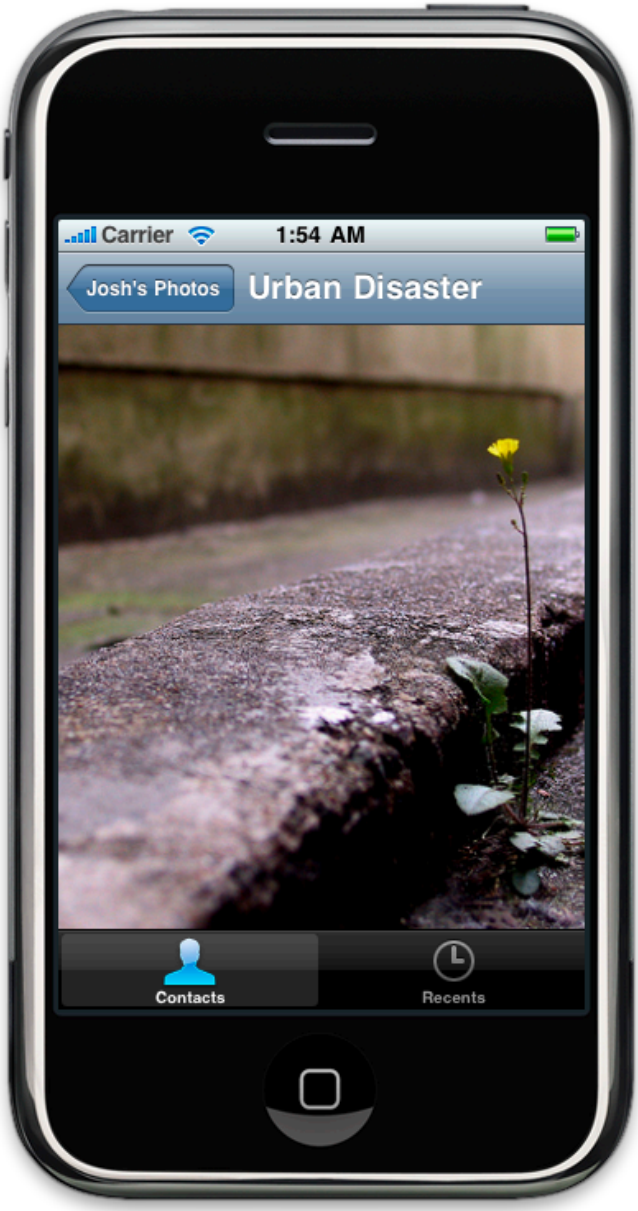
Create a second view controller subclass & NIB using the same steps as above. **When a button in your list of contacts is touched, push an instance of this second view controller**. You'll want to expose some properties so that it knows what to display. Make sure not to leak memory! The view should look something like this:



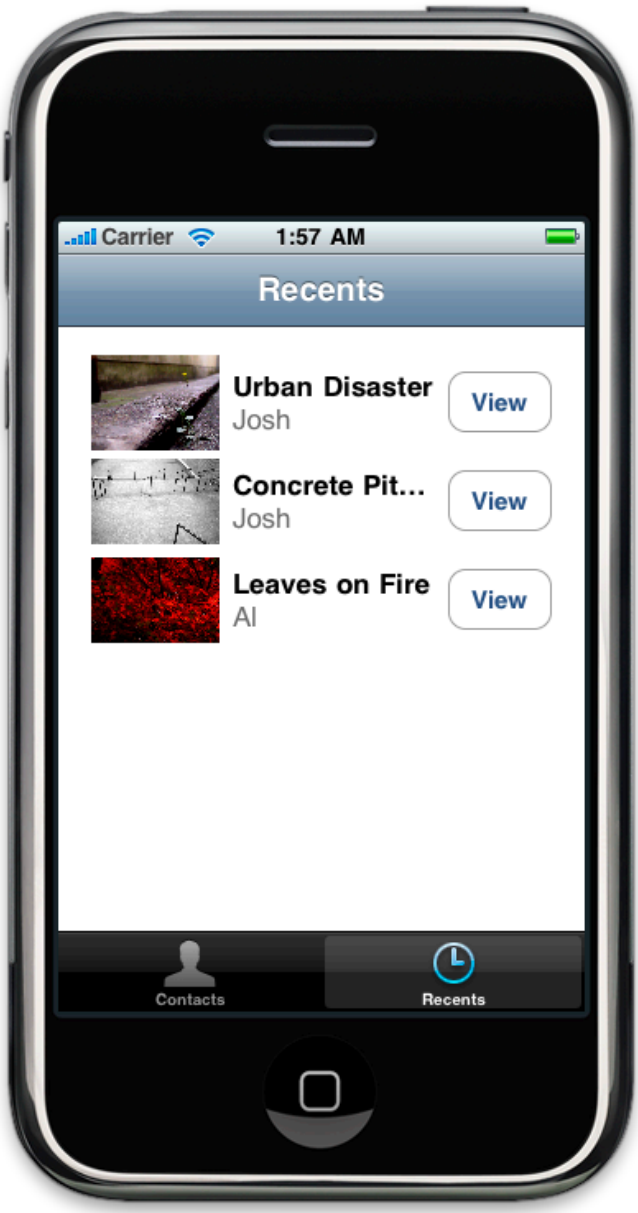
Remember that UIView has a hidden property you can use to prevent it from drawing. You can use this to hide some views if you have less images to display than views to display them.

In order to display a helpful title in the navigation bar when each view controller is being shown, you'll want to **set the title property for each view controller**. See the lecture slides for a hint on where to do this.

Create a final view controller subclass & NIB using the same steps as above. **When a button in your list of photos is touched, push an instance of this final view controller**. You'll want to expose some properties so that it knows what to display. Make sure not to leak memory! The view should look something like this:



Finally, we need to set up the Recents tab. We've already created all the view controller subclasses we need to implement this entire view. We can just **instantiate and push a photo list view controller onto the second navigation controller's stack** and set its list of photos to all of the available photos. This would probably be done as part of the initial application setup, back where we created the first view controller for the other navigation controller. At this point, it should look something like this:



Extra Credit

Here are some suggestions for enhancing this first iteration of Paparazzi.

- Add custom buttons on the left or right side of the navigation bar for the person list view controller. When pressed the button should provide some interesting feedback, like popping up an alert or animating your UI momentarily.
- Our Paparazzi screenshots are pretty dull. Spice yours up with some better colors and artwork.
- Think of a reason for a fourth view controller subclass, and push it onto the navigation stack in response to some user action. Make it load some data when it's about to appear & save the data when it's about to disappear.