Local And Push Notification

<https://thomashanning.com/push-notifications-local-notifications-tutorial/>

Collection View

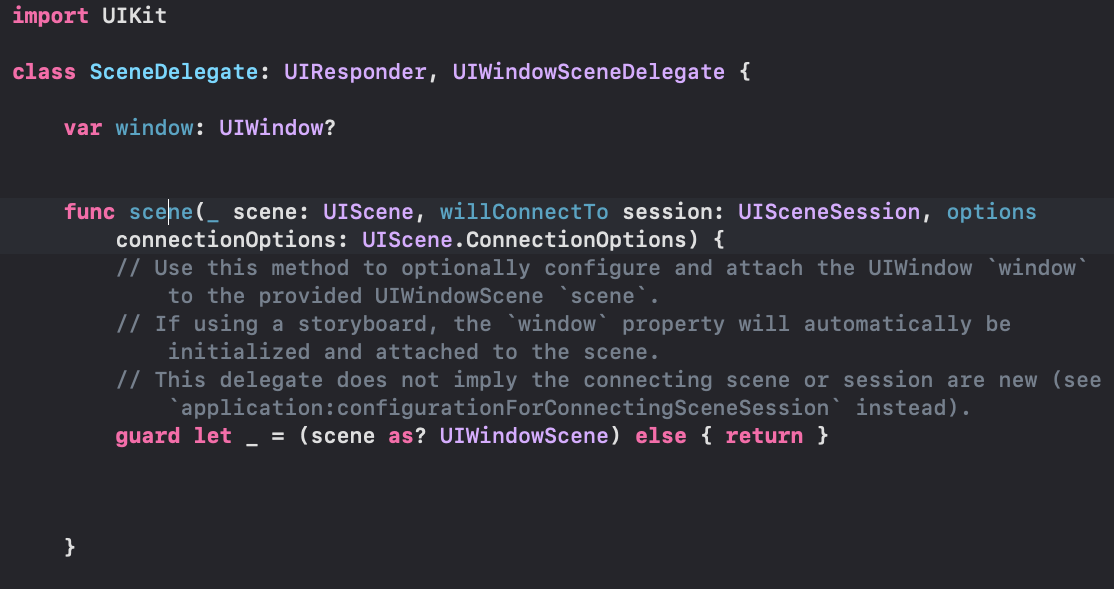
<https://developer.apple.com/documentation/uikit/uicollectionview>

IBOutlet Collection

<https://stackoverflow.com/questions/24052459/swift-iboutletcollection-equivalent>

**Programmatic segues + Navigation Controller**

Now that we have learned how to embed our root view controller into a navigation controller using the storyboard, let's see how we can do this programmatically (using no storyboards, just code). It can be a bit more complex but when working on a large codebase with storyboards can cause a lot of issues on Github so most iOS developers prefer programmatic navigation. Before we learn how to do this we need to briefly learn about the scene delegate. According to the Apple Documentation, the SceneDelegate is responsible for "The core methods you use to respond to life-cycle events occurring within a scene." It has 6 methods we will be focusing on the first one. To learn more about the other methods check out: [UISceneDelegate Documentation](https://developer.apple.com/documentation/uikit/uiscenedelegate" \t "_blank)

 This method tells the delegate about the addition of a scene to the app. Here is where our scene starts and where our first view controller is loaded from. Here we will add code to embed our ViewcontrollerA inside a navigation controller.

Now let’s finally embed this ViewControllerA inside a navigation controller.

func scene(\_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {

//Let's save our scene instead of just leaving it as \_

guard let appScene = (scene as? UIWindowScene) else {return}

//Here we are making our UIWindow's frame to fit the full

screen of the scene which is the phone screen

self.window = UIWindow(frame:appScene.coordinateSpace.bounds)

//Now we are making our ViewControllerA

let mainView = ViewControllerA()

//This code is where we are embedding our ViewControllerA

into a Navigation

let navigationController = UINavigationController(rootViewController: mainView)

//Now we make our window's windowScene the same as out

appscene.

self.window?.windowScene = appScene

//Finally we make the window's rootView the new

navigationController with the ViewControllerA inside of it

self.window?.rootViewController = navigationController

self.window?.makeKeyAndVisible()

}

We’re still not done. If we run this code on the simulator you’ll notice that we get a black screen. Why is that? Well since we only instantiated the ViewControllerA programmatically without giving it any properties that’s what is happening. So let’s go inside our ViewControllerA file and give it a white background and a title.

import UIKit

class ViewControllerA: UIViewController {

override func viewDidLoad() {

super.viewDidLoad()

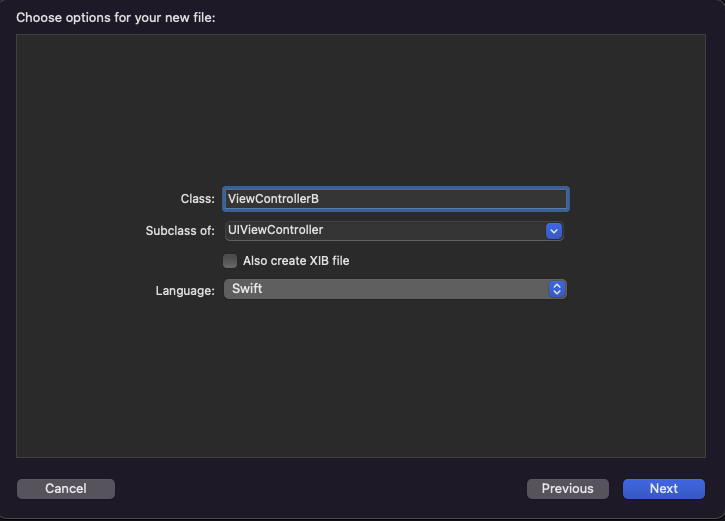
self.view.backgroundColor = .white

self.title = "ViewController A"

}

}

**IMPORTANT NOTE:** if you instantiate a view programmatically without the storyboard you will not be able to use storyboards to layout your UI. If you wish to instantiate a view with a storyboard you will have to use : self.storyboard?.instantiateViewController(withIdentifier: "ViewControllerName") as! ViewControllerName More on this in the later sections Now when we run our simulator we see our first view!! Now we’ve set up our root view embedded inside a navigation controller we can start creating new view controllers and navigating to them programmatically. Let’s make a new ViewControllerB file



Now let’s add a button in our ViewControllerA that will show the ViewControllerB when it’s pressed.

import UIKit

class ViewControllerA: UIViewController {

override func viewDidLoad() {

super.viewDidLoad()

self.view.backgroundColor = .white

self.title = "ViewController A"

//Set up next Button

setUpNavigationButton()

}

//Adding a button to the right side of the navigation bar

func setUpNavigationButton(){

// the action parameter takes a #selector() that takes an @objc

func that will execute when the button is tapped

self.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "Next", style: .plain, target: self, action: #selector(nextButtonTapped))

}

@objc func nextButtonTapped(){

print("Show ViewController B ")

//let's make ViewControllerB

let viewB = ViewControllerB()

//Now let's call our navigation controller to push and show this

view when the button is clicked

self.navigationController?.pushViewController(viewB, animated: true)

}

}

Don’t forget to make the background white and give a title to our ViewControllerB import UIKit

class ViewControllerB: UIViewController {

override func viewDidLoad() {

super.viewDidLoad()

self.view.backgroundColor = .white

self.title = "ViewController B"

}

}

**Passing data programmatically**

Passing data programmatically is essentially the same as when we use the storyboard. The main difference is that we don’t need to use the prepareForSegue function. Wherever we make the instance of our viewcontroller we can access the variables from it and set them before we present the next view controller. Let’s continue with the example from the previous section. We will add a textfield to the center or our ViewControllerA that will pass the Data to a label in the center of ViewControllerB when we press our Next Button. Let’s make a textField variable programmatically in our ViewControllerA

* It’ll have a width of 180 with a height of 100. The position won’t matter since we will use autolayout
* Let’s make a placeHolder of “Enter Data to pass”
* Let’s give it a border style
* And finally, we need to make the translatesAutoresizingMaskIntoConstraints = false so that autolayout can translate the constraints.

let textField: UITextField = {

let field = UITextField(frame: CGRect(x: 0, y: 0, width: 180, height: 100))

field.placeholder = "Enter Data to pass"

field.borderStyle = .bezel

field.translatesAutoresizingMaskIntoConstraints = false

return field

}()

Now let’s make a function that sets up the constraints for this textfield

* Here we first add the text field as a subview to our main self.view

Next, we activate the constraints that center the text field to the view’s center

func setUpTextField(){

self.view.addSubview(textField)

NSLayoutConstraint.activate([

textField.centerXAnchor.constraint(equalTo: view.centerXAnchor),

textField.centerYAnchor.constraint(equalTo: view.centerYAnchor)

])

}

Don’t forget to call this new function in the viewDidload()

override func viewDidLoad() {

super.viewDidLoad()

self.view.backgroundColor = .white

self.title = "ViewController A"

//Set up next Button

setUpNavigationButton()

//Set up the textfield in the middle of the screen

setUpTextField()

}

If we run this now you will see a text field in the middle of the screen. Now let’s make a label in our ViewControllerB. We will do the same and center it in the middle of the screen. This label will populate based on the data we have entered in ViewControllerA.

class ViewControllerB: UIViewController {

let label: UILabel = {

let label = UILabel(frame: CGRect(x: 0, y: 0, width: 200, height:

100))

label.text = "Data passed"

label.font = .preferredFont(forTextStyle: .title1)

label.translatesAutoresizingMaskIntoConstraints = false

return label

}()

Now again let’s make a function that sets up the constraints for this label inside ViewControllerBDon’t forget to call it in the viewDidload()

override func viewDidLoad() {

super.viewDidLoad()

self.view.backgroundColor = .white

self.title = "ViewController B"

setUpLabel()

}

func setUpLabel(){

view.addSubview(label)

NSLayoutConstraint.activate([

label.centerXAnchor.constraint(equalTo: view.centerXAnchor),

label.centerYAnchor.constraint(equalTo: view.centerYAnchor)

])

}

If you run this you will see the label in the middle of ViewControllerB but the label isn’t changing ?!? We are missing the final step which is to pass the data from the text field in ViewControllerA -> ViewControllerB let’s make a variable in ViewControllerB named

class ViewControllerB: UIViewController {

var dataString: String?

We are making this variable optional so that it can be nil if no data is passed. Now in our viewDidload() let’s set the label.text = dataString

override func viewDidLoad() {

super.viewDidLoad()

self.view.backgroundColor = .white

self.title = "ViewController B"

setUpLabel()

label.text = dataString

}

Finally, let’s pass the data from our text field in ViewControllerA to the Label in ViewControllerB. We need this in our nextButtonTapped() function right before we push the ViewControllerB. We will make sure that our text field has data. If it does then we will pass it to the dataString variable in ViewControllerB and then push the controller.

And with one line of code, we passed the data from the text field to the dataString variable that is set in our ViewControllerB’s viewDidLoad() method. This will update the label and show the data.

@objc func nextButtonTapped(){

print("Show ViewController B ")

//let's make ViewControllerB

let viewB = ViewControllerB()

//Passing the data

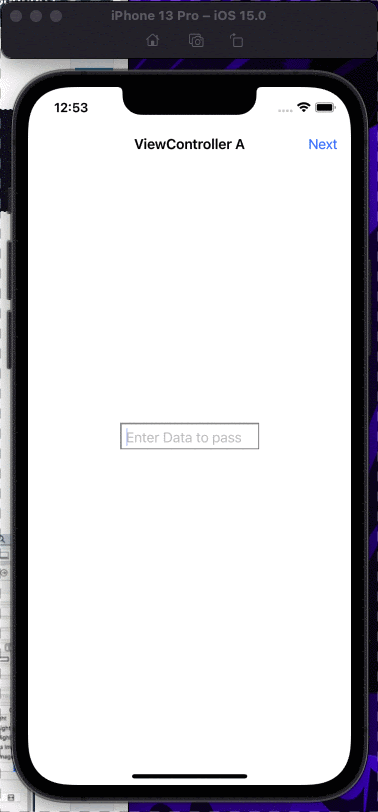
viewB.dataString = textField.text

//Now let's call our navigation controller to push and show this view when the button is clicked

self.navigationController?.pushViewController(viewB, animated:

true)

}



# Collection Views

Since you're a pro with Table Views already, checkout the documentation for Collection Views: <https://developer.apple.com/documentation/uikit/uicollectionview>.

* Learning how to read documentation is a skill in itself!

### Collection View Breakdown

If you've used iOS, there is a great chance you've used collection views. They’re used that frequently in iOS apps!

1. UICollectionView: This is the main view. Just like table views, a collection view is a UIScrollView subclass.
2. UICollectionViewCell: This is very similar to UITableViewCell in a table view. These cells are **subviews** to the collection view. You can create cells programmatically or inside Interface Builder.

Just like table views, most of the time, we will want to create our own custom UICollectionViewCell subclass.

* The collection view gets its data from the data source object, stored in the collection view’s [dataSource](https://developer.apple.com/documentation/uikit/uicollectionview/1618091-datasource" \t "_blank) property.

### Register the Collection View Cell

collectionView.register(UICollectionViewCell.self, forCellWithReuseIdentifier: "Cell")

### Next, we dequeue a cell like this:

func collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {

let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "Cell", for: indexPath)

return cell

}

Be sure to read through Apple documentation before moving forward! Like Table Views, Collection Views are powerful for your applications.

Your next assignment will take advantage of Collection View Cells!

صورة تحتوي على نص, مختلف, لقطة شاشة, العديد

تم إنشاء الوصف تلقائياً