ATLS 4120/5120: Mobile Application Development
Week 6: Multiple View Controllers

Although we could build this in one view, in this app we're going to learn how to handle multiple views as well as store model data used in both views.

Model-View-Controller(MVC)
(slide)
The MVC architecture is the pattern used throughout iOS and Android

- Model
  - classes that manage your data and often map to real-world objects.
  - Models should be independent of the view and the controller.
  - Model classes should be reusable.
  - Should be UI independent
- View
  - all items for the user interface(objects in IB)
- Controller
  - View controllers act as the glue connecting the view and the model
  - View controller classes need access to the view and the model
  - Controllers are usually paired with a single view that they manage, that's where ViewController comes from
  - Apps that have multiple views need multiple view controllers
- The goal of MVC is to have any object be in only one of these categories.
  - These categories should never overlap.
- Ensures reusability

Multiple Controllers

Modal (slide)
(wallet, stocks)

- A modal view controller presents another view on top of the current view
- In utility apps the main view is dedicated to a single task
- A second view is used for configuration settings or to provide more information.

Tab bar (slide)
(clock, phone)

- A tab bar controller organizes views in a tabbed list
- present different types of information
- present the same information using a completely different style of interface
- Data is not passed between tabs
- Don't confuse this with a toolbar(safari) which holds buttons for actions

Table view/Collection View w/Navigation Controller (slide)
(settings, mail, contacts, photos)

- A navigation controller organizes views in an hierarchy, often a table view
- Selecting an item drills down into more detail
- Data flows down and back up
- One of the most commonly used in iOS

Split View (slide)
(mail, safari iPad)

- A split view has the navigation in the left column and the content on the right.
- First introduced for the iPad instead of the table view due to screen size
- Now also supposed on the Plus size iPhone

View Controllers

- The primary view controller for an app is called the root controller.
  - Its job is to present views based on the user's input
- Each view is typically controlled by its own controller class.
- A view controller and its view are called a scene.

Storyboards (slide)

- Storyboards provide a single visual workspace to build apps with multiple views
- Easily let's you define transitions between views
- Provides a good conceptual overview of all the scenes in an app
- Arrow on the left points to the initial view controller to be loaded when the app loads this storyboard.
- Change the zoom level to see the entire flow of your app through the storyboard

Segues

Segues define how one scene transitions to another scene.

- Makes it easier to visually see and edit the entire flow of your app.
- A segue is called in a controller whenever a controller's view is about to be replaced by another controller's view
- Storyboard segue types (slide)
  - Show
    - Shows a view controller in a navigation controller
    - Pushes a view controller on a navigation controller's stack
  - Show detail
    - Replaces the controller in a split view or table view with a new controller full screen
      - For example on a master Detail project on an iphone it would move from the table view onto the detail view and takes up the full screen, where as on the ipad it would still show the table view as well as the detail view.
  - Present modally
    - One view controller presents another view controller
  - Present as popover
    - Presents a popover controller
  - Custom
    - A segue that uses a custom class
  - Don't use deprecated types

UIStoryboardSegue class https://developer.apple.com/documentation/uikit/uistoryboardsegue

- **prepare(for:sender:)** in the UIViewController class is called whenever a segue is being activated but before the transition occurs
- The **destination** property tells us which VC is about to be displayed
- The **source** property tells us which VC is about to be removed from the display
- The **identifier** property is a String that can be used to identify the segue

Unwind Segues (slide)
Unwind segues allow you to define the relationship for reversing a segue transition

- Define an unwind method in your destination VC to be called by the unwind segue
  - Must be an **IBAction** method
  - Must take a parameter of type **UIStoryboardSegue**
  - Using this method format will tell Xcode that this is a method that can be *unwound* to.
- Connect the element in the scene to the Exit icon and select the unwind method
- It might seem as though the obvious choice is to simply implement a segue back. Instead of returning the original instance of scene 1, however, this would create an entirely new instance of the StoryboardViewController class. If a user were to perform this transition repeatedly the application would continue to use more memory and would eventually be terminated by the operating system.

Passing Data
To pass data from one view controller to another during transition

- Create the segue and give it a unique identifier
- Implement **prepare(for:sender:)**
- Use the **destination** property to create an object to access the destination VC
- Assign the data to properties of the destination VC
  - You can't assign the data directly to UI objects because they might not exist yet.
- Use the unwind method or **viewWillAppear()** to refresh any needed elements in the destination view

Favorites
Create a new project using the single view template called favorites.
Add a label on top that says Favorites.

Put a toolbar at the bottom. Add a flexible space bar to move the button over to the right.
Give the button the name Info.
More on toolbars https://developer.apple.com/ios/human-interface-guidelines/bars/toolbars/
We want to keep the toolbar at the bottom of the view and going all the way across the bottom no matter what size the view has so create pin constraints for leading(-16), trailing(-16), and bottom(0).

Now let's add a new scene to the storyboard. Drag a view controller object from the Object Library panel onto the canvas to the right of your first scene. It automatically gets selected.
Go into the Identity Inspector and see that its class is listed as UIViewController. But if we want to write any code for this view, we will need to create our own class to control it.
File | New | File
iOS | Source | Cocoa Touch class named Scene2ViewContoller.
Subclass of: UIViewController.
Don't select Also create XIB file
Make sure the language is Swift.
Save it in your Favorites project.
This creates the Scene2ViewContoller.swift
In the Project Navigator you can drag it into favorites if you want to remain organized.
Go back into the MainStoryboard and select Scene2 View Controller. In the identity inspector change the class to be Scene2ViewContoller.
This is a very important step as it determines that the Scene2ViewContoller will control the second view.

Now add a label to the top of this scene that says My Favorites.
Add a toolbar, a flexible space bar to move the button to the right, and a button labeled Done.
Add the pin constraint for leading, trailing, and bottom, same as before.

We can have more scenes but for today we're just going to have two.
Now let's configure the segues between these two scenes.
Select the first scene. To setup a segue, hold down the Ctrl key, click on the toolbar button (make sure you get the bar button item and not the toolbar, use the document hierarchy) and drag the resulting line to the second scene. When you let go of the mouse button a menu will appear. Select the present modally menu option to establish the segue. Click on the segue to see that it's present modally. Look in the attributes inspector to see that Animates is checked and there are 4 other transitions other than default. Try these later.
Go into the attributes inspector to give it the identifier favInfo.

Save and run and you'll see that the Info button takes you to the second scene, but you can't go back yet. Let's set that up now.

Although you might think you just set up another segue from the done button back to scene1, you should not do it that way. Instead, set up an unwind action.
First you need to add a method to scene1 (destination view controller).
In ViewController.swift add the method

```
@IBAction func unwindSegue(_ segue:UIStoryboardSegue){
}
```

You can name this method anything you want but remember it must be an IBAction method and take in a UIStoryboardSegue parameter. Using this method format will tell Xcode that this is a method that can be *unwound* to.

It's ok that it's empty for now. We'll put code in here that we want to run when we return from scene 2 to 1.

The next step is to establish the unwind segue. Go back into MainStoryboard and select scene 2. ctrl-click and drag from the Done button (make sure you get the bar button item and not the toolbar, use the document hierarchy) to the "exit" icon (the orange button with the white square and arrow) at the top of the view. Release the line and select the unwindSegue method from the resulting menu.
Click on that segue and go into the attributes inspector to give it the identifier doneFavs.
Run your app and see that now the Done button takes you back to scene 1.

Now let's get some data and pass it between scenes.

In scene 1 add two labels, make them blank, and create outlets named bookLabel and authorLabel respectively. Make sure you make these connections to the right class – ViewController.

In scene 2 add a label for favorite book and a text field next to it.
Do the same for author.

Connect the textfields as outlets named userBook and userAuthor respectively. Make sure you make these connections to the right class – Scene2ViewController.
We also want to make sure the keyboard is dismissed for both textfields when the user hits return.

In Scene2ViewContoller.swift adopt the `UITextFieldDelegate` protocol

```
class Scene2ViewController: UIViewController, UITextFieldDelegate
```

Set the textfield's delegates

```
    override func viewDidLoad() {
        userBook.delegate=self
        userAuthor.delegate=self
        super.viewDidLoad()
    }
```

Implement `textFieldShouldReturn(_:)`

```
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }
```

Model data

Now we're going to create the Favorite model class to keep track of the data.
File | New File
iOS Source Swift file
Name it Favorite
Add it to the favorites folder and make sure the target is checked as well.
Create
Let's define the Favorite class to store the user's favorite book and author
Edit Favorite.swift

```
class Favorite {
    var favBook : String?
    var favAuthor : String?
}
```

Swift will automatically create an initializer method for us and both Strings will have the value nil or we can create our own initializer method if we want it to have an initial value. (in the class brackets)

```
    init(){
        favBook="Your favorite book"
        favAuthor="Your favorite author"
    }
```

I'm not going to do this since I want the user to enter their favorites.

Now let's create an instance for the Favorite class to ViewController.swift called user

```
    var user=Favorite()
```

In Scene2ViewController.swift we need to save the data the user enters back to our model. The prepare(for:sender:) method is called automatically when a segue is about to appear.

```
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "doneFavs"{
            let scene1ViewController = segue.destination as! ViewController
            //check to see that text was entered in the textfields
            if userBook.text!.isEmpty == false{
                scene1ViewController.user.favBook=userBook.text
            }
            if userAuthor.text!.isEmpty == false{
```

```
                scene1ViewController.user.favAuthor=userAuthor.text
            }
        }
    }
```

In ViewController.swift we need to update our labels when we return from scene 2.

```
    @IBAction func unwindSegue (_ segue:UIStoryboardSegue){
        bookLabel.text=user.favBook
        authorLabel.text=user.favAuthor
    }
```

We also have to cast the destination view controller as the ViewController class so we can access its variables (and methods).
You must assign the data to objects/variables in the destination view controller that you know already exist. You can't assign data directly to UI objects in the destination view controller because they might not be loaded.

Constraints
Scene 1:
All labels align horizontal center
Favorites label top space to superview
First label top space to Favorites label
Bottom label top space to first label label
Or put them in a vertical stack view and space accordingly.

Scene 2:
My Favorites label Align Center X, top space to top layout.
Favorite book label top and textfield embed in stack view
Top space to My Favorites label
Pin leading and trailing to superview.
Favorite author label and textfield embed in stack view
Top space to book stack view
Pin leading and trailing to superview.

I also added a pin for the textfields to have equal width because I thought it looked better.