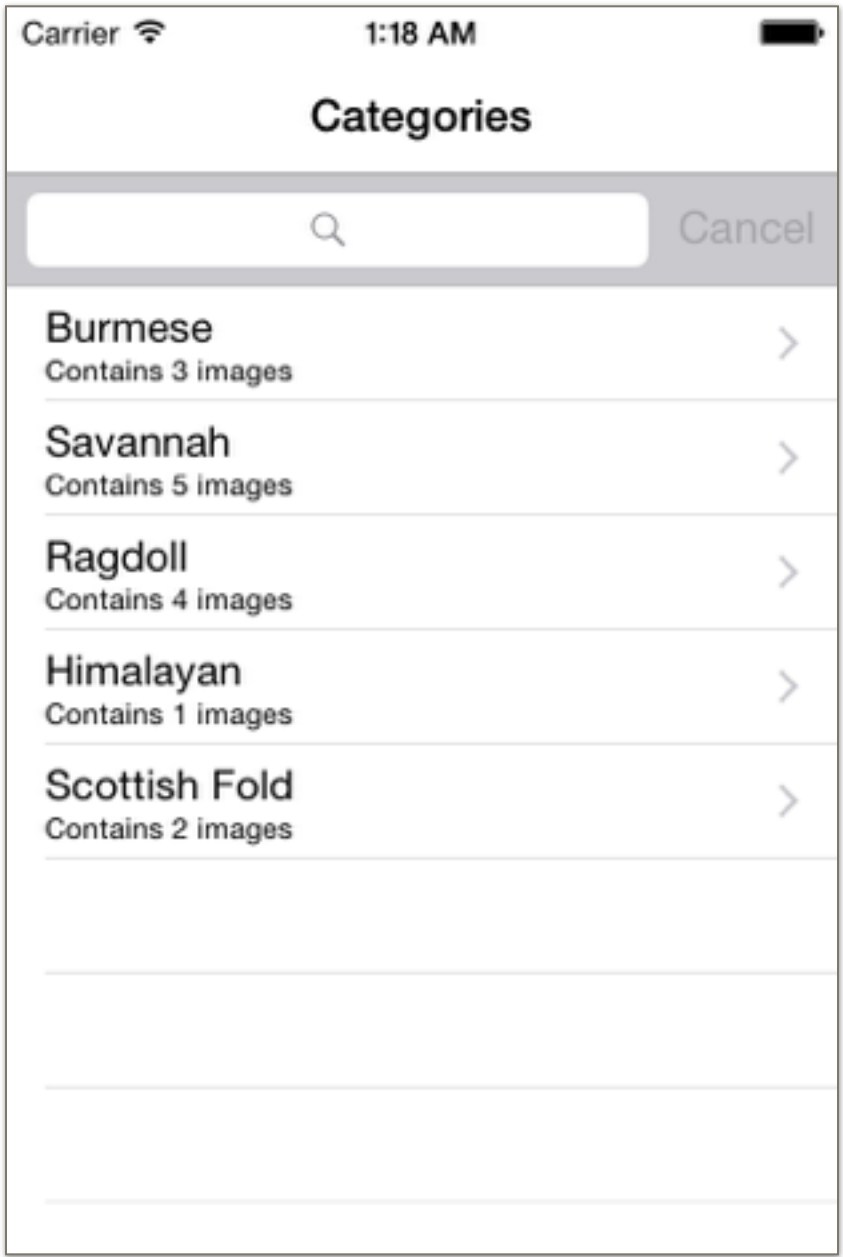
## Goal:

The goal of this assignment is to become more familiar with creating a rudimentary iOS application using custom UIViewController subclasses, Storyboards, SDK provided container view controllers, and SDK provided standard views to display content stored in plist form and in CoreData.

## Part 1 Deliverables:

**Important - The commit in your git repository that contains your implementation of Part 1 must be tagged "part1". See chapter 2.6 of the Pro Git book for details.**

A modified version of the provided base project which has two different custom `UIViewController` subclasses named `CategoryListViewController` and `CatImagesViewController`. The user interface (view) elements for these view controllers should be created in Main.storyboard. The initial view controller of the application should be a `UINavigationController`, and the `CategoryListViewController` should be presented initially.



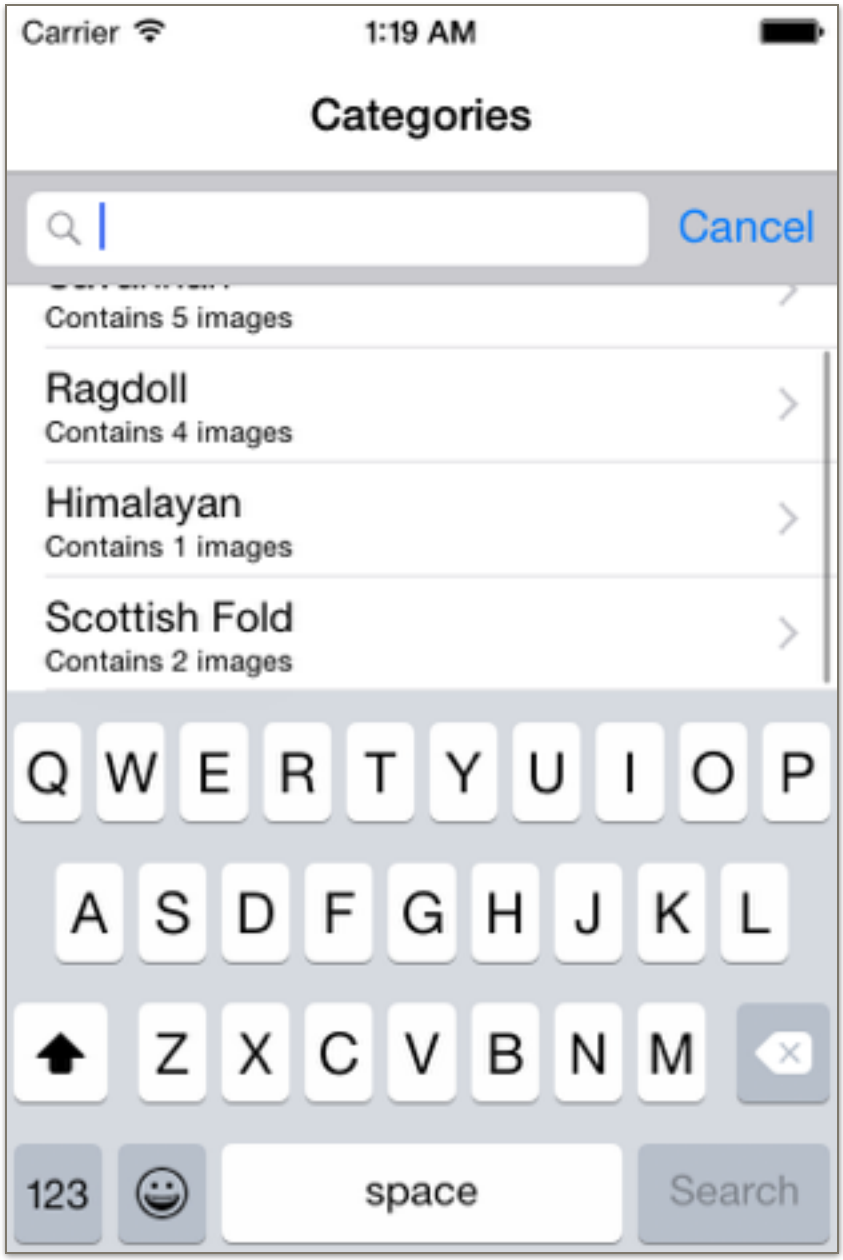
**CategoryListViewController**: This view controller should be presented inside of a `UINavigationController` instance. It's navigation item should have the title "Categories". The scene for this view controller should contain two subviews as depicted in the screenshot above. The first view is a `UISearchBar` view, which should be constrained to the left and right edges of the scene and to the top layout guide of the scene. The search bar should be enabled to show its cancel button. The second view is a `UITableView`, which should be constrained to the search bar, the left and right edges of the scene, and the bottom layout guide. The table view should be configured with a prototype cell of the "Subtitle" style, which should have an accessory type of "Disclosure Indicator" so that it shows the arrow on its right side, and a valid Reuse Identifier.

The `CategoryListViewController` will adopt the `UITableViewDataSource` and `UITableViewDelegate` protocols. Using the methods of these protocols, it will configure the table view to show all of the category titles and subtitles returned from the provided `CatService.sharedCatService().catCategories()` method as shown in the screenshot above.

When a row is selected in the table view, a storyboard segue should transition the user to the `CatImagesViewController` which is described below. The `CatImagesViewController` should be configured to show images from the category that was selected by the user in an override implementation of the `prepareForSegue(_:sender:)` method.

Additionally the `CategoryListViewController` will adopt the `UISearchBarDelegate` protocol. Using the methods of this protocol it will allow the user to dismiss the keyboard by tapping the Cancel button of the search bar if they have selected the search bar to bring the keyboard up (hint: you can dismiss the keyboard from a view controller by calling `self.view.endEditing(true)`). Note that if you are using the simulator you may have to toggle the software keyboard with the following menu command: Hardware menu -> Keyboard menu item -> Toggle Software Keyboard menu item. The CategoryListViewController does not need to perform any actual search functionality.

Lastly, the CategoryListViewController will register for notifications so that it can respond to the keyboard showing and hiding. In the code that handles these notifications, it can call the `adjustInsetsForWillShowKeyboardNotification(_:)` and `adjustInsetsForWillHideKeyboardNotification(_:)` methods that are provided for you as an extension on `UITableView` in the project. These methods will adjust the content insets to allow the content to scroll out from under the keyboard, as depicted in the screenshot below.




**CatImagesViewController**: This view controller should be pushed onto the `UINavigationController` instance when a cell in the `CategoriesListViewController` scene's table view is selected by the user using a segue configured in the storyboard. The navigation item for this view controller should have the title "Cat Images". The scene for this view controller should contain a collection view, as depicted in the screenshot below. Finally, the back button provided automatically by the `UINavigationController` should not be broken and the `CategoryListViewController` should reset its selection before it appears on screen again.

The `CatImagesViewController` will adopt the `UICollectionViewDataSource` and `UICollectionViewDelegate` protocols. Using the methods of these protocols, it will configure the collection view to show all of the cat images referenced by the array of image names provided (hint: `UIImage(named: <%imageName%>)` can be used to load the images) by `CatService.sharedCatService().imageNamesForCategoryAtIndex(<#category index#>)`, where <#category index#> is the index of the category that was configured during the segue.

A custom `UICollectionViewCell` will be necessary for this implementation. The collection view will need to contain an `@IBOutlet` to a `UIImageView` which will be used to display the cat image. Note that this is necessary because the views inside a prototype cell in a storyboard cannot be connected to outlets of the view controller (this makes sense, because the view controller has a single outlet but may create many cells).

**CatService:** The service class that is used to retrieve the model data for the application will need to be finished. The provided class includes a small amount of functionality (implementation of the singleton pattern and some simple code to load the plist containing the cat data). To support the function of the view controllers the `catCategories()` and `imageNamesForCategoryAtIndex(_:)` methods will need to be implemented. The `catCategories()` method should return an array of tuples that each contain a category name and a description of the category (as seen in the first screenshot above in each cell). The `imageNamesForCategoryAtIndex(_:)` method should return an array of image names for the category corresponding to the index parameter.

## Part 2 Deliverables:

Part 2 of the assignment is to modify the app to take the plist data and store it in `CoreData`. For the purposes of this assignment the included `CoreDataService` should be used.

**Model Entities**: Two `CoreData` entities should be created: `Category` and `Image`. The `Category` entity should have a required attribute of type `String` to store the category title and a to-many optional relationship to `Image`. The `Image` entity should have a required attribute of type `String` to store the image name, a required attribute of type `Integer 32` to store an ordering index that will be used to sort, and a to-one required inverse relationship to `Category`.

Subclasses of `NSManagedObject` should be created for these entities. Each subclass should implement the `NamedEntity` protocol from the provided `CoreDataService` framework. The subclass for the `Category` entity should also implement a computed property that returns the subtitle for the category.

**CatService**: The `CatService` implementation should be modified such that it no longer stores the plist content, and instead relies on `CoreData`.

The `init()` should access the `NSManagedObjectContext` provided by the `CoreDataService` singleton (hint: this can be accomplished with the code `CoreDataService.sharedCoreDataService.mainQueueContext`) convert the plist data into the entities described above, and save them (hint: It's important to call the

`save()` method on the context being used and also the `saveRootContext(_:)` method of the `CoreDataService`).

The `catCategories()` and `imageNamesForCategoryAtIndex(_:)` methods should both be converted to return an `NSFetchedResultsController` created with an appropriate `NSFetchRequest` instead of their previous return types. Additionally, `imageNamesForCategoryAtIndex(_:)` should be renamed to `imageNamesForCategory(_:)` and have its parameter type changed to accept the subclass of `NSManagedObject` that was created to represent `Category` as described above. The `NSFetchedResultsController` for the cat categories should sort by the title attribute and the `NSFetchedResultsController` for the image names should sort by the ordering attribute.

**View Controllers:** Both view controller implementations should be modified to utilize the `NSFetchedResultsController` instance returned by the `CatService` instead of the previous arrays of data. Additionally, the CategoryListViewController should be modified to pass the selected Category instance to the CatImagesViewController during prepareForSegue(_:sender:)