UIKit Assignment

Goal:

Create a simple application that contains a number of views and some simple application logic to gain a basic understand of a UIKit based iOS application.

Deliverables:

Follow the instructions to implement your application. Make intermittent commits as you feel is appropriate (there are suggestions for good commit points in these instructions but feel free to make more commits as you see fit). When you have finished your work make sure that you have pushed your commits to origin and they appear there as expected when a new copy of the repository is cloned.

Notes:

These instructions refer to the objects library when working in the storyboard. The library is a window in Xcode that shows content you can add to whatever you have open in the editor. When you have a storyboard open the first tab in the library will be the objects library. The Objects library contains the various elements that are available for you to add to your storyboard canvas. It is accessed with the Show Library menu item in the View menu.

In numerous places these instructions refer to navigators and inspectors. Navigators are shown to the left of the editor in Xcode and include things like the Project navigator and the Test navigator. You can always change which navigator is showing using the menu items in the Navigators submenu in the View menu in Xcode (you can also use the tabs at the top of the navigator panel if it is not hidden).

Similarly inspectors are shown on the right of the editor in Xcode. Like with navigators you can select which inspector is shown using the menu items in the Inspectors submenu in the View menu in Xcode (or the tabs along the top of the inspector panel if it is not hidden). Which inspectors are available will change based on what you have selected in the Xcode user interface.

There are a few places these instructions refer to the storyboard editor's Document Outline. This is hidden by default. You can show and hide this using the Document Outline option in the Editor menu.

In these instructions strings appearing in quotes should be used without quotes in your application. The quotes are here for clarity and formatting purposes.

Instructions:

MainViewController

The `MainViewController` class contains the logic necessary to update its label and support proper transitions to the other views in the app. Perform these steps to create your MainViewController.swift file:

1. Create a new file named MainViewController.swift in the Sources/UIKitAssignment/Controller group in the project navigator.

Open the MainViewController.swift file and perform the following steps:

1. Import UIKit so that the types and other symbols declared in UIKit are available to you
2. Create a class named `MainViewController` that is a subclass of `UIViewController`.

Open the Main.storyboard file and perform the following steps to setup the main view:

1. Add a new scene to the storyboard by dragging a view controller from the objects library onto the storyboard canvas.
2. Select the new view controller and change its type to `MainViewController` in the identity inspector. The placeholder for this should say `UIViewController`. If it does not you are probably changing the wrong object in the storyboard canvas.
3. With the view controller still selected choose Embed In -> Navigation Controller from the Editor menu.
4. Select the new navigation controller and enable the is initial view controller checkbox in the attributes inspector.
5. Change the title of your view controller's navigation item to "Main View".
6. Drag a bar button item from the objects library onto the right side of your view controller's navigation item.
7. Change the title of the new bar button item to "Show Image" in the attributes inspector.
8. Drag a button from the objects library and place it in the center of your view controller's view.
9. Change the new button's title to "Show Details" in the attributes inspector.
10. Add constraints to the new button so that it will stay in the center of its container view.
11. Drag a label from the objects library and place it below your button.
12. Make the following changes to your label in the attributes inspector:
    a. Change its text to "The detail action has been performed 0 times".
    b. Change its lines value to 0 so that it can be multiple lines (0 means it can be as many lines as it needs to be).
    c. Change its text alignment that it centers its text.
13. Add constraints to your label so that:
    a. It is horizontally centered.
    b. It is below the button by the standard spacing (8 points).
    c. It cannot get closer than 40 points to the leading and trailing edges of its container view.

At this point you can run your application in the simulator and you should see your main view as you have designed it in the storyboard. The buttons will respond to being tapped but will not do anything.

If you are happy with your implementation so far, now might be a good time to commit your work.

DetailViewController

The `DetailViewController` class contains the logic necessary for the function of the detail view that is shown with a show segue from the main view when the user taps on the show details button. Perform these steps to create your DetailViewController.swift file:

1. Create a new file named DetailViewController.swift in the Sources/UIKitAssignment/Controller group in your project navigator in Xcode

Open the DetailViewController.swift file and perform the following steps:

1. Import UIKit so that the types and other symbols declared in UIKit are available to you
2. Create a class named `DetailViewController` that is a subclass of `UIViewController`.

Open the Main.storyboard file and perform the following steps to setup the detail view:

1. Add a new scene to the storyboard by dragging a view controller from the objects library onto the storyboard canvas to the right of your main view controller.
2. Select the new view controller and change its type to `DetailViewController` in the identity inspector. The placeholder for this should say `UIViewController`. If it does not you are probably changing the wrong object in the storyboard canvas.
3. Select the show details button in your main view.
4. From the connections inspector make a connection from the triggered segues action to the detail view scene in your storyboard. Select show as the segue kind.
5. Change the title of your view controller's navigation item to "Detail View".
6. Drag a label from the objects library and place it in the center of your view controller's view.
7. Make the following changes to your label in the attributes inspector:
    a. Change its text to "Presented on …".
    b. Change its lines value to 0 so that it can be multiple lines.
    c. Change its text alignment so that it centers its text.
8. Add constraints to your label so that:
    a. It will stay in the center of its container view.
    b. It cannot get closer than 40 points to the leading and trailing edges of its container view.
9. Drag a button from the objects library and place it below your label.
10. Change the new button's title to "Perform Action" in the attributes inspector.
11. Add constraints to your button so that:
    a. It is horizontally centered.
    b. It is below the label by the standard spacing (8 points).

Open the DetailViewController.swift file and perform the following steps to implement logic needed to properly set the text of the label:

1. Add an `@IBOutlet` property for your label.
2. Override the `viewWillAppear(_ animated: Bool)` method and set the label's text to "Presented on <current date & time>" where <current date & time> is the current date and time. Make sure you call the super implementation in your override implementation.

Hint: You can create a constant containing the current date and time with the following line of code:

```
let currentTimeValue = DateFormatter.localizedString(from: Date(), dateStyle: .medium, timeStyle: .medium)
```

Open the Main.storyboard file and perform the following steps to connect the label in your canvas to the outlet:

1. Select your detail view controller (be careful to select the view controller itself with the icon on the left side of the header bar above its view or using the storyboard's document outline view).
2. From the connections inspector make a connection from your label outlet to the label in your detail view.

At this point you can run your application in the simulator and you should see your main view as you did before. When you tap the show details button it should present your detail view using a standard navigation view transition animation (the detail view will slide in from the right covering the main view). The detail view's label should show the correct date and time and the button to navigate back to the main view should function. If you navigate to the detail view again, the time should be updated appropriately. The perform action button on the detail will respond to being tapped but nothing will happen.

If you are happy with your implementation so far, now might be a good time to commit your work.

DetailViewControllerDelegate

We will use the delegate pattern as described in lecture to inform the main view when the detail view's button has been tapped so that it can update its label:

1. Create a new group named Protocol in the existing Sources/UIKitAssignment/Controller group in your project navigator in Xcode
2. Create a new file named DetailViewControllerDelegate.swift in the group you created above

Open the DetailViewControllerDelegate.swift file and perform the following steps to implement the delegate protocol:

1. Add a protocol named `DetailViewControllerDelegate`. This protocol should have a parent protocol of the `AnyObject` protocol so that only classes can conform to it.
2. Declare a method in this protocol named `detailViewControllerDidPerformAction` that takes a single parameter of type `DetailViewController`. This parameter should have no label (i.e., use the `_` keyword for the label).

Open the DetailViewController.swift file and perform the following steps to setup the `DetailViewController` so that it has a delegate property and interacts with its delegate:

1. Add a property named `delegate` that is of type `DetailViewControllerDelegate?`. To avoid a memory leak, this property should be a weak reference (this is done by putting the `weak` keyword before the `var` keyword on the property declaration). This prevents a cycle of strong references that would keep the objects from being deallocated.
2. Add an `@IBAction` method that calls the `detailViewControllerDidPerformAction` method on the `delegate` property if it is not `nil`. You should use appropriate optional syntax to safely call this method on the delegate.

Open the MainViewController.swift file and perform the following steps to make it conform to the delegate protocol and update its label:

1. Change the class declaration to declare that `MainViewController` conforms to `DetailViewControllerDelegate` that you defined above.
2. Add an `@IBOutlet` property for the label in your main view.

3. Implement the `detailViewControllerDidPerformAction` method that you declared in your delegate protocol. This function should do the following:
   a. Keep track of how many times the action has been performed.
   b. Update your label text to be "The detail action has been performed <n> times" where <n> is the number of times the action has been performed. If n is 1 it should say "time" instead of "times".

Next you will need to configure the `DetailViewController` that is created during the show segue so that the `MainViewController` is its delegate. To do this you will utilize an `@IBSegueAction` method that will allow you to manually create and configure the `DetailViewController` during the screen transition. Still in the MainViewController.swift file perform the following steps:

1. Implement an `@IBSegueAction` method that creates the `DetailViewController` instance and sets its delegate property to the `MainViewController` instance (i.e., `self`). You can base your implementation on this example `@IBSegueAction` method:

```
@IBSegueAction private func createExampleViewController(_ coder:
NSCoder) -> ExampleViewController? {
      let viewController = ExampleViewController(coder: coder)

      // Configure viewController here…

      return viewController
}
```

The last steps that are required are to setup the storyboard so that the main view label is connected to the outlet created above, the segue calls the new `@IBSegueAction` method created above, and the perform action buttons calls the new `@IBAction` method created above. Open the Main.storyboard file and perform the following steps:

1. Select your main view controller (be careful to select the view controller itself with the icon on the left side of the header bar above its view or using the storyboard's document outline view).
2. From the connections inspector make a connection from your label outlet to the label in your main view.
3. Select the segue arrow between your main view and your detail view.
4. From the connections inspector make a connection from the segue actions instantiation action to the method you created above on your `MainViewController`.
5. Select the perform action button on the detail view.
6. From the connections inspector make a connection from the touch up inside sent events action to detail view controller's IBAction that you made above.

At this point you can run your application in the simulator and you should see your main view and detail view as you did before. Tapping the perform action button on the detail view should now trigger an appropriate update to your main view.

If you are happy with your implementation so far, now might be a good time to commit your work.

Modal View

The modal view will be designed entirely in the storyboard and does not require a `UIViewController` subclass to function.

The modal view will display an image of your choosing (please pick a “safe for work” image). Open Assets.xcassets and perform the following steps to make the image available in your application:

1. Download the image of your choice.
2. Drag the image onto the asset catalog editor.
3. Give the image a reasonable name.

Open the Main.storyboard file and perform the following steps to setup the modal view:

1. Add a new scene to the storyboard by dragging a view controller from the objects library onto the storyboard canvas above your detail view.
2. With the view controller still selected choose Embed In -> Navigation Controller from the Editor menu.
3. Change the title of your view controller’s navigation item to “Modal View”.
4. Drag a bar button item from the objects library onto the right side of your view controller’s navigation item.
5. Change the title of the new bar button item to “Done” in the attributes inspector.
6. Drag an image view from the objects library and place it in the center of your view controller’s view.
7. Change the image for your image view in the attributes inspector to the image you added to the asset catalog above.
8. Add constraints to the new image view so that it will fill its container view.

Still in the Main.storyboard file perform the following steps to create a segue that shows the modal view when the show image button is tapped in the main view:

1. Select the show image bar button item in your main view.
2. From the connections inspector make a connection from the triggered segues action to the navigation view controller that contains the modal view scene in your storyboard. Select present modally as the segue kind.

Open the MainViewController.swift file and perform the following steps to implement the logic needed to setup an exit segue `@IBAction` for the modal view’s done bar button item:

1. An exit segue `@IBAction` method is an `@IBAction` method that has a sender with the type `UIStoryboardSegue`. Create an exit segue `@IBAction` method. The body of an exit segue `@IBAction` can contain logic if needed, but it does not need to. For this step an empty method body is appropriate.

Open the Main.storyboard file and perform the following steps to connect the done bar button item in your modal view to the exit segue IBAction:

1. Select the exit item for your modal view scene (this can be found on the right side of the scene’s title bar or in the storyboard’s document outline view).
2. In the connections inspector you should see the exit segue `@IBAction` that you created above. Make a connection from this to your done button’s triggered segues action.

At this point you can run your application in the simulator and you should see your main view and detail view as you did before. When you tap the show image button it should present your

modal view using a standard modal transition animation (the modal view will slide in from the bottom covering the main view and its navigation view). The modal view's image should show the image you set and the done bar button item should dismiss the model view.

If you are happy with your implementation so far, now might be a good time to commit your work.

Validation with UITests

The project includes a handful of UI tests that validate the functionality of your app's user interface. In order for these tests to function you will need to configure accessibility identifiers on the two labels in your storyboard. Open the Main.storyboard file and perform the following steps to setup the accessibility identifiers:

1. Select the label in your main view
2. Set the accessibility identifier to "DetailActionCount" using the identity inspector
3. Select the label in your detail view
4. Set the accessibility identifier to "AppearedTimestamp" using the identity inspector

Run the UI tests by selecting Test from the Product menu. If you have test failures you can inspect them using the test navigator. Please note that these tests rely on you using the exact text as specified in the assignment and will fail if it does not exactly match (it is case sensitive).

Manual Validation

While the UITests validate the functionality of the app, they do not validate the appearance of the app. You will need to run your app on multiple device simulators and in different orientations and validate that the constraints you have created work properly in all cases. If they do not you should go back to your storyboard and examine the constraints you have in place and make any adjustments as needed. The assignment repository includes a Screenshots folder that contains a selection of screenshots you can compare with your implementation (obviously the image will likely be different).