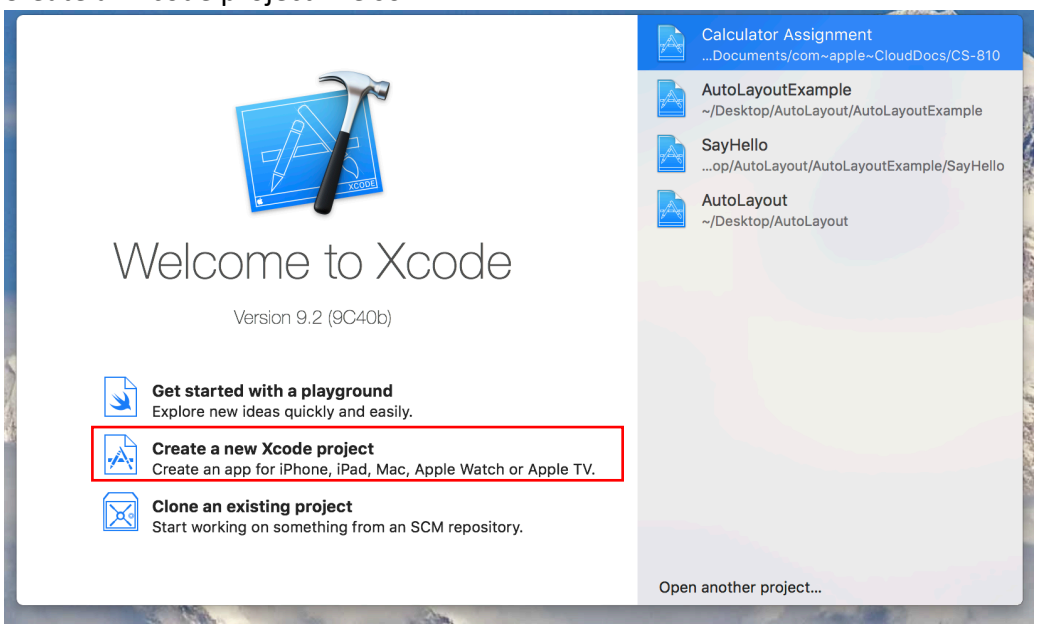
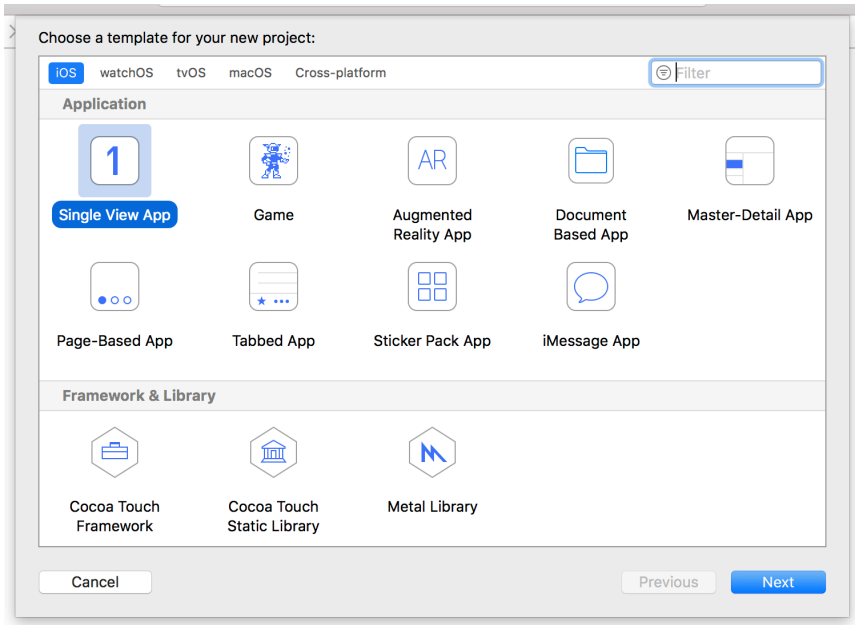
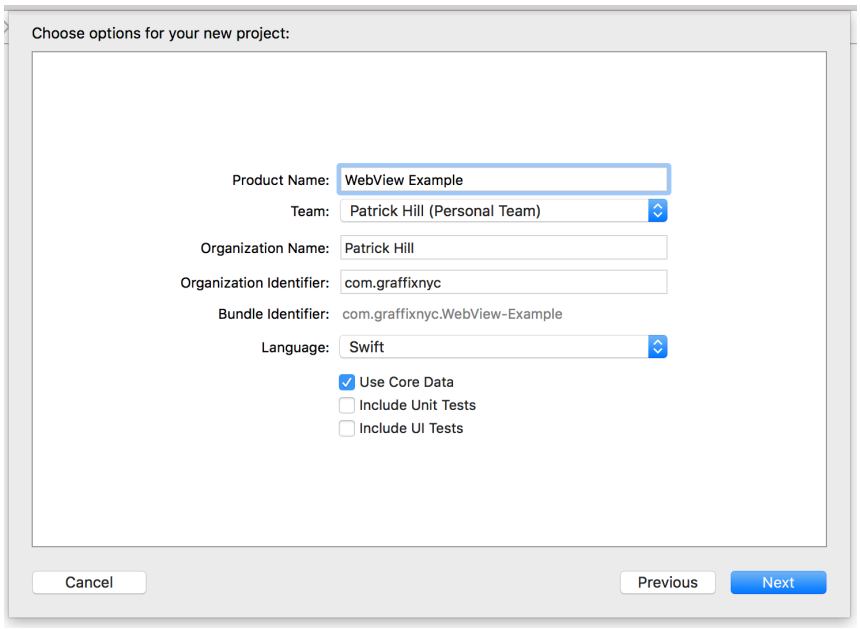
CS 523

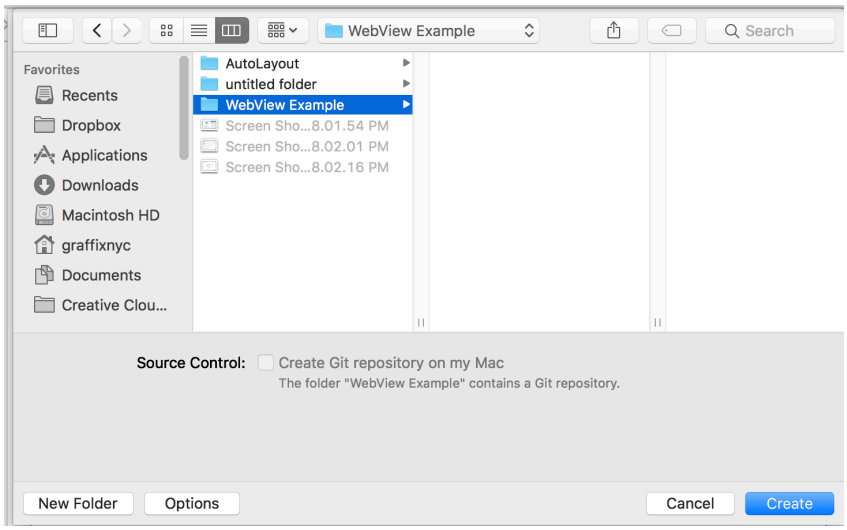
In this exercise we are going to create a basic web browser app that contains a WebView, two buttons (back and forward) and a text field to enter a url. When the application first loads the web view will load www.apple.com. Since there will be no browsing history when the application first loads we will disable the back and forward buttons.

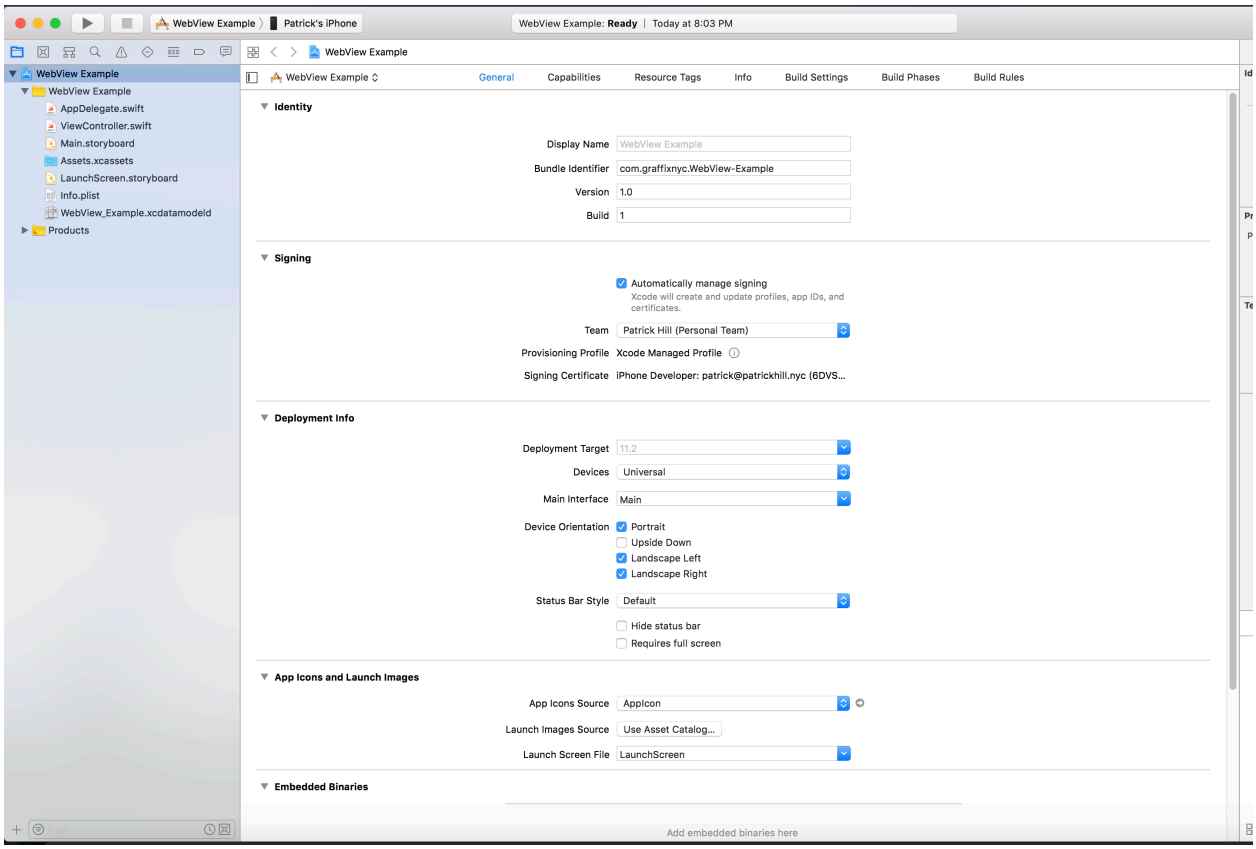
1. Create an Xcode project like so:



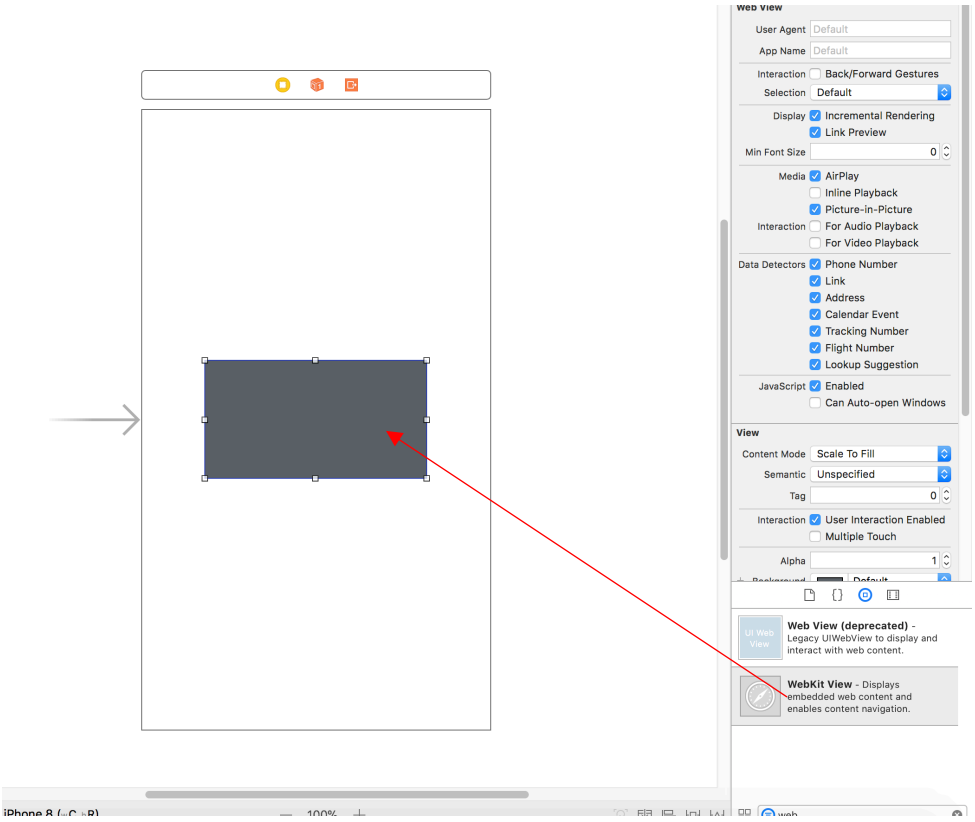



Choose options for your new project:
Product Name: WebView Example
Team: Patrick Hill (Personal Team)
Organization Name: Patrick Hill
Organization Identifier: com.graffixnyc
Bundle Identifier: com.graffixnyc.WebView-Example
Language: Swift
Use Core Data
Include Unit Tests
Include UI Tests
Cancel
Previous
Next

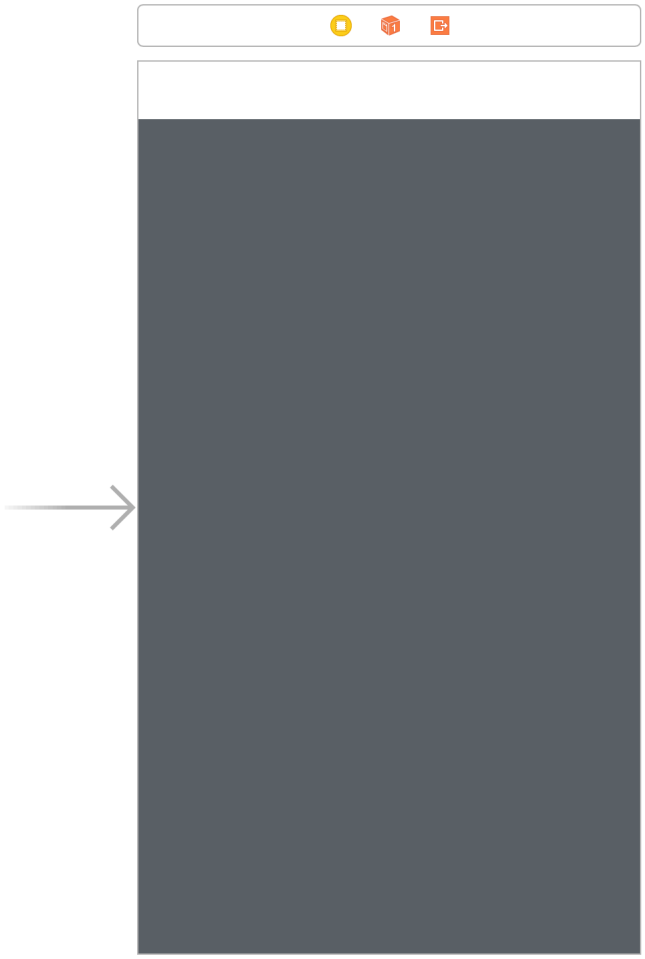

WebView Example
Search
Favorites
Recents
Dropbox
Applications
Downloads
Macintosh HD
graffixnyc
Documents
Creative Clou...
AutoLayout
untitled folder
WebView Example
Screen Sho...8.01.54 PM
Screen Sho...8.02.01 PM
Screen Sho...8.02.16 PM
Source Control: Create Git repository on my Mac
The folder "WebView Example" contains a Git repository.
New Folder
Options
Cancel
Create



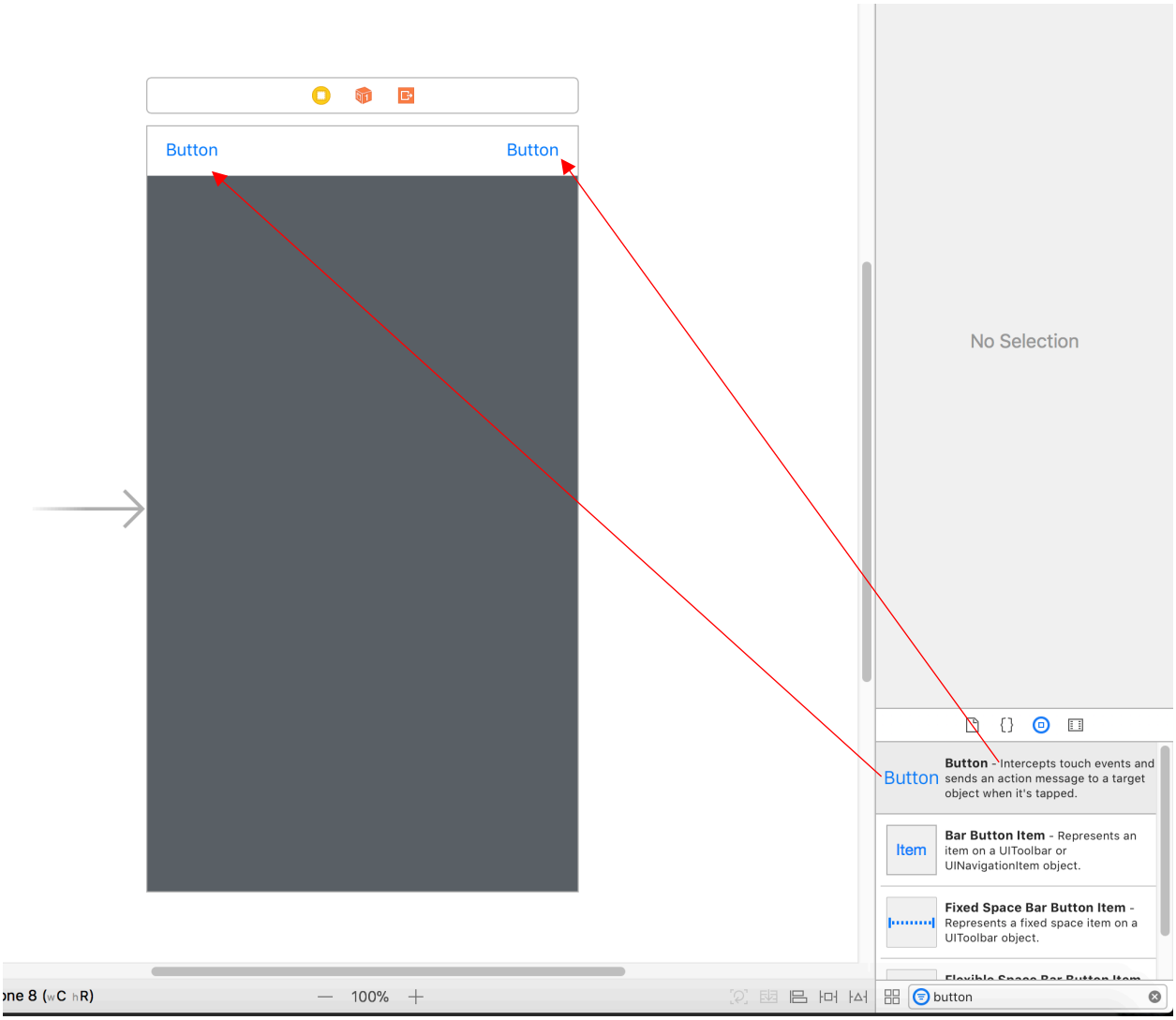
2. Once we get to the main screen of the application double click the Main.storyboard. We are going to be adding a WebKit View, two buttons and a textfield.
   a. Add the WebKit View by selecting it in the object browser and then dragging it onto the storyboard. (Make sure you select WebKit View and not WebView (depreciated)



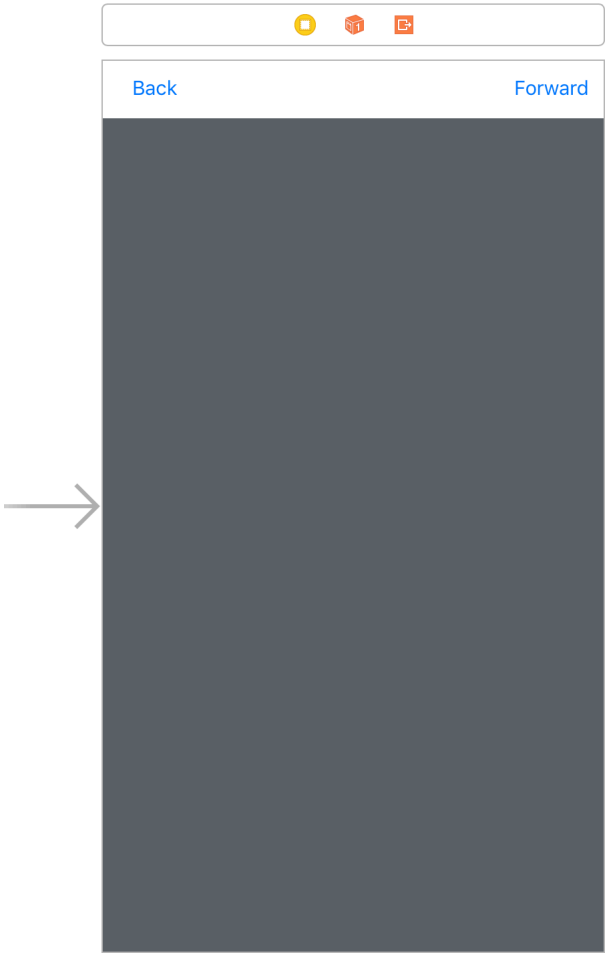
b. Resize it so it fills the sides and bottom leaving a little room at the top for the two buttons and text field



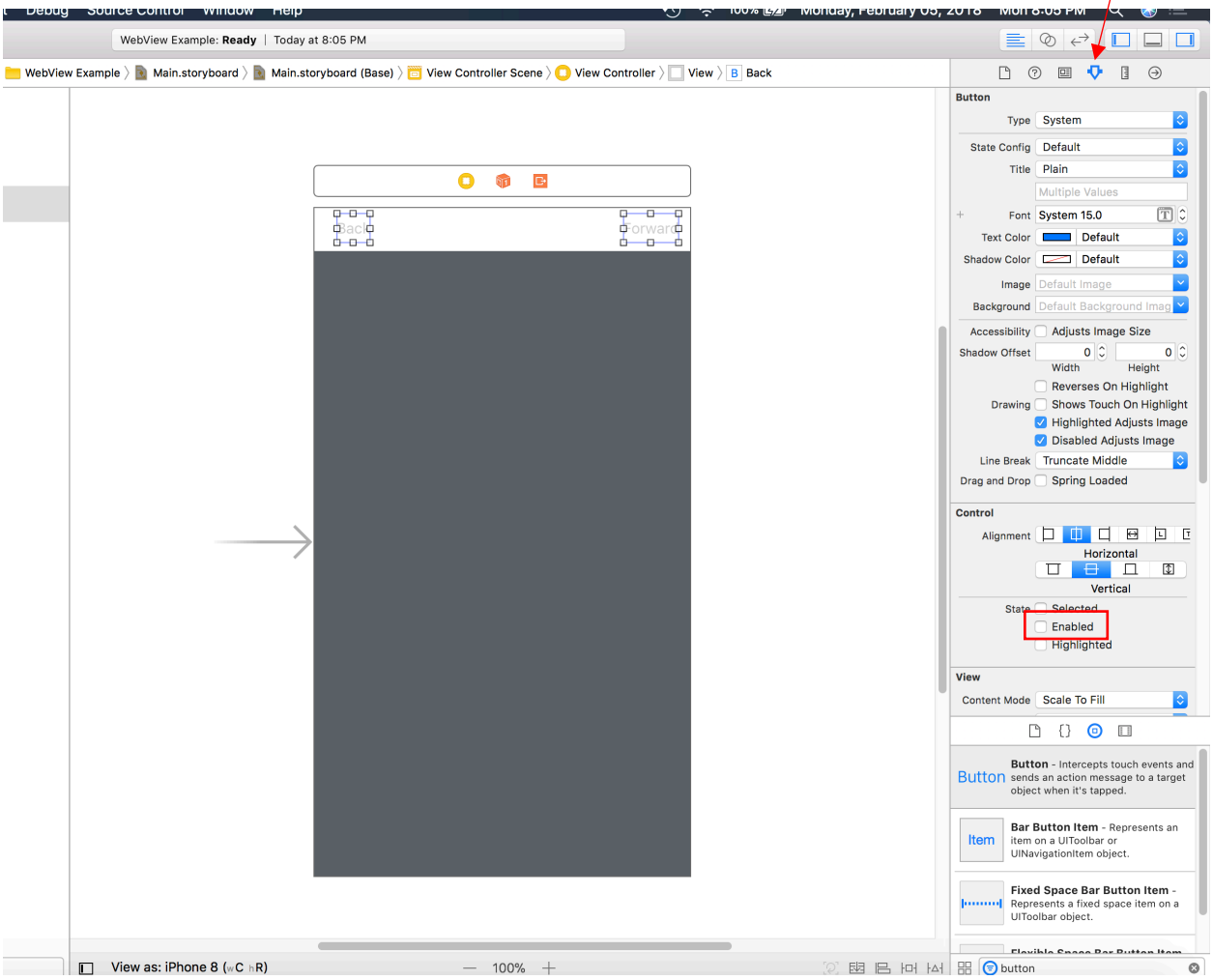
c. Now add two buttons



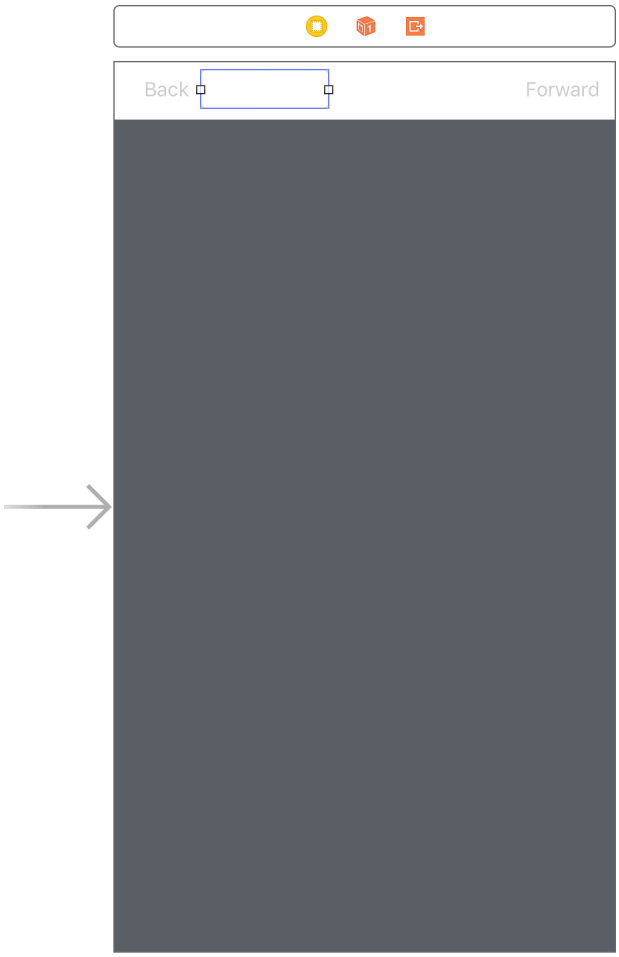
d. Change the values of the text on the buttons to read “Back” and “Forward” like so:



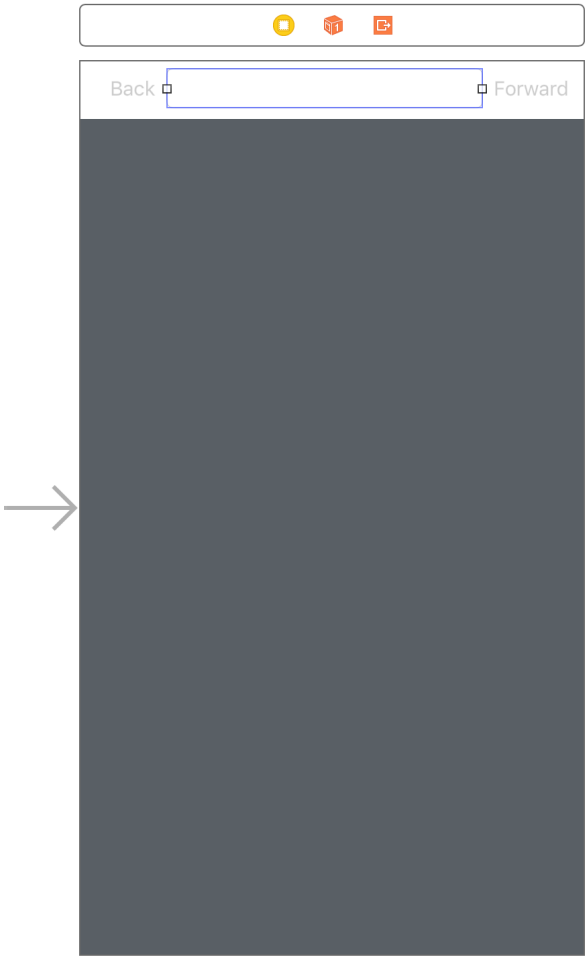
e. Now select both buttons by clicking one and then holding down command while you click the other. In the properties pane, uncheck enabled so they will be disabled (notice the text on the buttons goes from blue to gray to show they are disabled)



f. Now we are going to drag a Text Field to the view and position it between the two buttons.

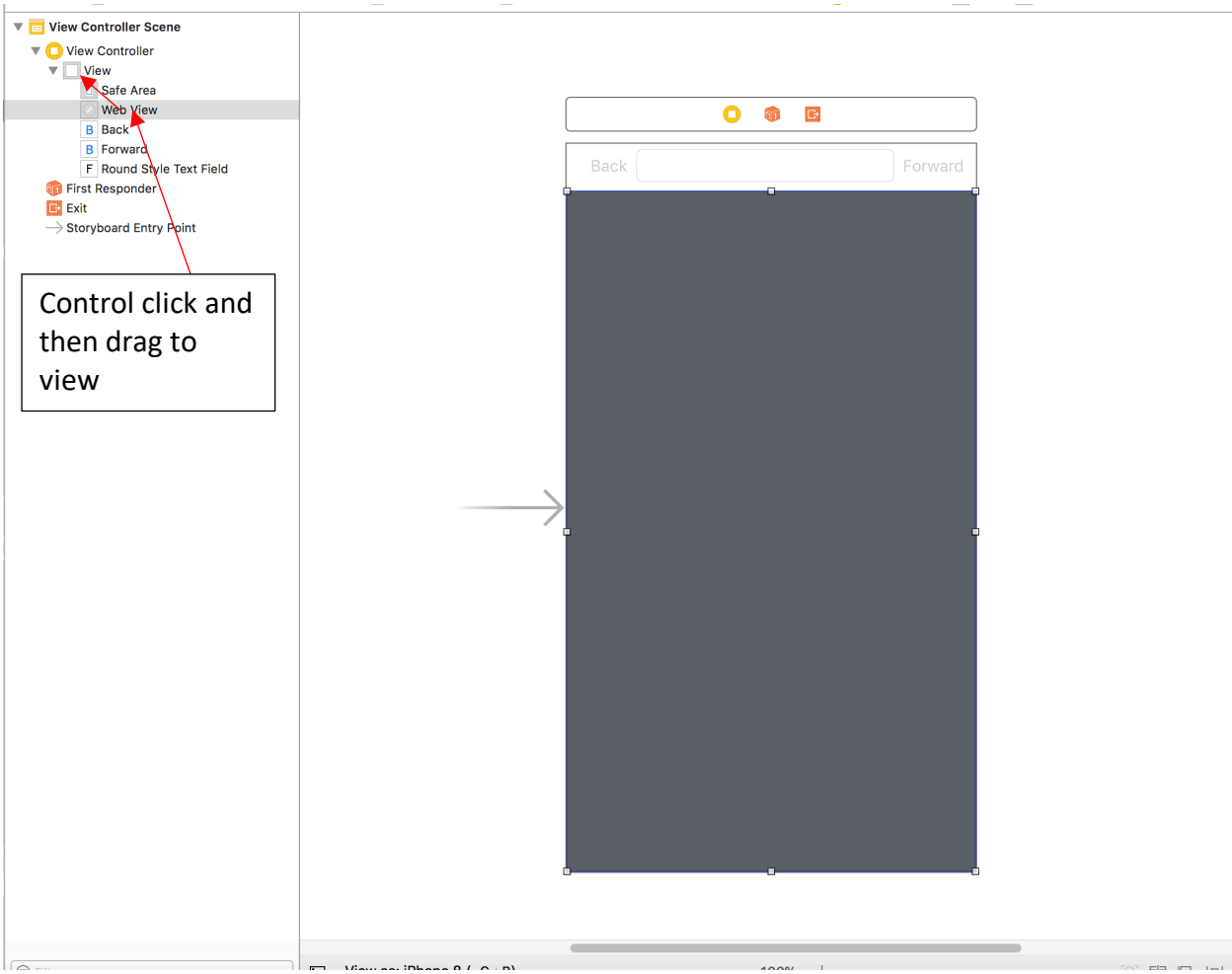


g. And then stretch it across to the other side so it fills the space between the buttons like so:

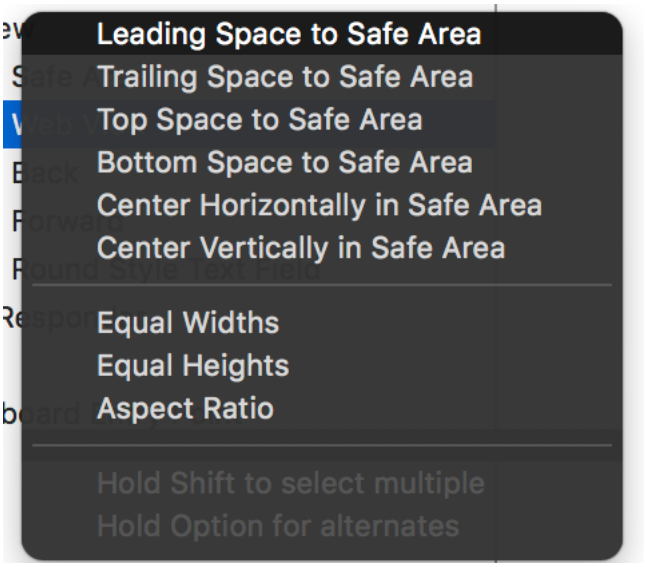


3. Now we are going to add some constraints to the controls so the UI looks good on different sized devices.
    a. First select the WebKit View

    b. Now we are going to add some constraints a little different than we saw last time. In the left pane hold down the Control button, click the Web View in the hierarchy and drag it to the top view.

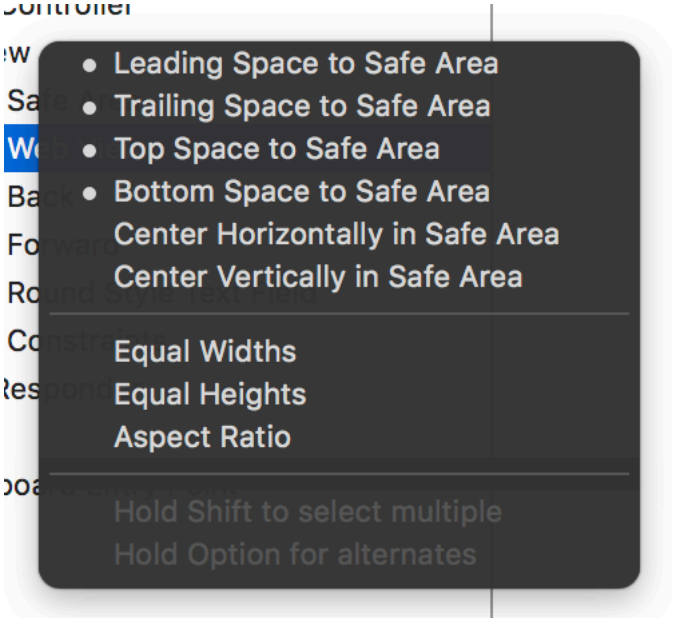


c. When you control click and drag the webview to the view controller a context menu is shown.

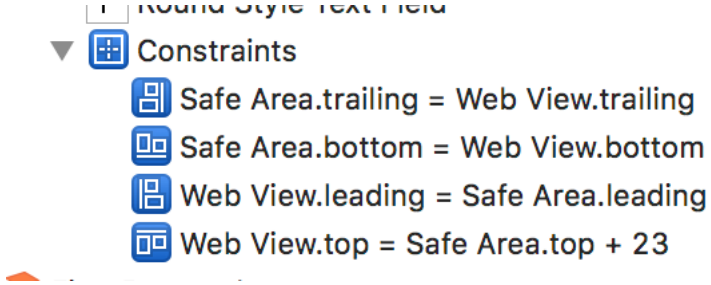


d. Select “Leading Space to Safe Area” repeat the steps for “Trailing Space to Safe Area”, “Top Space to Safe Area” and “Bottom Space to Safe Area”

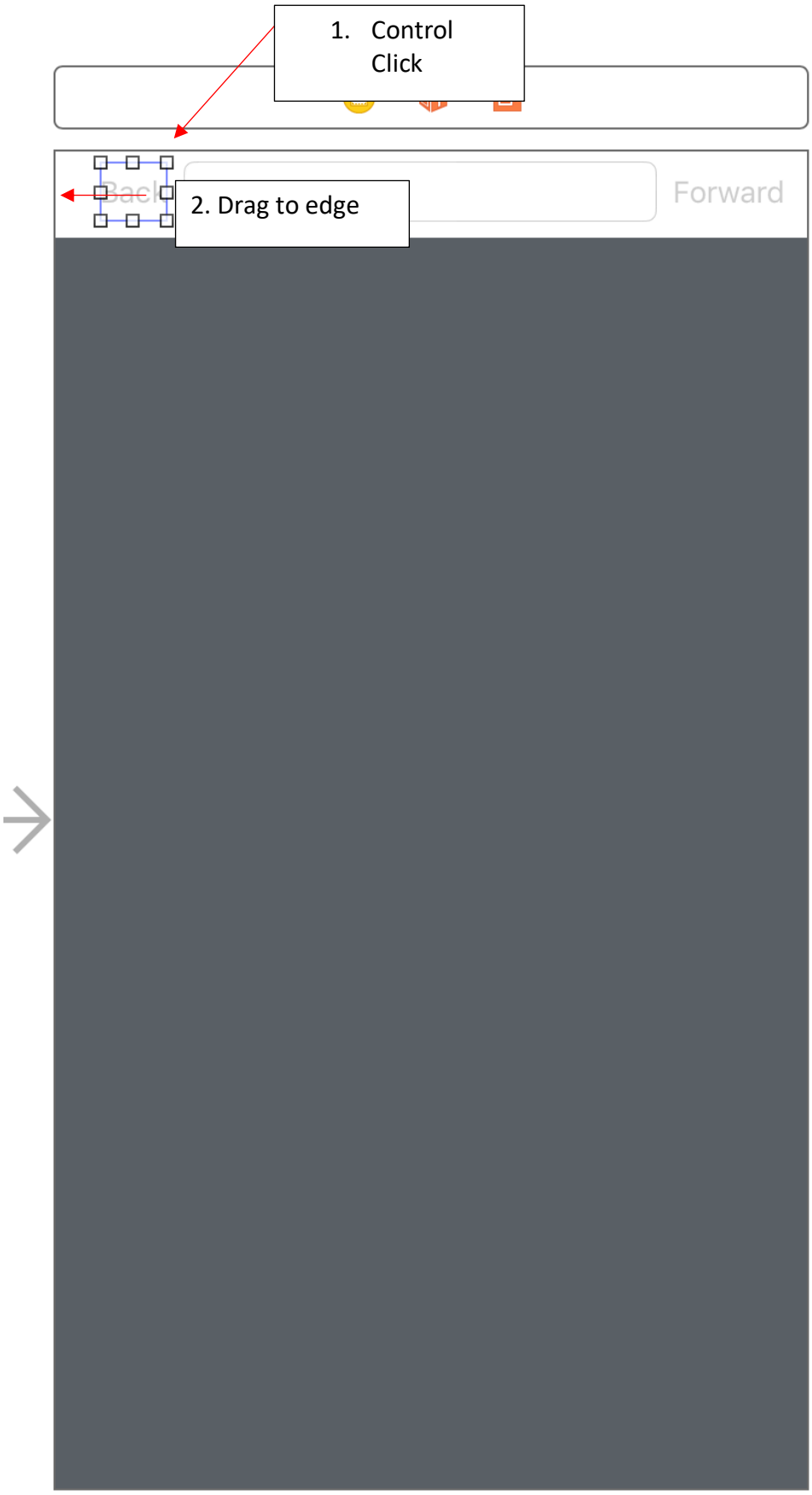
This is what the context menu will look like when you selected all 4 they will have the indicators next to them:



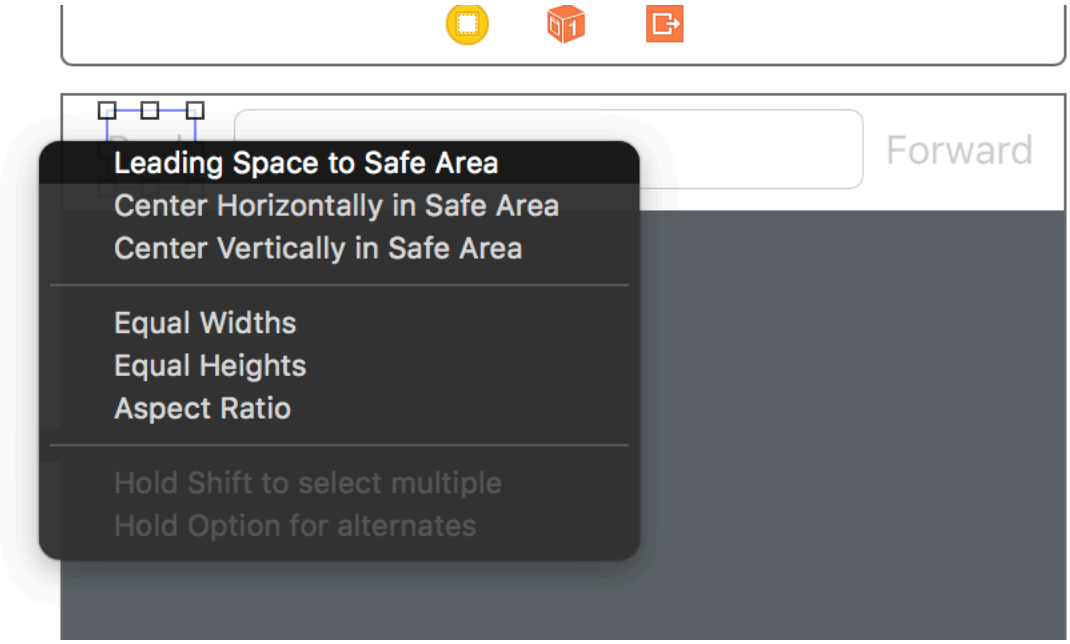
Also notice it created some constraints:



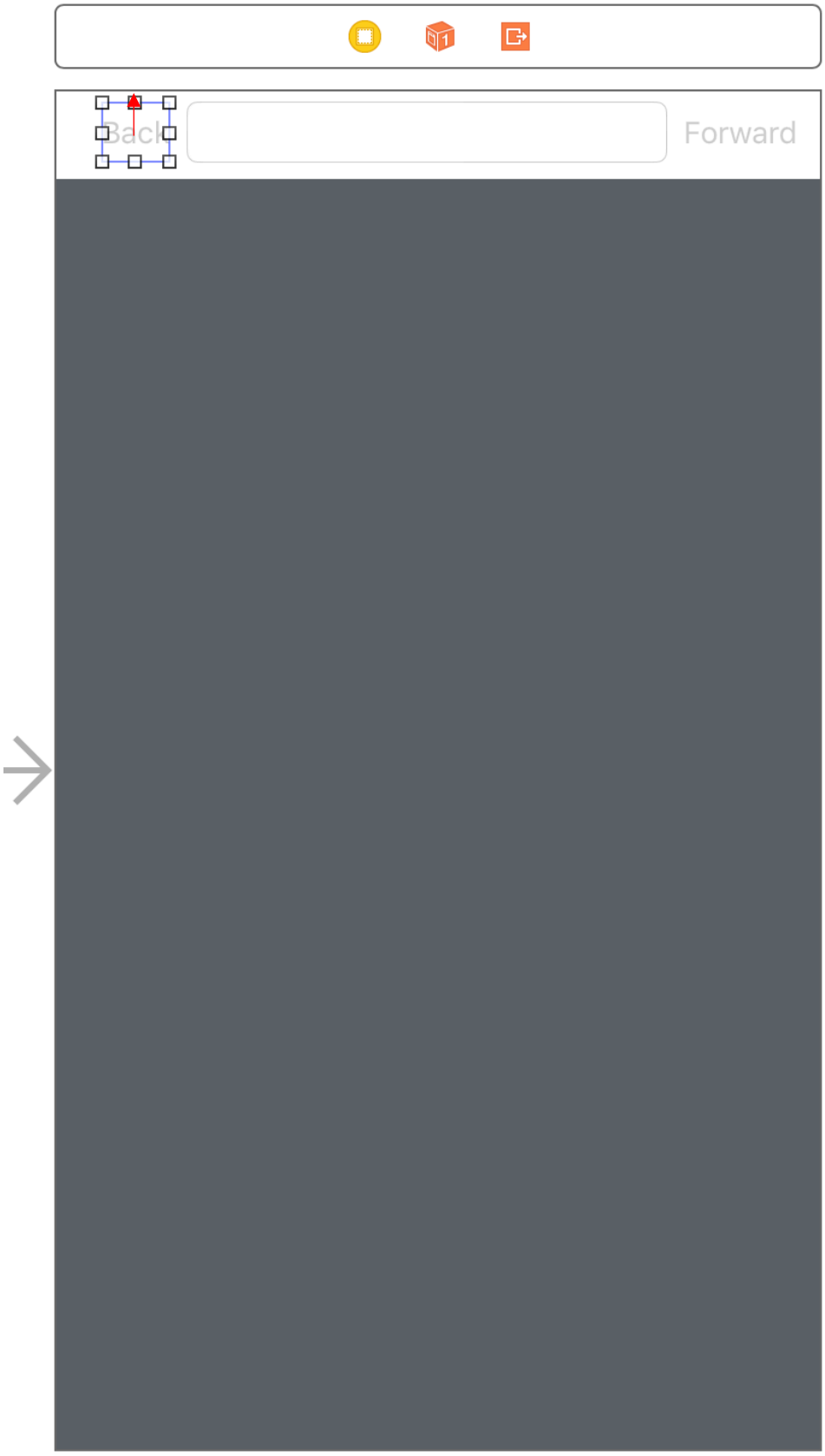
e. Now will we add some constraints to the buttons.
Select the back button on the view controller while it’s selected, hold down the control key and then drag to the left hand side of the screen


1. Control Click
Forward
2. Drag to edge

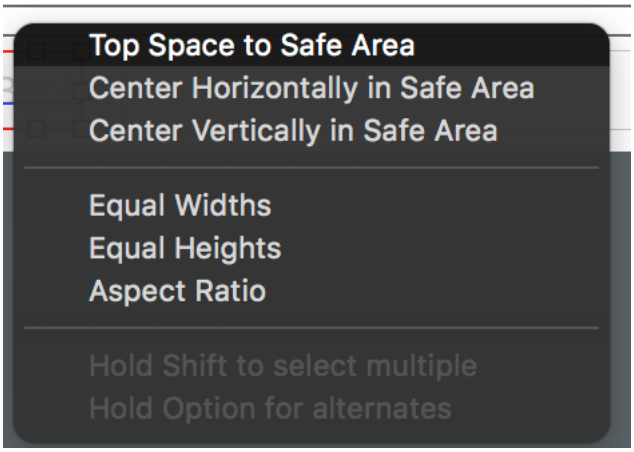
f. When the context menu appears select “Leading Space to Safe Area”



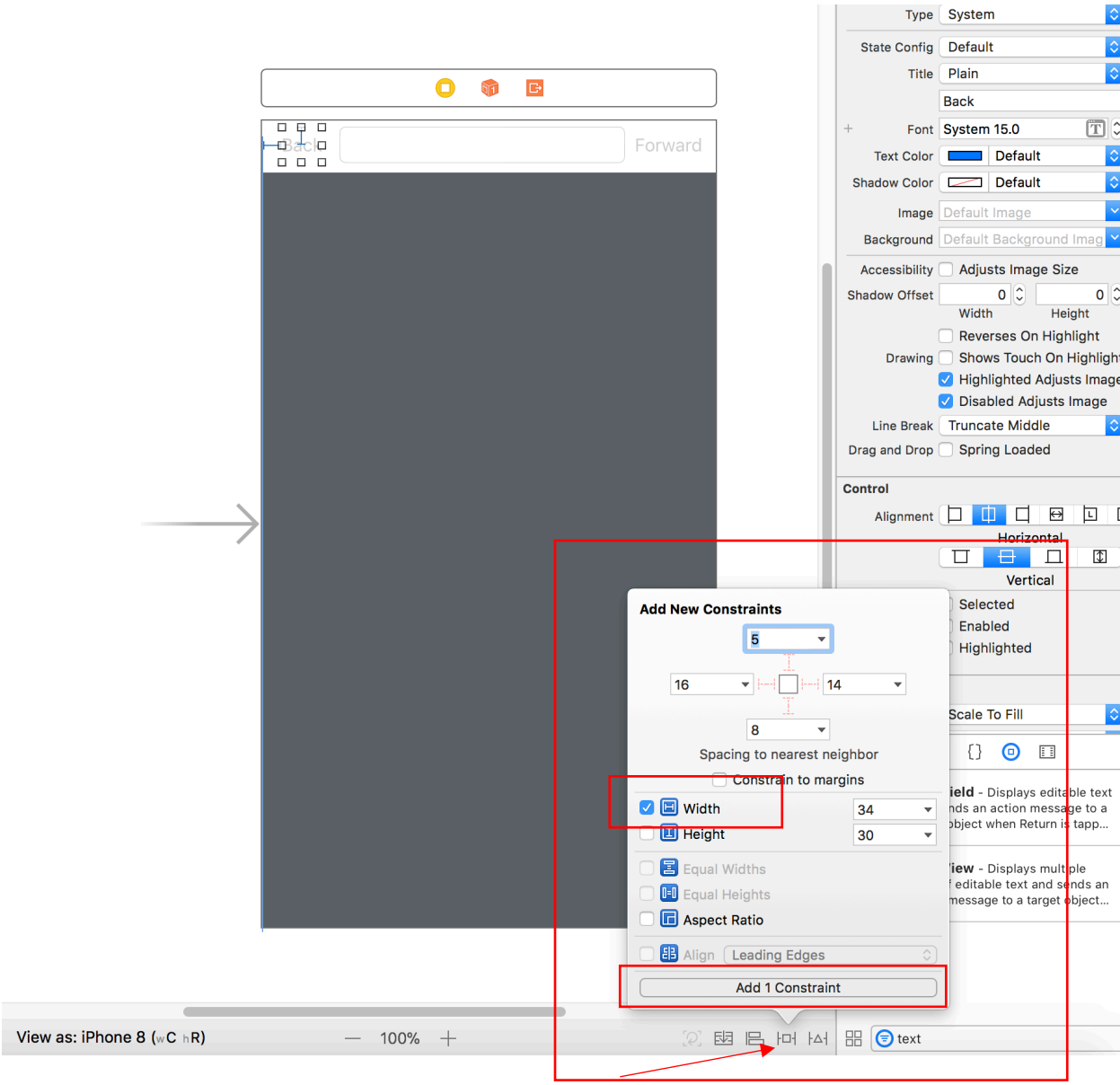
g. Now select the button again, this time control click and drag to the top of the screen


Back
Forward

h. Now select “Top Space to Safe Area”

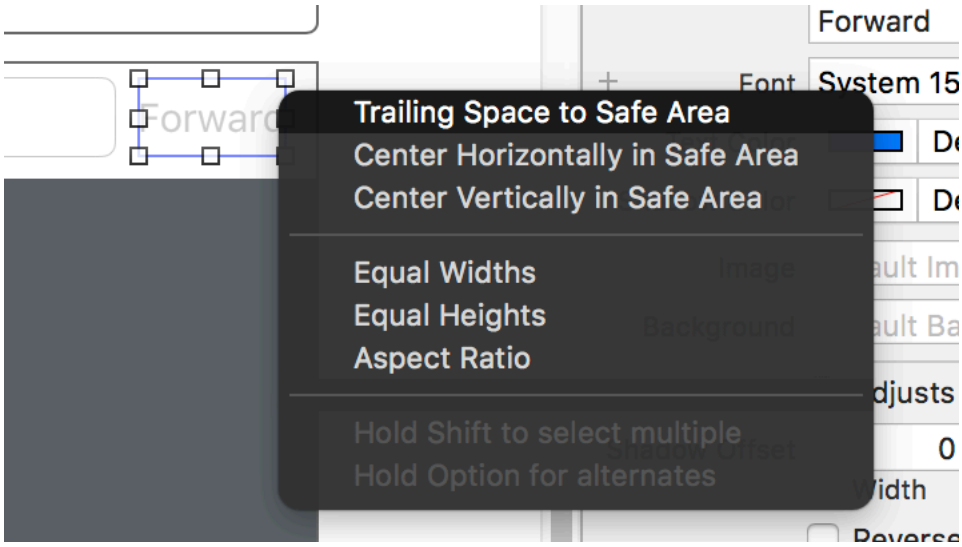


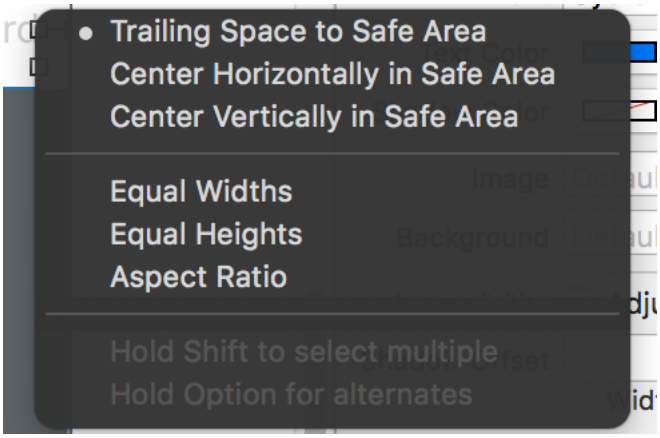
i. So now we can see we have some constraints for the button, We need to add one more and that is for the width. Select the button and then click the pinning option in the bottom right:

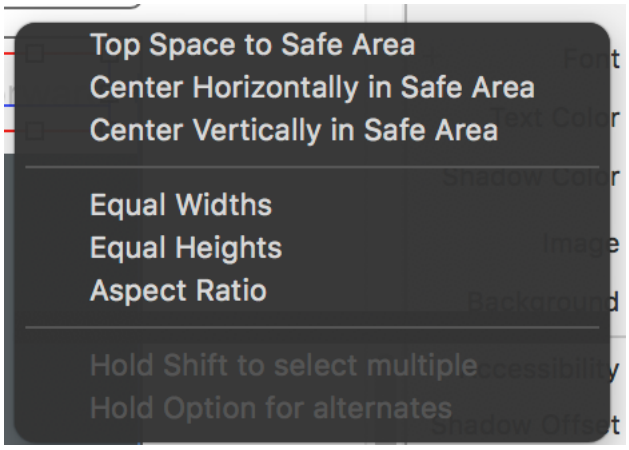


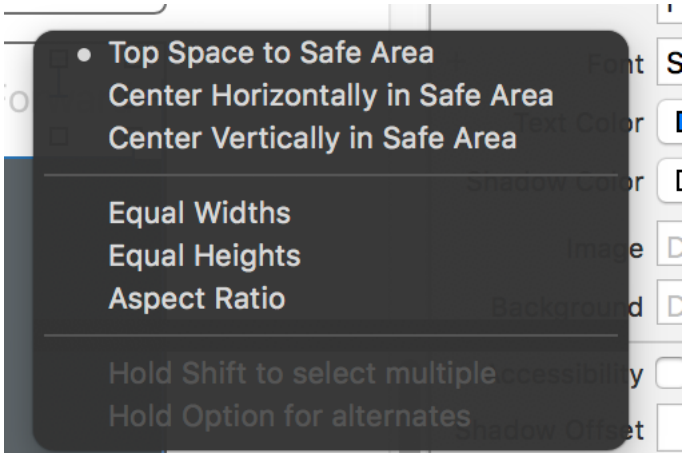
j. Check width and then Add 1 constraint

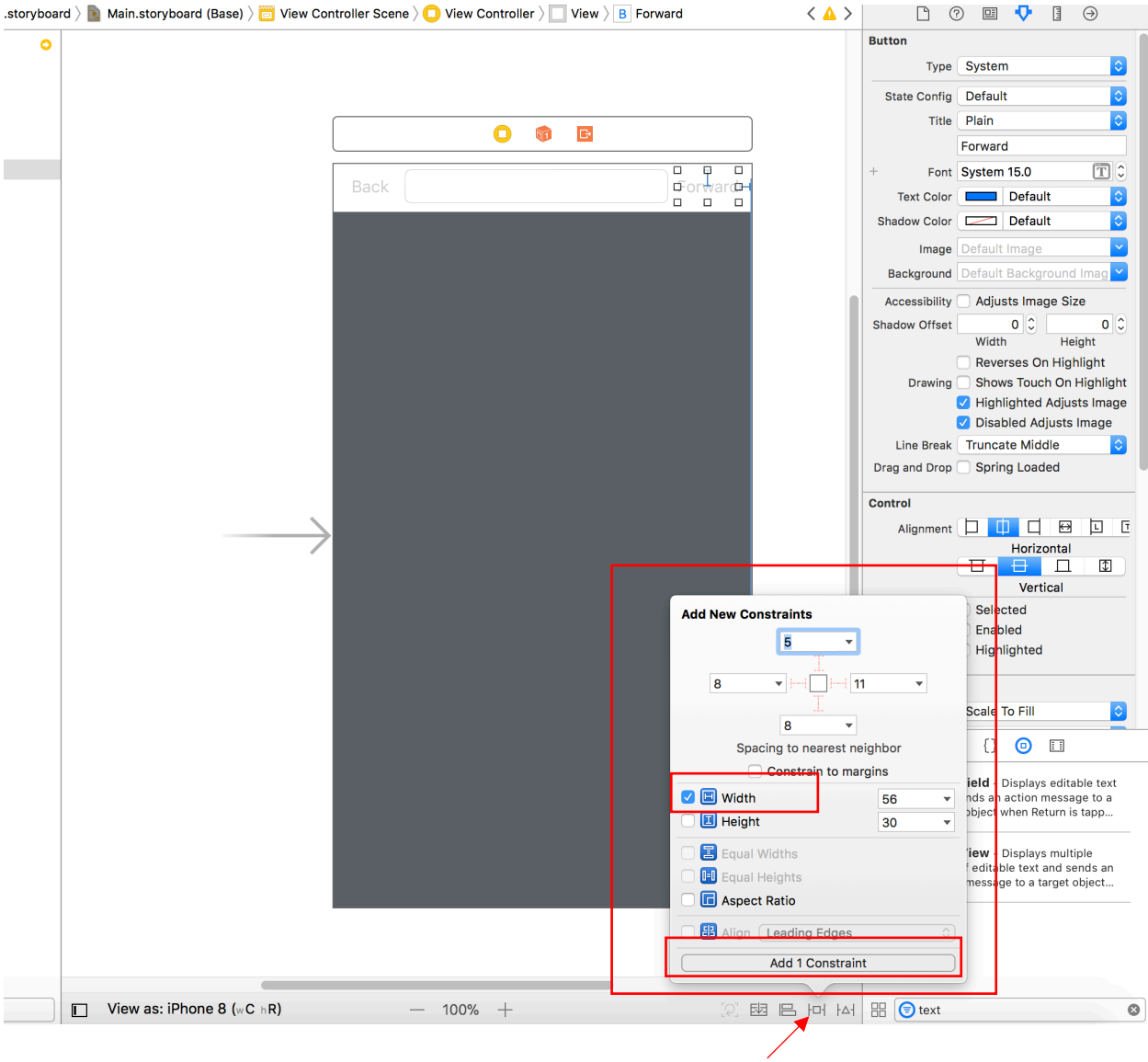
k. We are now done with the back button. Now we will do the same for the Forward button except instead of the leading space we will use the trailing space to the right, and do the same for the top and the width that we did for the back button


Forward
Font
System 15
Trailing Space to Safe Area
Center Horizontally in Safe Area
Center Vertically in Safe Area
Equal Widths
Equal Heights
Aspect Ratio
Hold Shift to select multiple
Hold Option for alternates

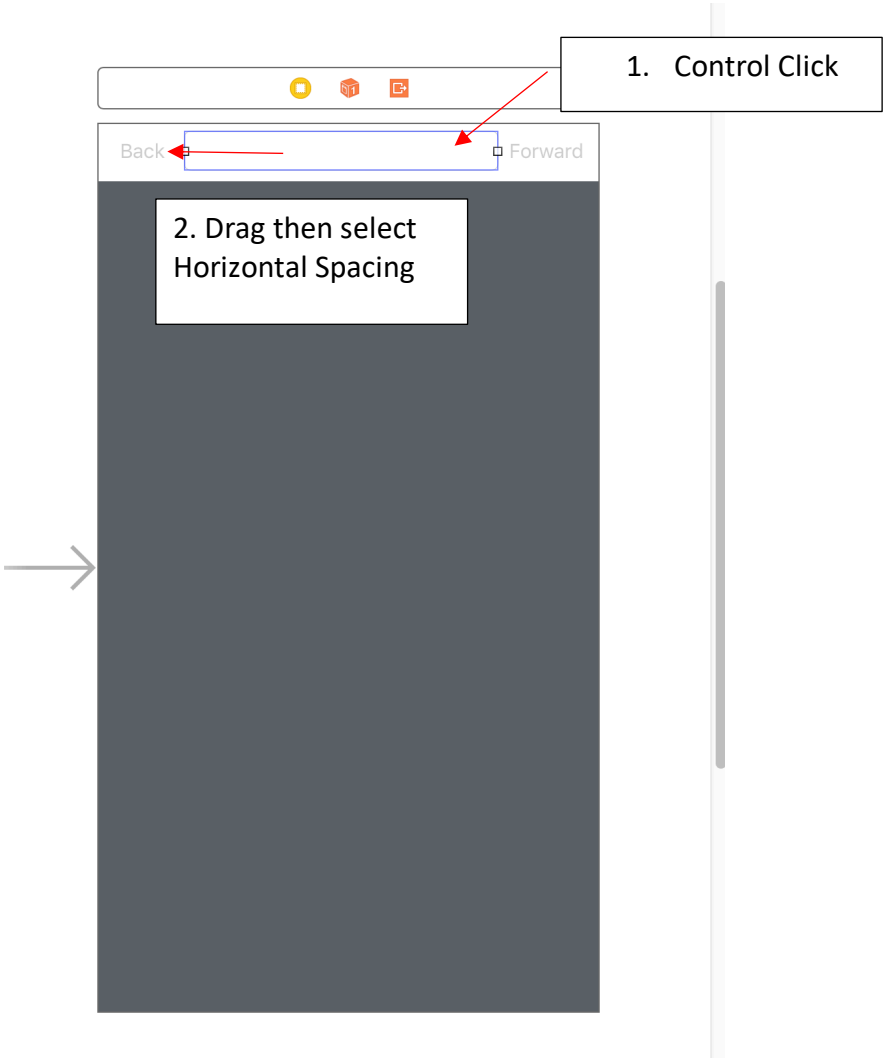

Trailing Space to Safe Area
Center Horizontally in Safe Area
Center Vertically in Safe Area
Equal Widths
Equal Heights
Aspect Ratio
Hold Shift to select multiple
Hold Option for alternates

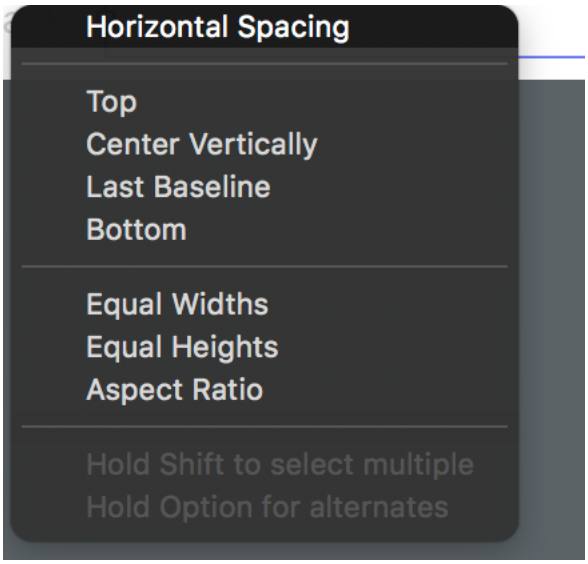

Top Space to Safe Area
Center Horizontally in Safe Area
Center Vertically in Safe Area
Equal Widths
Equal Heights
Aspect Ratio
Hold Shift to select multiple
Hold Option for alternates


Top Space to Safe Area
Center Horizontally in Safe Area
Center Vertically in Safe Area
Equal Widths
Equal Heights
Aspect Ratio
Hold Shift to select multiple
Hold Option for alternates

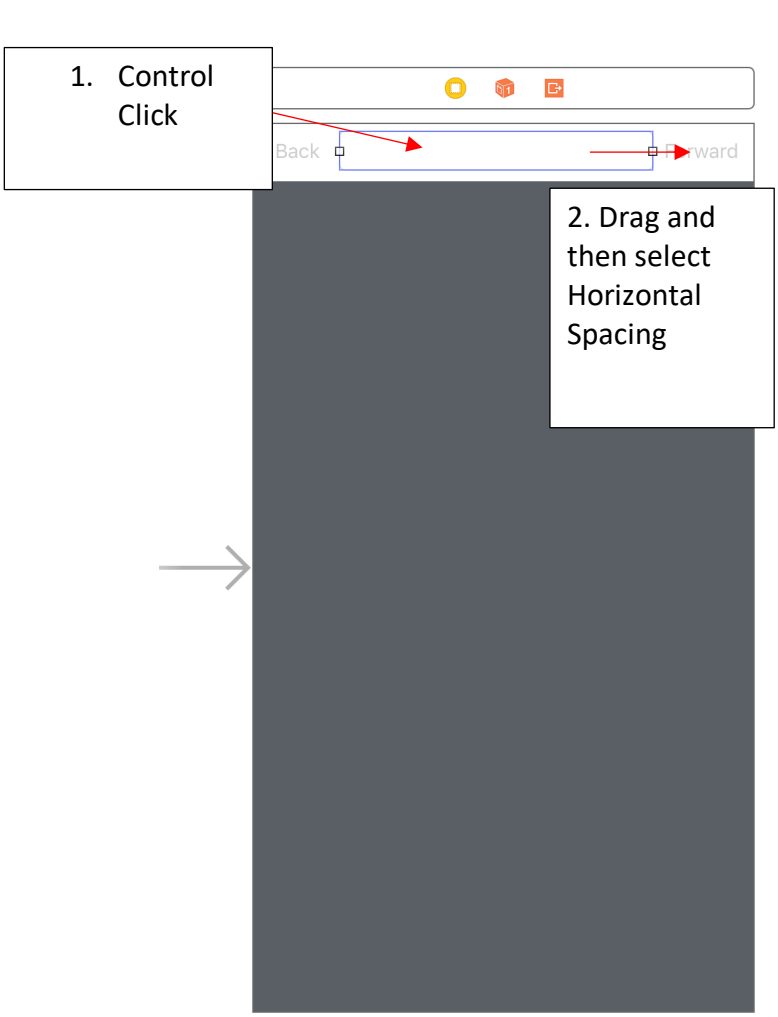

.storyboard
Main.storyboard (Base)
View Controller Scene
View Controller
View
Forward
Back
Forward
Button
Type
System
State Config
Default
Title
Plain
Forward
Font
System 15.0
Text Color
Default
Shadow Color
Default
Image
Default Image
Background
Default Background Imag
Accessibility
Adjusts Image Size
Shadow Offset
Width
Height
Reverses On Highlight
Drawing
Shows Touch On Highlight
Highlighted Adjusts Image
Disabled Adjusts Image
Line Break
Truncate Middle
Drag and Drop
Spring Loaded
Control
Alignment
Horizontal
Vertical
Selected
Enabled
Highlighted
Scale To Fill
Add New Constraints
Spacing to nearest neighbor
Constrain to margins
Width
Height
Equal Widths
Equal Heights
Aspect Ratio
Align
Leading Edges
Add 1 Constraint
View as: iPhone 8
100%
text

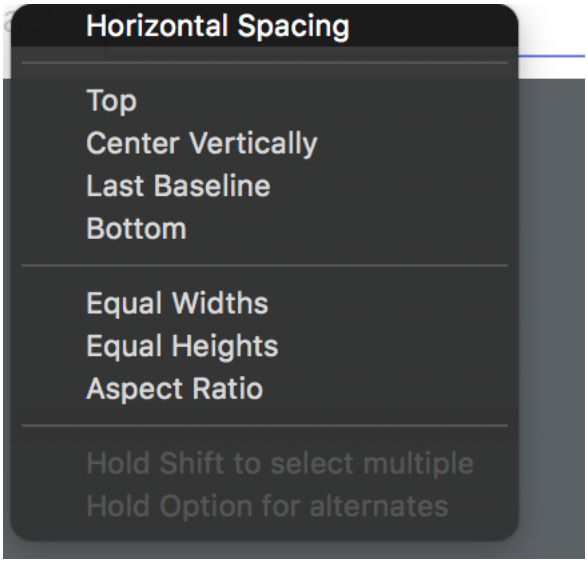
l. Now we just have to add some constraints to the text field. Control click the text field and then drag to the left to the back button (like we did for the buttons) and select Horizontal Spacing



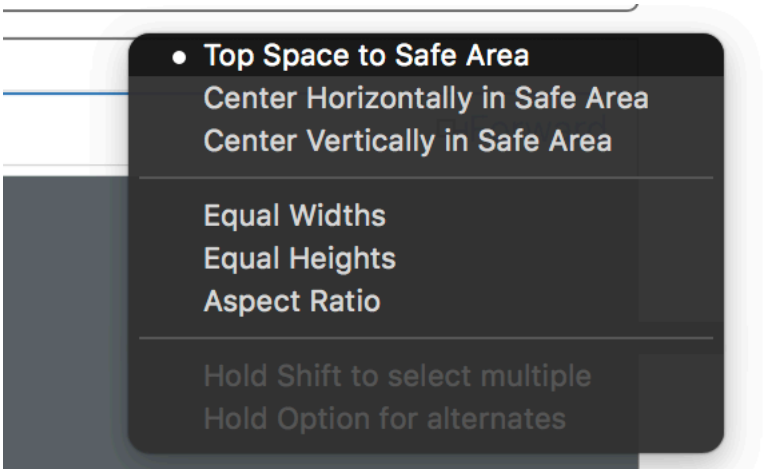


Now do the same thing but control drag to the forward button



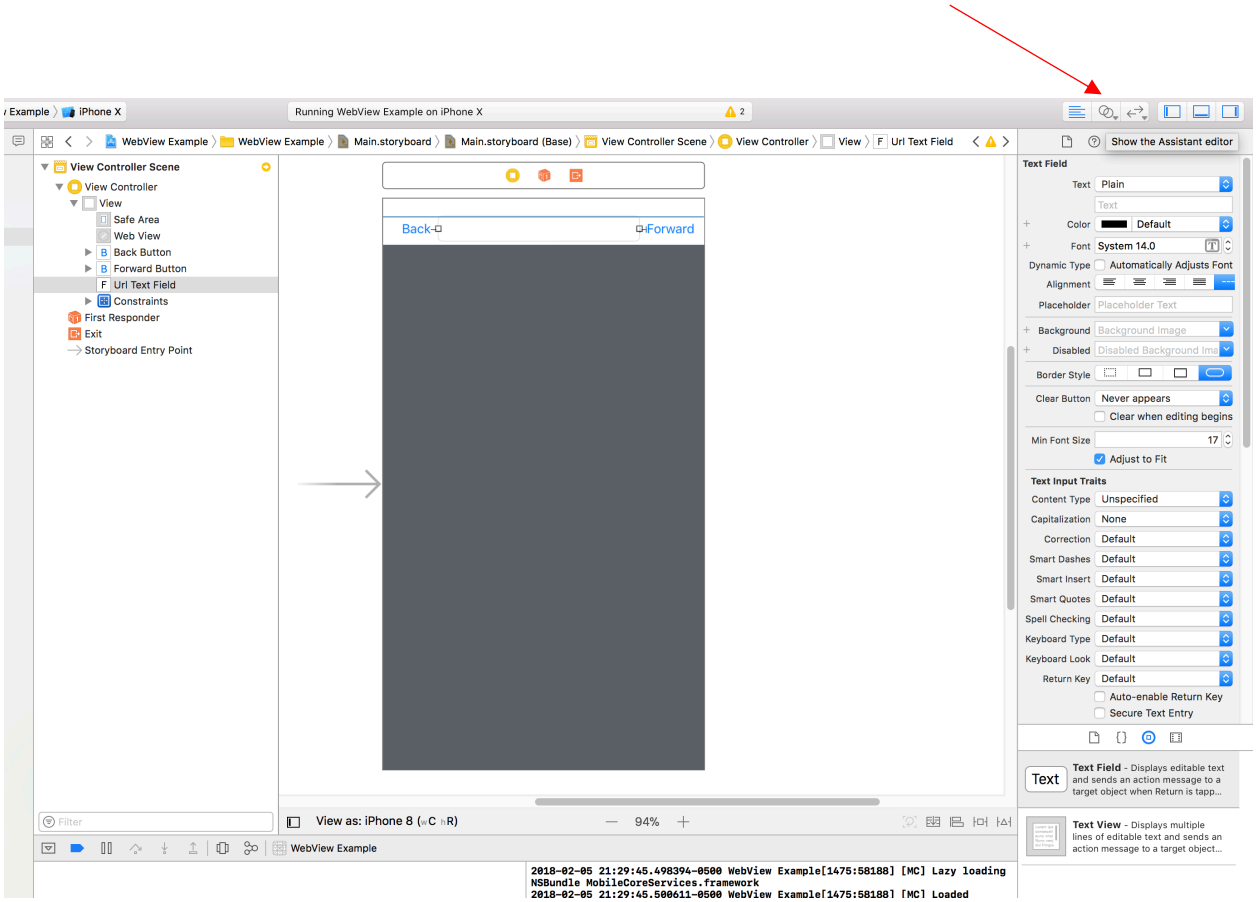


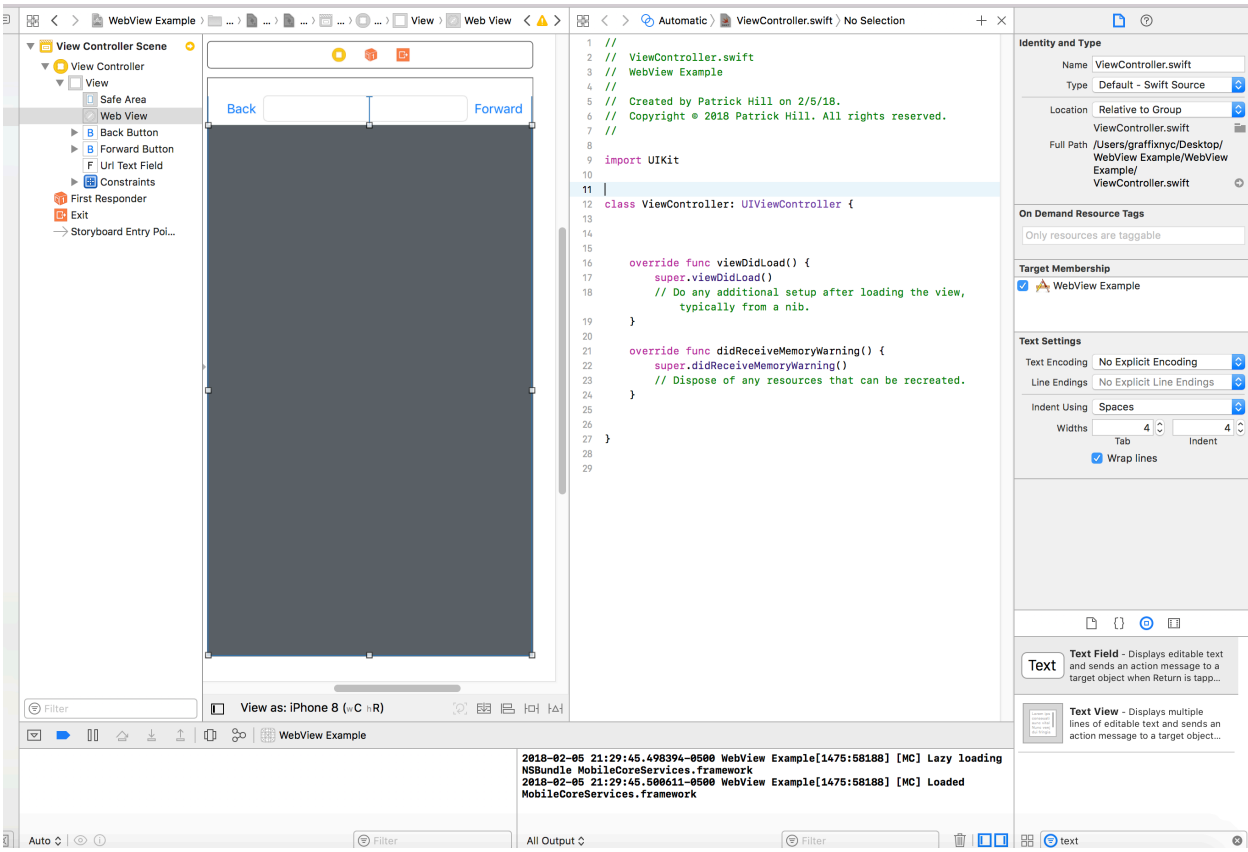
We just have to add some top spacing to the text field. Select the text field and then control click and drag to the top of the screen and set the top leading space



Now that is it for the UI now, lets add the references to the controls in the code.

4. Select Assistant Editor to open up the split screen view, we will not add references to the UI controls in the code.


Example
iPhone X
Running WebView Example on iPhone X
WebView Example
Main.storyboard
Main.storyboard (Base)
View Controller Scene
View Controller
View
Url Text Field
Show the Assistant editor
Safe Area
Web View
Back Button
Forward Button
Constraints
First Responder
Exit
Storyboard Entry Point
Back
Forward
Text Field
Text
Plain
Color
Default
Font
System 14.0
Dynamic Type
Automatically Adjusts Font
Alignment
Placeholder
Placeholder Text
Background
Background Image
Disabled
Border Style
Clear Button
Never appears
Clear when editing begins
Min Font Size
17
Adjust to Fit
Text Input Traits
Content Type
Unspecified
Capitalization
None
Correction
Smart Dashes
Smart Insert
Smart Quotes
Spell Checking
Keyboard Type
Keyboard Look
Return Key
Auto-enable Return Key
Secure Text Entry
Text Field - Displays editable text and sends an action message to a target object when Return is tapp...
Text View - Displays multiple lines of editable text and sends an action message to a target object...
View as: iPhone 8 (wC hR)
94%
WebView Example
2018-02-05 21:29:45.498394-0500 WebView Example[1475:58188] [MC] Lazy loading NSBundle MobileCoreServices.framework
2018-02-05 21:29:45.500611-0500 WebView Example[1475:58188] [MC] Loaded


WebView Example
View
Web View
Automatic
ViewController.swift
No Selection
View Controller Scene
Safe Area
Web View
Back Button
Forward Button
Url Text Field
Constraints
First Responder
Exit
Storyboard Entry Point
Back
Forward
//
// ViewController.swift
// WebView Example
//
// Created by Patrick Hill on 2/5/18.
// Copyright © 2018 Patrick Hill. All rights reserved.
//
import UIKit
class ViewController: UIViewController {
override func viewDidLoad() {
super.viewDidLoad()
// Do any additional setup after loading the view,
typically from a nib.
}
override func didReceiveMemoryWarning() {
super.didReceiveMemoryWarning()
// Dispose of any resources that can be recreated.
}
}
Identity and Type
Name
ViewController.swift
Type
Default - Swift Source
Location
Relative to Group
ViewController.swift
Full Path
/Users/graffixnyc/Desktop/WebView Example/WebView Example/ViewController.swift
On Demand Resource Tags
Only resources are taggable
Target Membership
WebView Example
Text Settings
Text Encoding
No Explicit Encoding
Line Endings
No Explicit Line Endings
Indent Using
Spaces
Widths
Tab
Indent
Wrap lines
Text
Text Field - Displays editable text and sends an action message to a target object when Return is tapp...
Text View - Displays multiple lines of editable text and sends an action message to a target object...
View as: iPhone 8 (wC hR)
WebView Example
2018-02-05 21:29:45.498394-0500 WebView Example[1475:58188] [MC] Lazy loading NSBundle MobileCoreServices.framework
2018-02-05 21:29:45.500611-0500 WebView Example[1475:58188] [MC] Loaded MobileCoreServices.framework
Auto
All Output
text

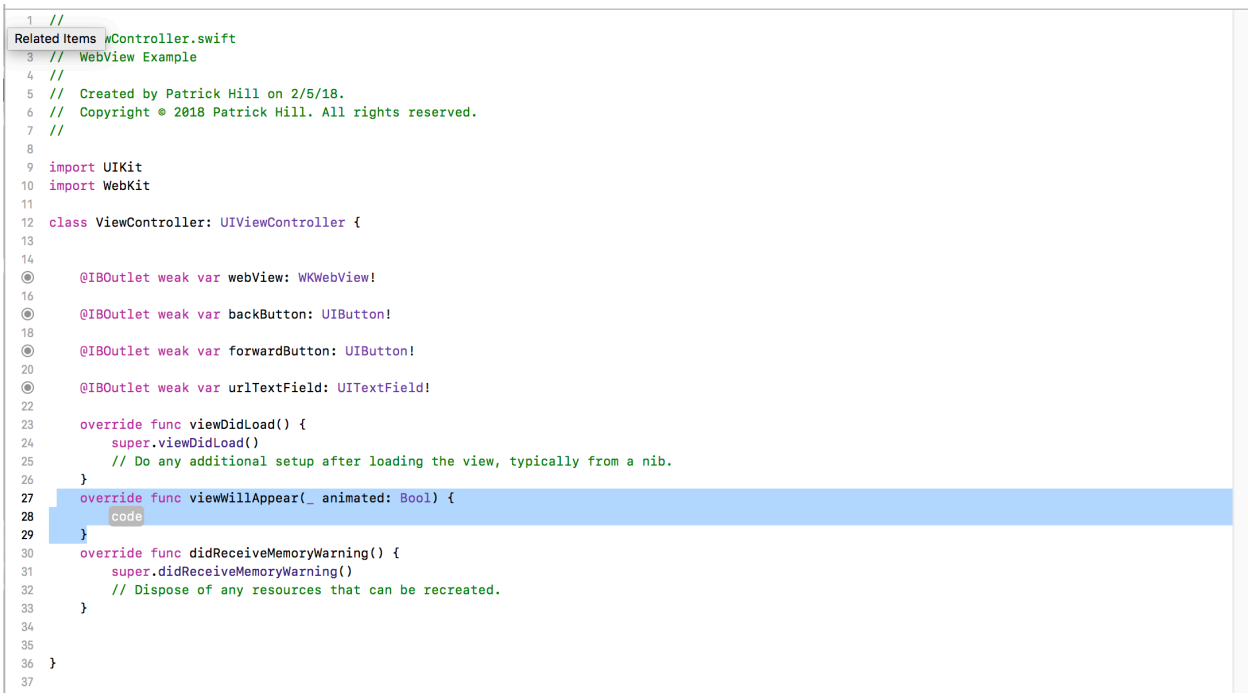
Now select the WebView and control drag it to the code window to create the reference. Name it webView, next do the same for the back button and name it backButton, do the same for the forward button and name it forwardButton, and then finally do the same for the text field and name it urlTextField. Now your code should look like this:

```
//

import UIKit


class ViewController: UIViewController {


    @IBOutlet weak var webView: WKWebView!                    2 Use of undeclared type 'WKWebView'

    @IBOutlet weak var backButton: UIButton!

    @IBOutlet weak var forwardButton: UIButton!

    @IBOutlet weak var urlTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

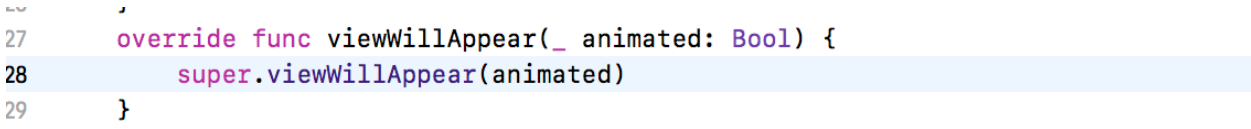
If you notice the compiler is giving us an error on the reference to the webview. This is because we need to import WebKit. Right under the like import UIKit type import WebKit and it should look like this

```
//
//  ViewController.swift
//  WebView Example
//
//  Created by Patrick Hill on 2/5/18.
//  Copyright © 2018 Patrick Hill. All rights reserved.
//

import UIKit
import WebKit

class ViewController: UIViewController {


    @IBOutlet weak var webView: WKWebView!

    @IBOutlet weak var backButton: UIButton!

    @IBOutlet weak var forwardButton: UIButton!

    @IBOutlet weak var urlTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

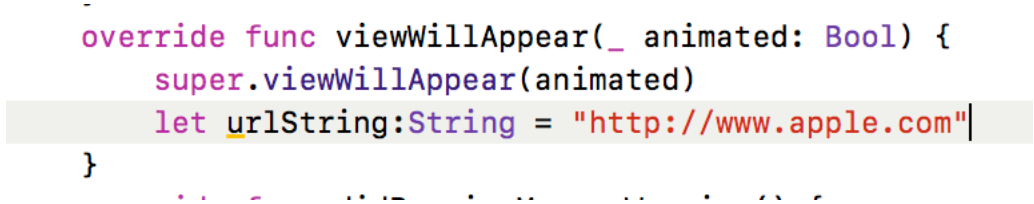
Now we will write some code that will display apple.com in the webview when the application runs. We are going to need to override a method viewWillAppear. Right after the viewDidLoad method create an overridden viewWillAppear function like so:



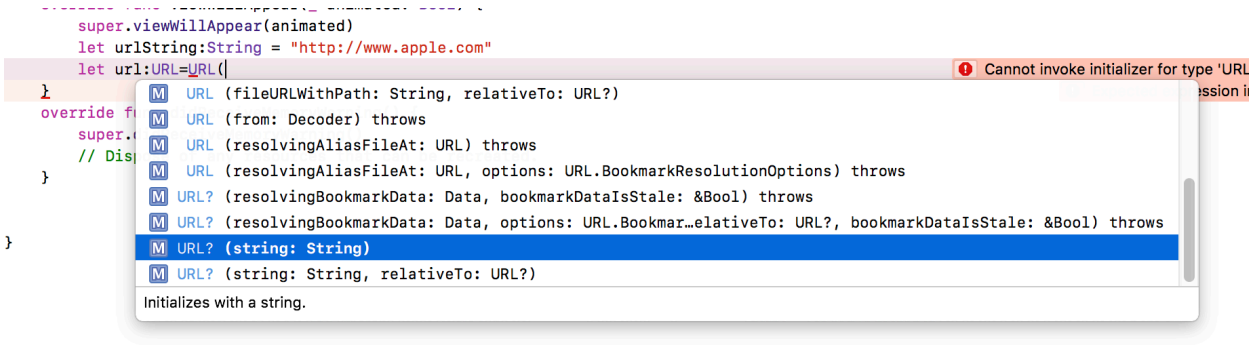
In the viewWillAppear method we first need to call the super class. Type super.viewWillAppear(animated) so now your code should look like this:



Next we are going to create a string variable to hold the url



Next we will create a url object and set it to the urlString:




```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    let urlString:String = "http://www.apple.com"
    let url:URL = URL(string: urlString)!
}
override func didReceiveMemoryWarning() {
```

Next we need to create a URLRequest and then pass the URL into the URLRequest

```
    }
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        let urlString:String = "http://www.apple.com"
        let url:URL = URL(string: urlString)!
        let urlRequest:URLRequest = URLRequest(url: url)

    }
```

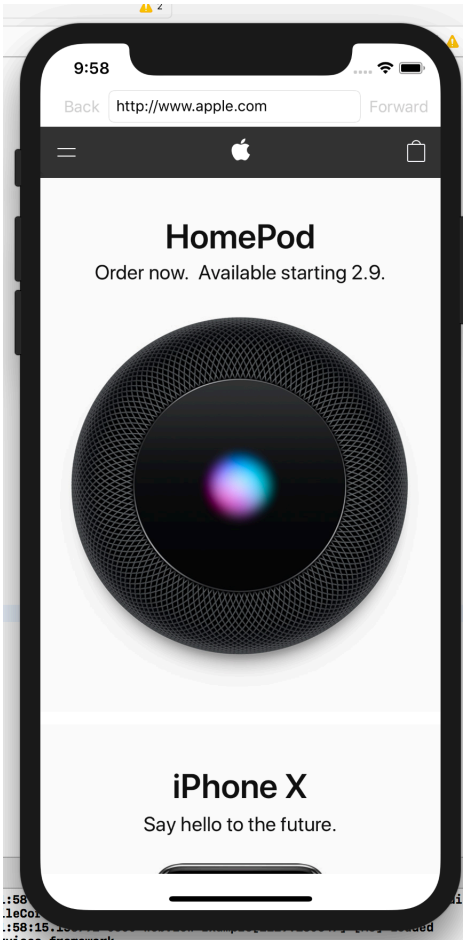
And then we are going to load the URLRequest into the WebView:

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    let urlString:String = "http://www.apple.com"
    let url:URL = URL(string: urlString)!
    let urlRequest:URLRequest = URLRequest(url: url)
    webView.load(urlRequest)

}
```

And then finally we are going to set the text of the text field to the URL string.

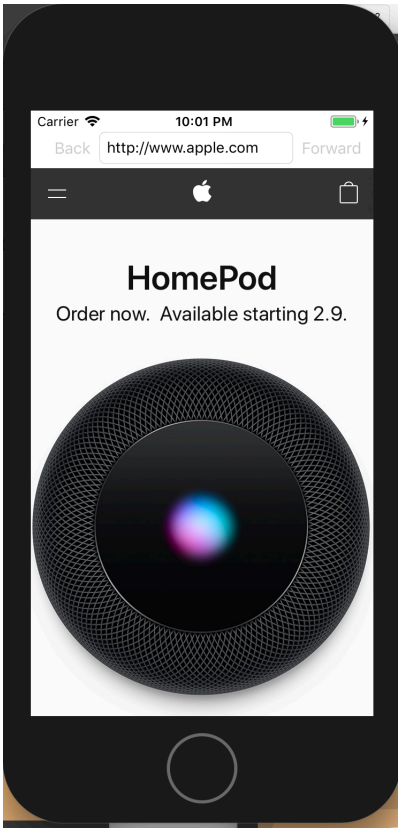
```
}
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    let urlString:String = "http://www.apple.com"
    let url:URL = URL(string: urlString)!
    let urlRequest:URLRequest = URLRequest(url: url)
    webView.load(urlRequest)
    urlTextField.text = urlString
}
```

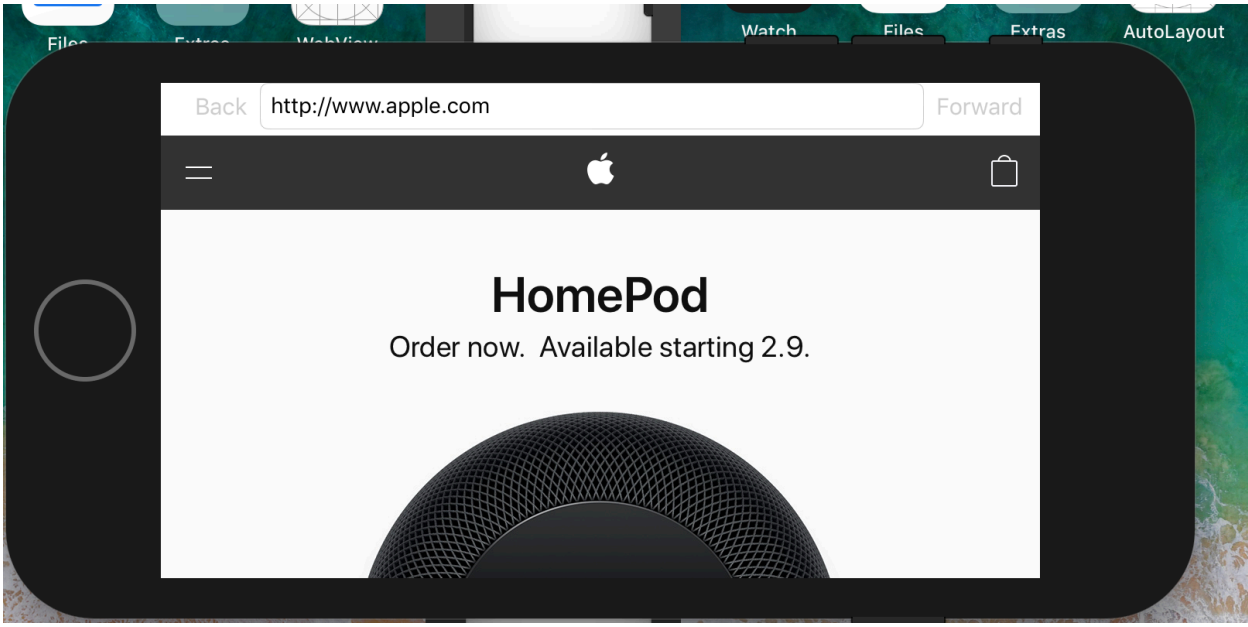
Now let's run the application and see if it's working!



As we can see the webview loads the page and displays the url in the text field!

Also let's see how it looks on a different size device.
IPhone SE:

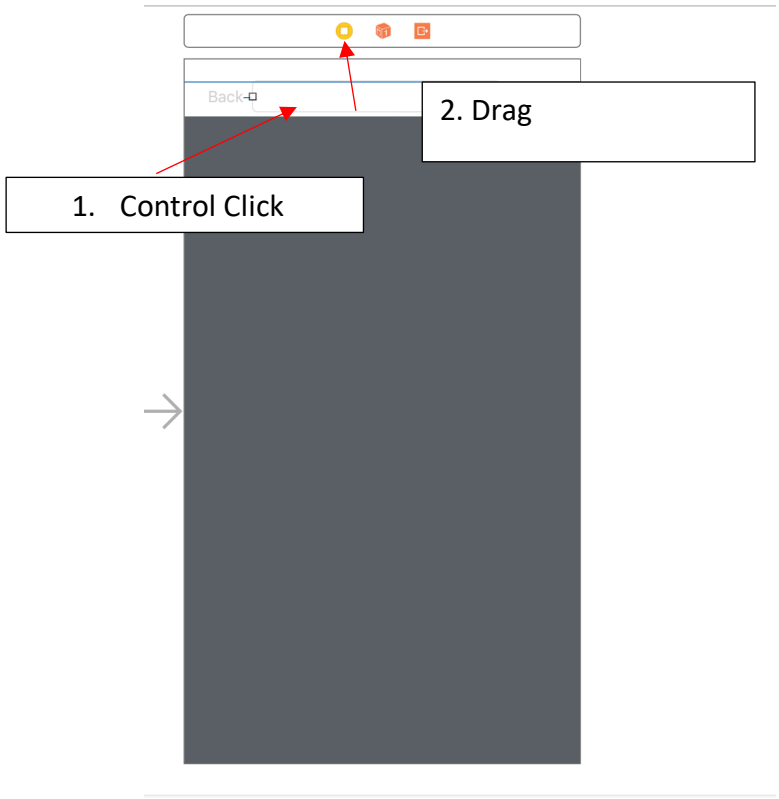




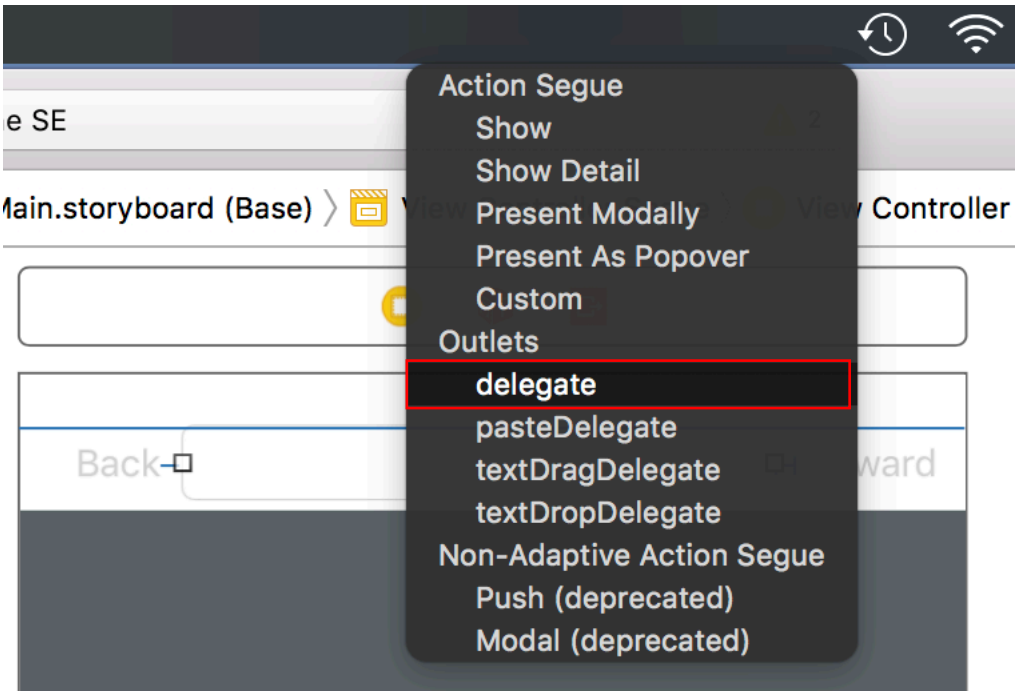
Now we are going to add in the functionality of if the user types a URL in the textfield and hits enter, the webview will load the URL.

For this to work we need to use a delegate. When you press enter in the text field, the text field lets its delegate know that the enter event has happened.

On the view controller, select the text field and then control drag it to the view icon as shown below



When the context menu pops up select delegate



So in this step we are telling the text field that it's delegate is the view controller.

The next step is to open the code window. Modify ViewController class definition to include UITextFieldDelegate

Before:

```
class ViewController: UIViewController {
```

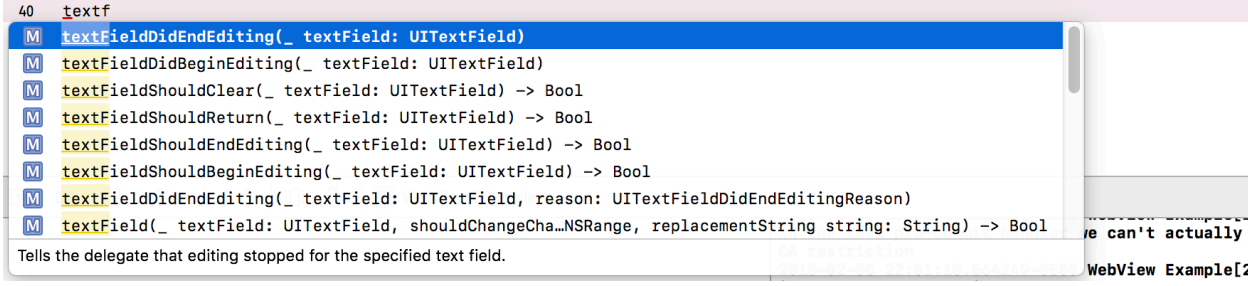
After:

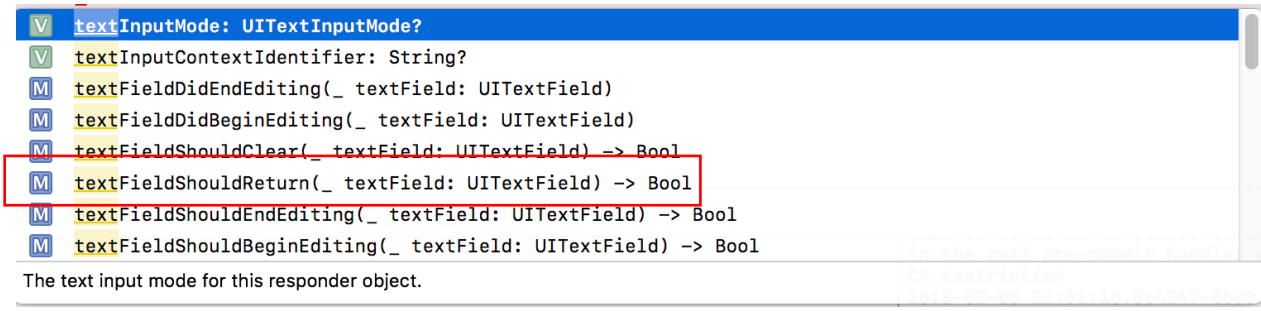
```
class ViewController: UIViewController, UITextFieldDelegate {
```

The next step is to create a method for when the user hits enter.

At the bottom of the code just start typing "text" the context menu appears and shows us the various events that are available:

```
//

import UIKit
import WebKit

class ViewController: UIViewController, UITextFieldDelegate {


    @IBOutlet weak var webView: WKWebView!

    @IBOutlet weak var backButton: UIButton!

    @IBOutlet weak var forwardButton: UIButton!

    @IBOutlet weak var urlTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        let urlString:String = "http://www.apple.com"
        let url:URL = URL(string: urlString)!
        let urlRequest:URLRequest = URLRequest(url: url)
        webView.load(urlRequest)
        urlTextField.text = urlString
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

40  textf
```





You can see the various methods, Did End editing, Did Begin Editing etc.. Since we want it when the user hits return we will select textFieldShouldReturn

It then creates that function for us:

```
import UIKit
import WebKit

class ViewController: UIViewController, UITextFieldDelegate {


    @IBOutlet weak var webView: WKWebView!

    @IBOutlet weak var backButton: UIButton!

    @IBOutlet weak var forwardButton: UIButton!

    @IBOutlet weak var urlTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        let urlString:String = "http://www.apple.com"
        let url:URL = URL(string: urlString)!
        let urlRequest:URLRequest = URLRequest(url: url)
        webView.load(urlRequest)
        urlTextField.text = urlString
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        code
    }
}
```

So first we are going create a variable and then store the value of the textfield into the variable

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    let urlString:String = urlTextField.text!
}
```

Now we are going to add the code to tell the webview to load that URL. We can copy the code we typed into the viewWillAppear method:

```
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        let urlString:String = urlTextField.text!
        let url:URL = URL(string: urlString)!
        let urlRequest:URLRequest = URLRequest(url: url)
        webView.load(urlRequest)
        urlTextField.text = urlString
    }
}
```

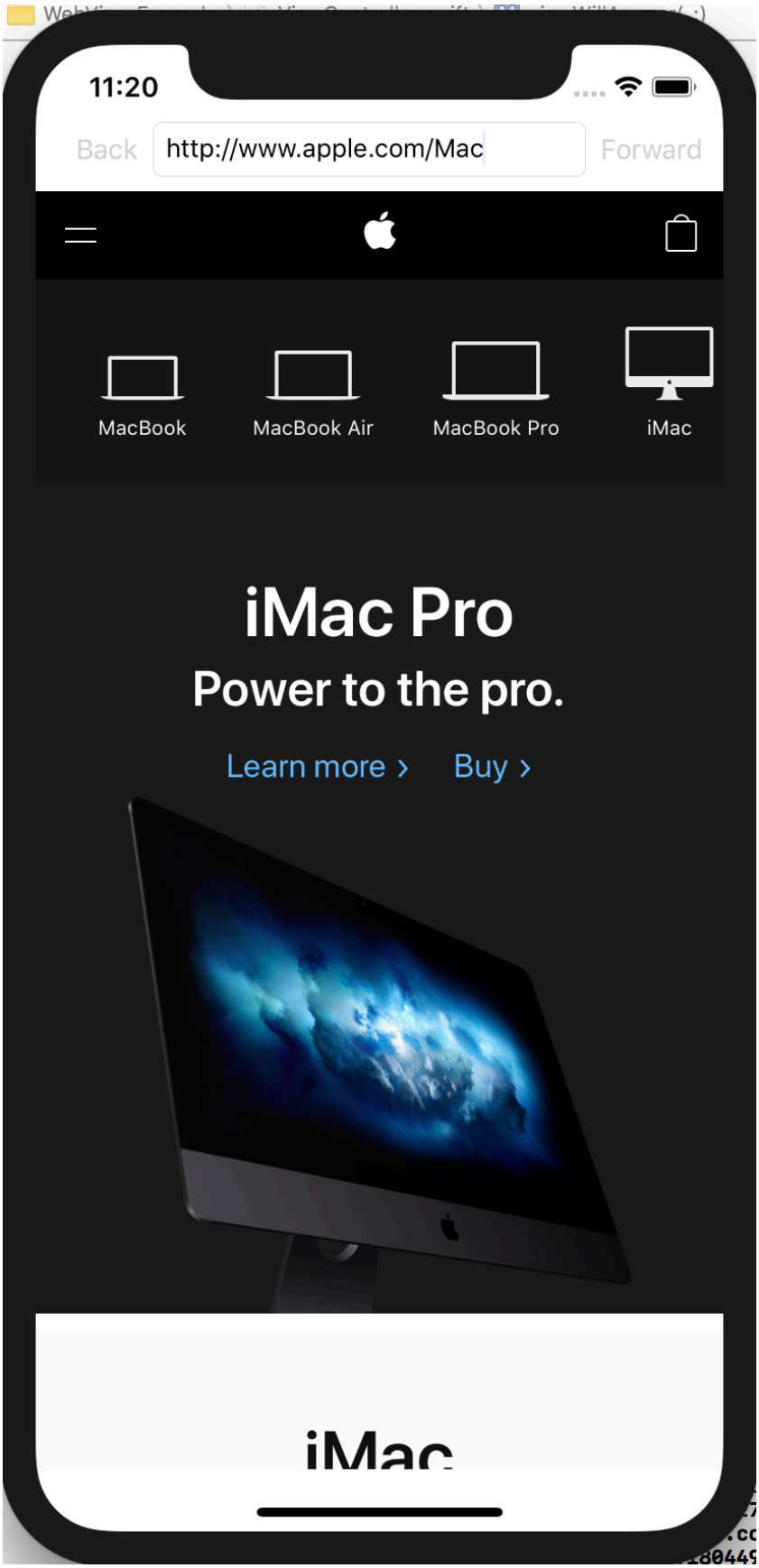
We also have to tell the system that when the user hits return to hide the keyboard. We use resignFirstResponder to do that like so:

```
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        let urlString:String = urlTextField.text!
        let url:URL = URL(string: urlString)!
        let urlRequest:URLRequest = URLRequest(url: url)
        webView.load(urlRequest)
        urlTextField.text = urlString
        textField.resignFirstResponder()
    }
```

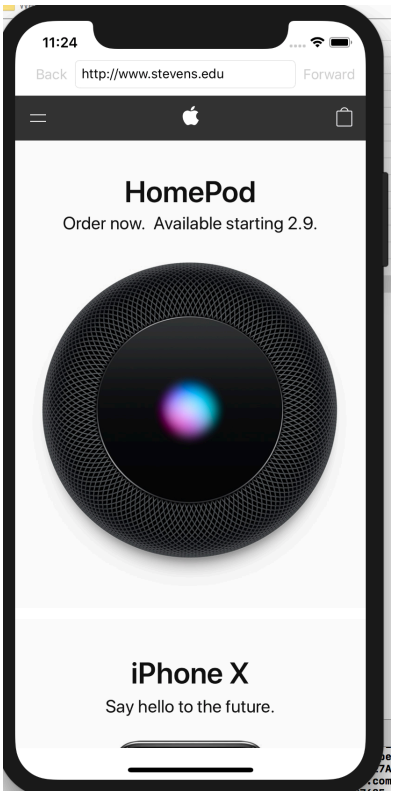
And then finally since this method returns a Boolean we can just return true:

```
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        let urlString:String = urlTextField.text!
        let url:URL = URL(string: urlString)!
        let urlRequest:URLRequest = URLRequest(url: url)
        webView.load(urlRequest)
        urlTextField.text = urlString
        return true
    }
```

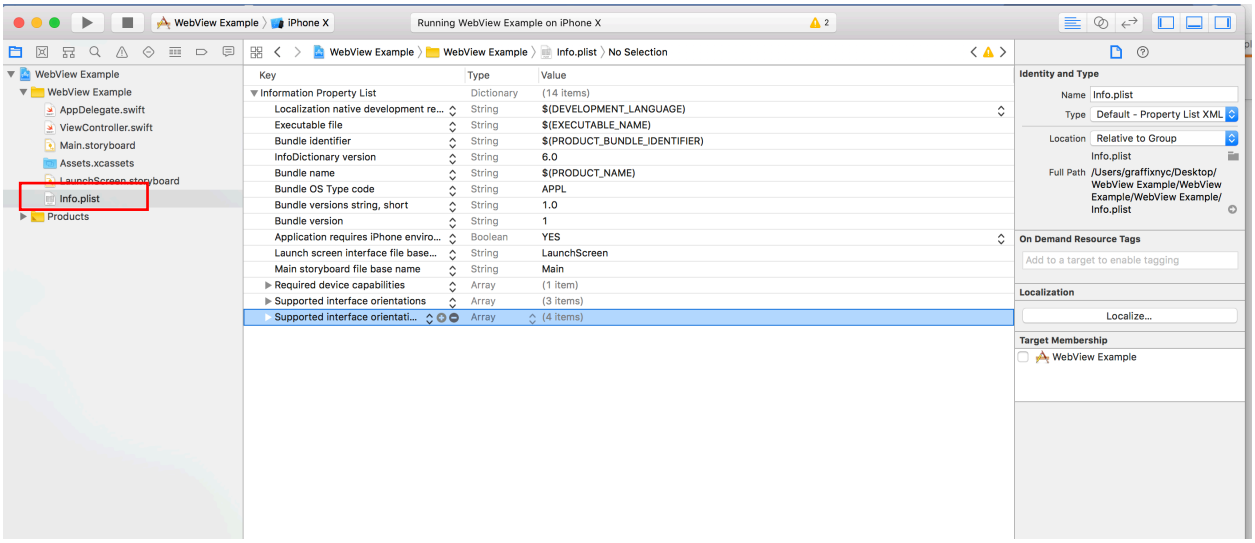
Now if we run the app and add /mac to the end of the URL in the textfield it loads


11:20
Back
http://www.apple.com/Mac
Forward
MacBook
MacBook Air
MacBook Pro
iMac
iMac Pro
Power to the pro.
Learn more ›
Buy ›
iMac

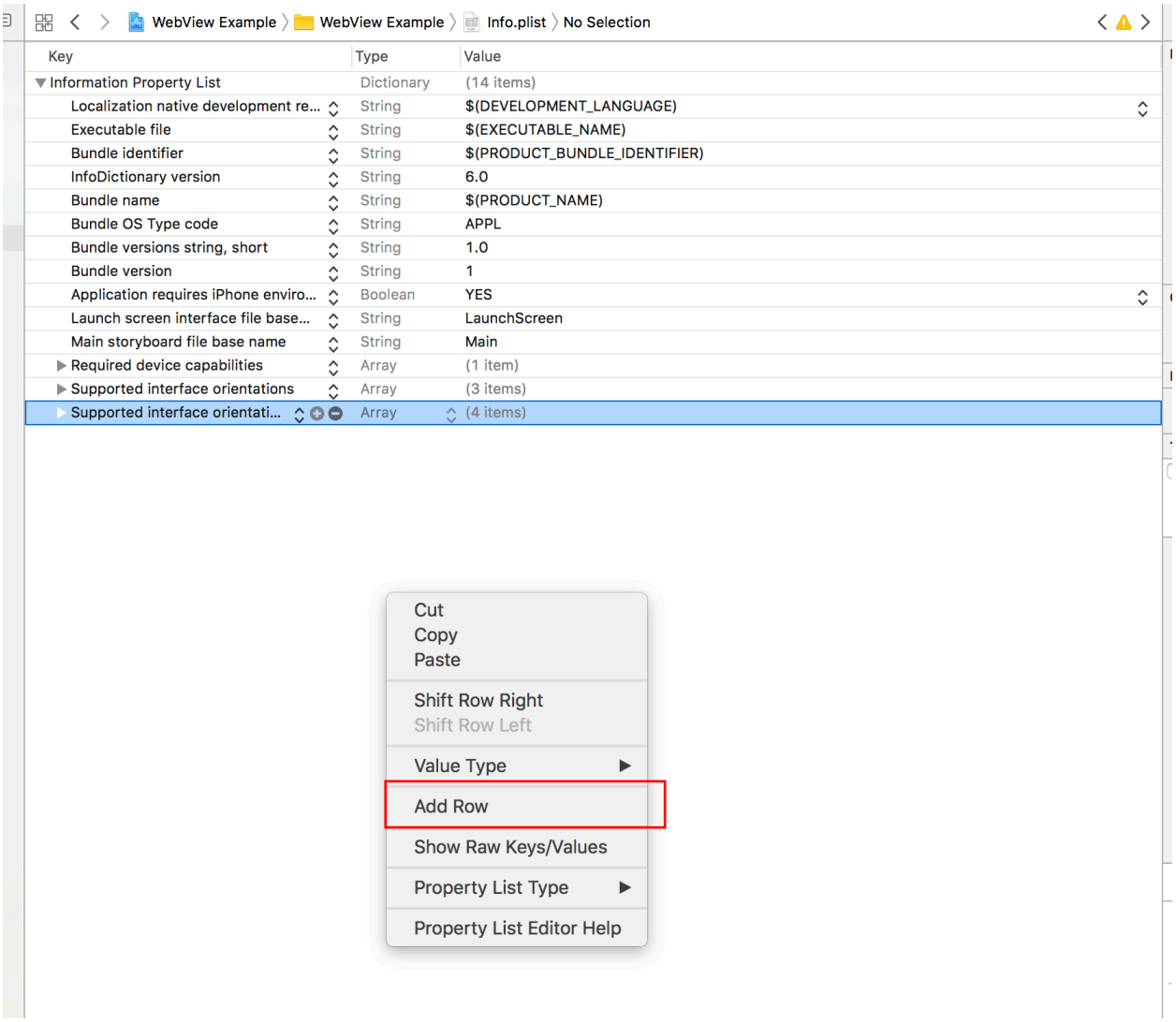
But what happens if we try going to stevens.edu?
You'll notice the page doesn't load. We need to give our app permission!



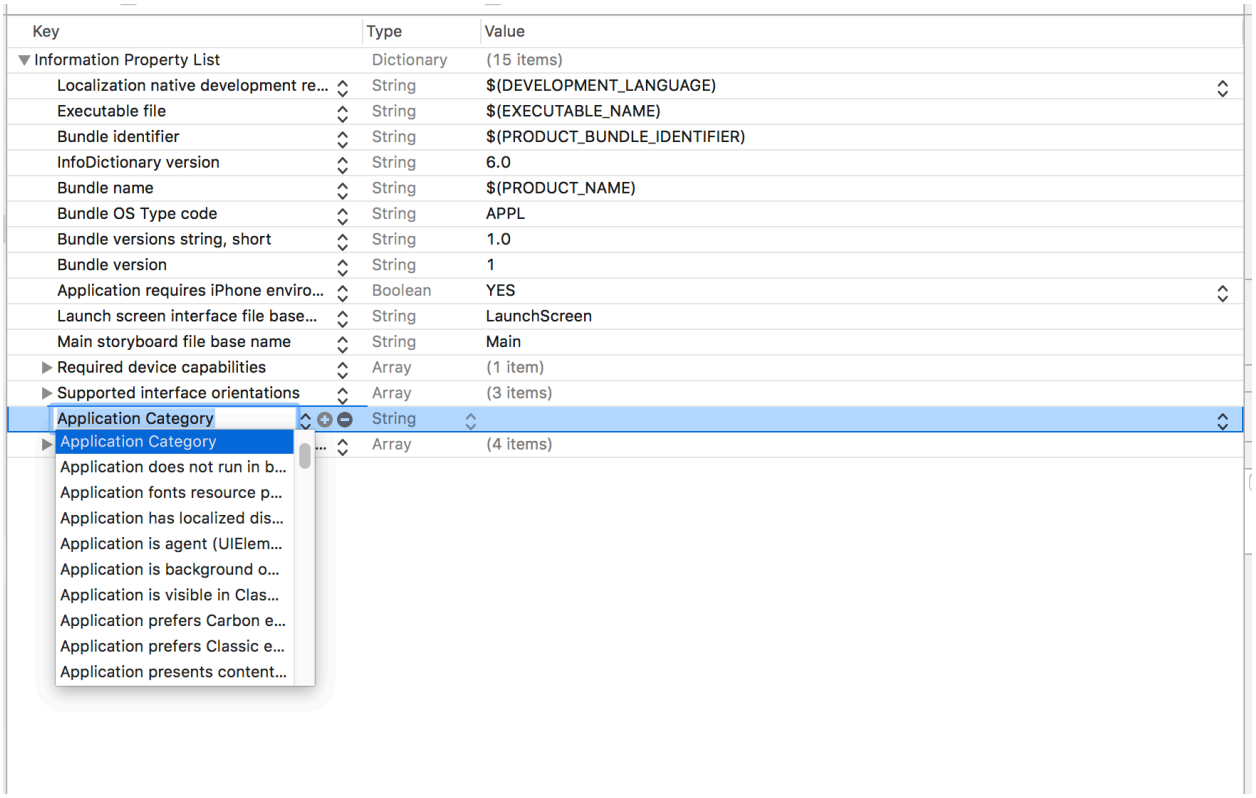
In the project view double click and open info.plist.



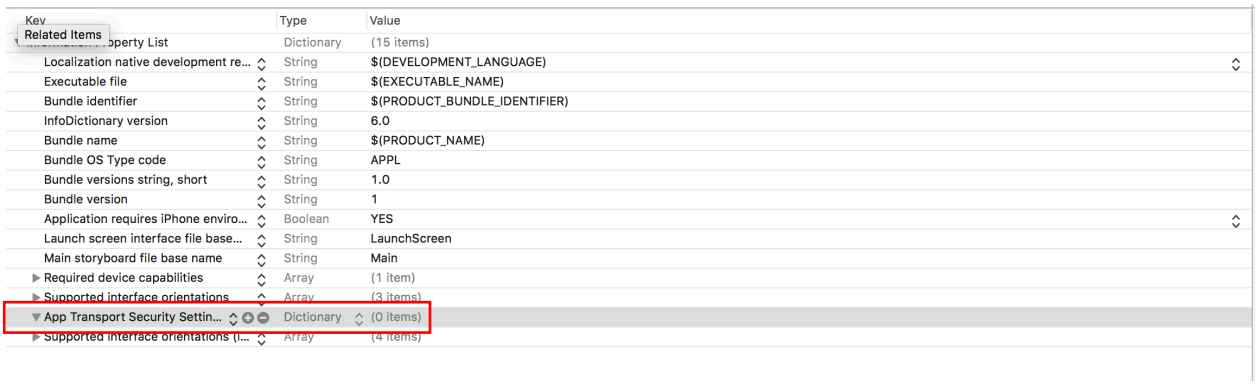
Right click (Control click) in the empty space in the bottom and select Add Row



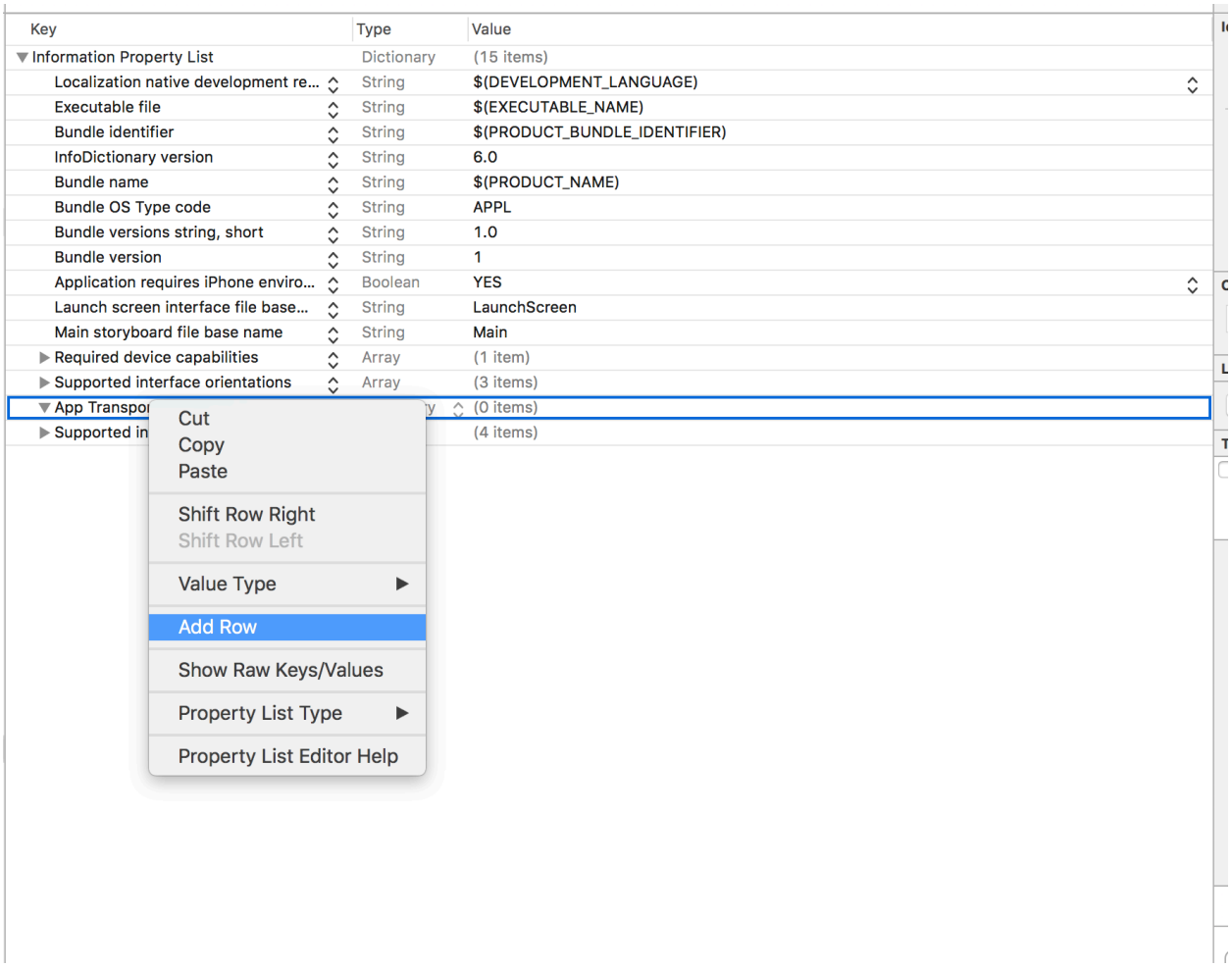
When we select add row it adds a new row to the plist.



Type in “App Transport Security Settings”

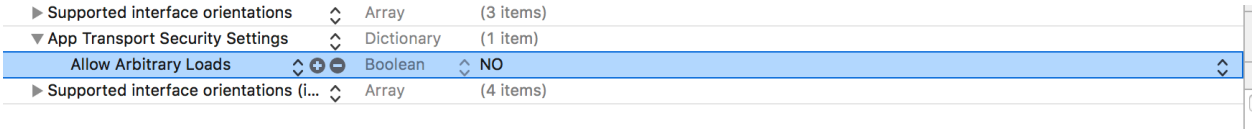


With it highlighted right click (control click the entry and then select add row again.

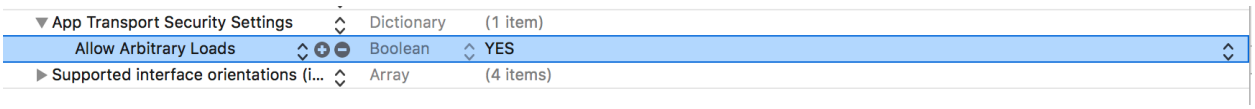


It will create an entry “Allow Arbitrary Loads” and where the value defaults to “NO” Change it to YES:

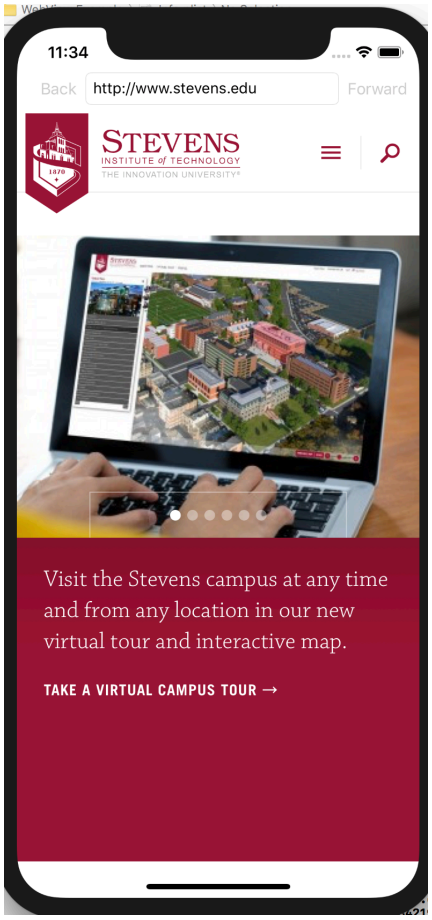
Before:



After:

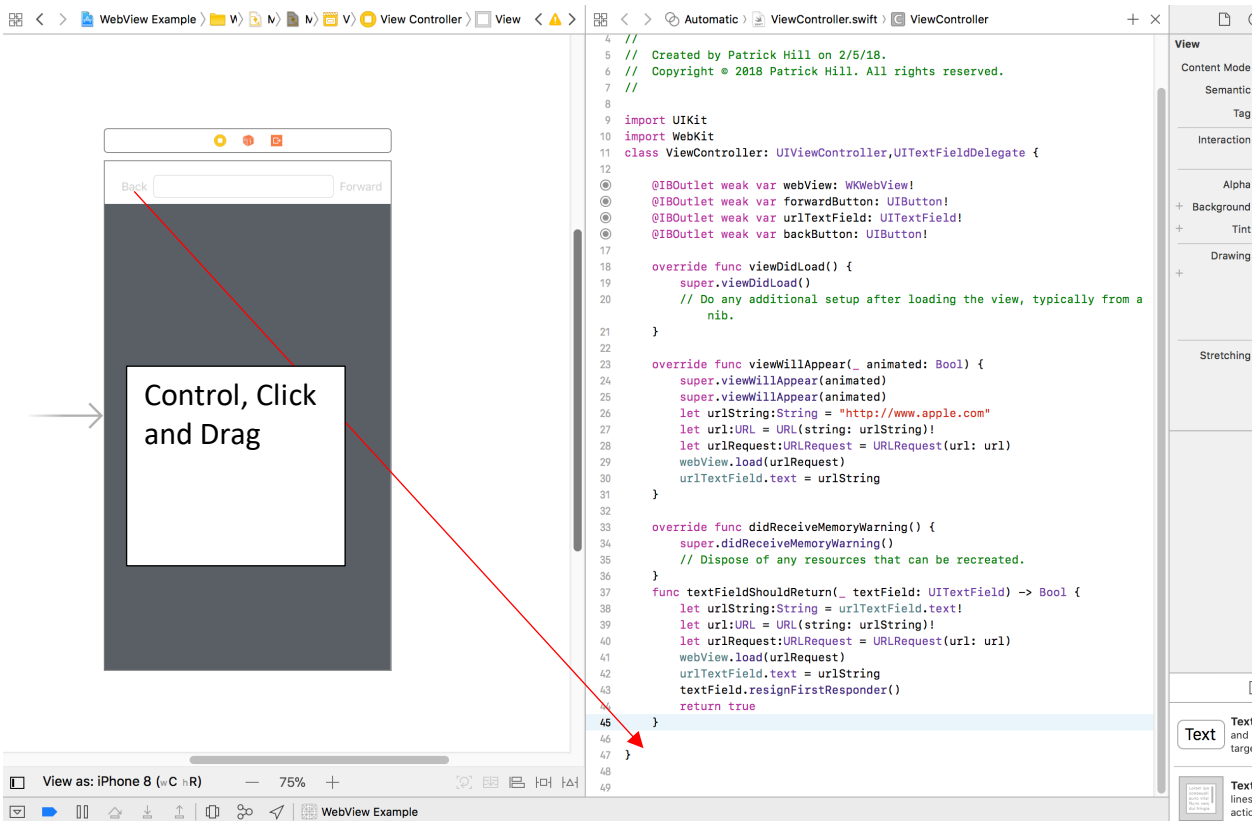


Now if we run the application and try going to Stevens.edu it now works!

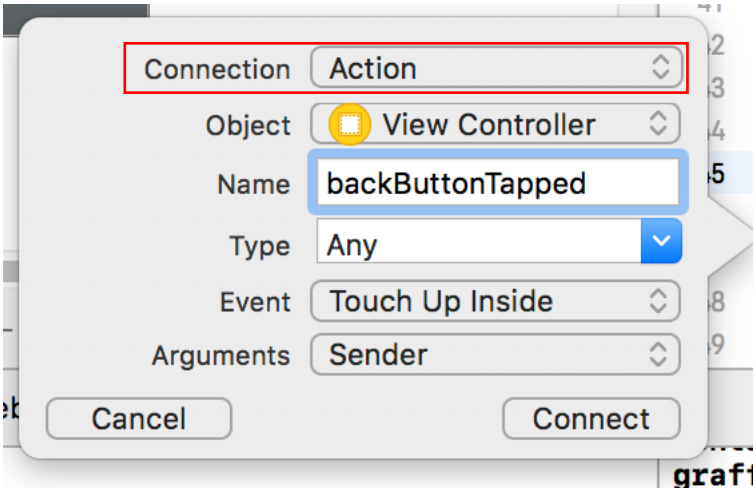


Now the final things we need to do are:
Right now, if you're on a webpage and you click a link on the webpage, the URL in the text field does not change, Also the back and forward buttons need to be done.

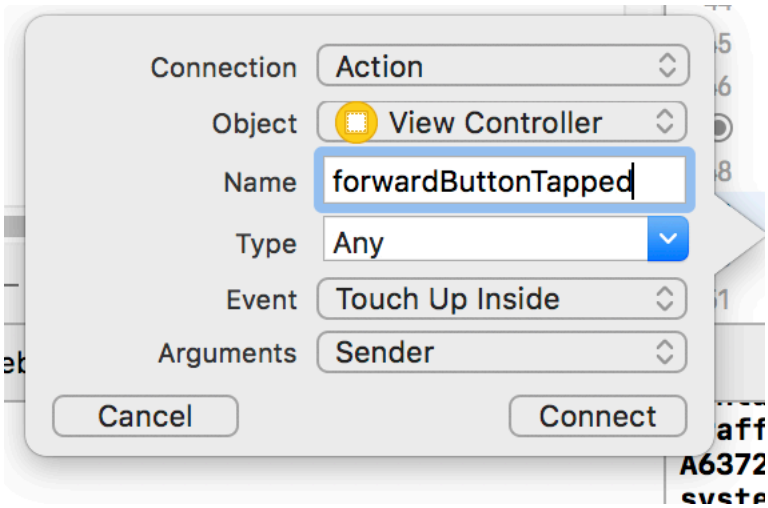
Go back to your Open up the Assistant Editor again, and we are going to control click and drag the back button to the code window again. This time, instead of creating an outlet, we will create an action.



When the context menu appears, make sure you change “outlet” to “Action” and name it backButtonTapped and then click “connect”



Do the same thing for the forward button and name it forwardButtonTapped



It creates the two actions and we can see the methods in the code:

```
@IBAction func backButtonTapped(_ sender: Any) {
}

@IBAction func forwardButtonTapped(_ sender: Any) {
}
```

So now in the backButtonTapped method, we will first check to see if we can go back, if we can, then we tell the webview to go back like so:

```
@IBAction func backButtonTapped(_ sender: Any) {
    if webView.canGoBack{
        webView.goBack()
    }
}
```

And we will do the same for forward (except we check if we can go forward instead of back) So that method should look like this:

```
@IBAction func forwardButtonTapped(_ sender: Any) {
    if webView.canGoForward{
        webView.goForward()
    }
}
```

Now the buttons are still disabled by default, so now we will change that so they are enabled if there is a back/forward history.

So we need to listen to when the webview is done loading, and if we have back or forward history, then we will enable the buttons.

So in the class definition we are going to add another delegate.

Before:

```
import WebKit
class ViewController: UIViewController,UITextFieldDelegate {
```

So we will add WKNavigationDelegate to the class definition after UITextFieldDelegate like this:

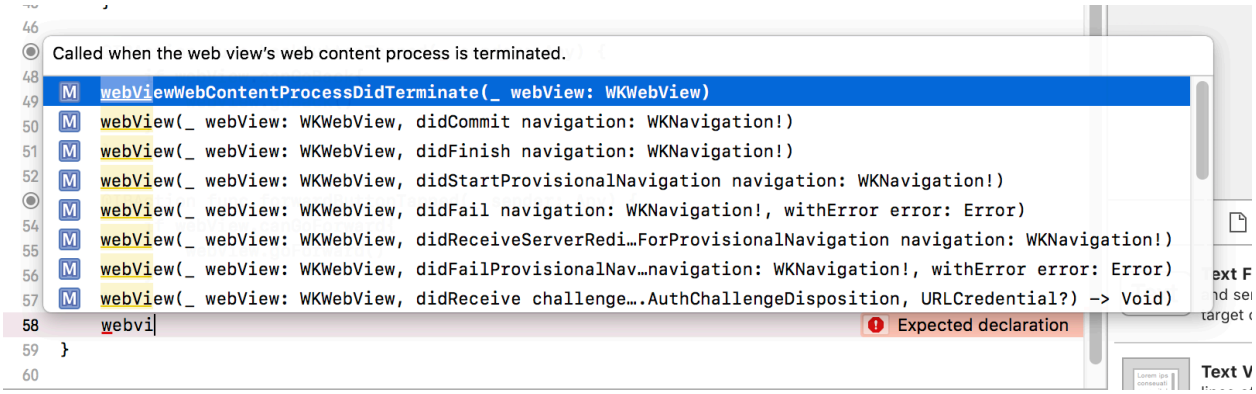
After:

```
import WebKit
class ViewController: UIViewController,UITextFieldDelegate,WKNavigationDelegate {
```

Next we have to add the delegate to the viewDidLoad function so it knows to listen for it. In the viewDidLoad function type: webView.navigationDelegate = self so it looks like so:

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    webView.navigationDelegate = self
}
```

After the button actions in the code type “webview” and a list will come up



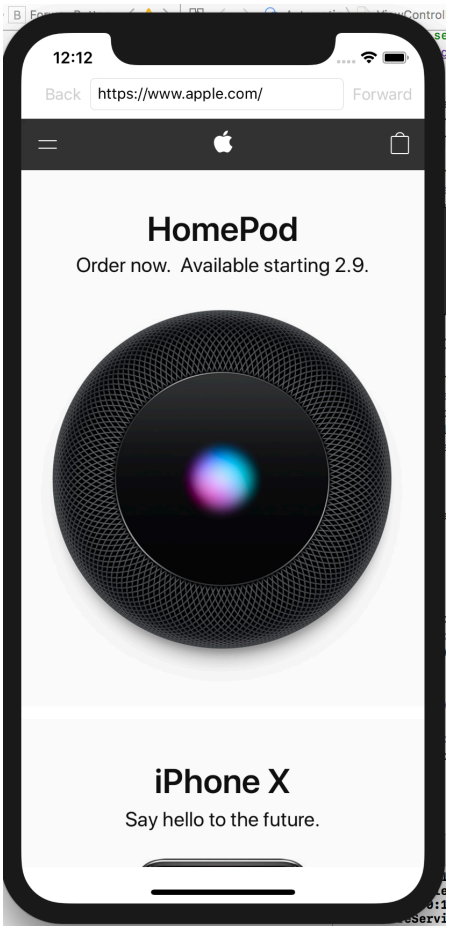
We want the one “didFinish”

```
func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
    code
}
```

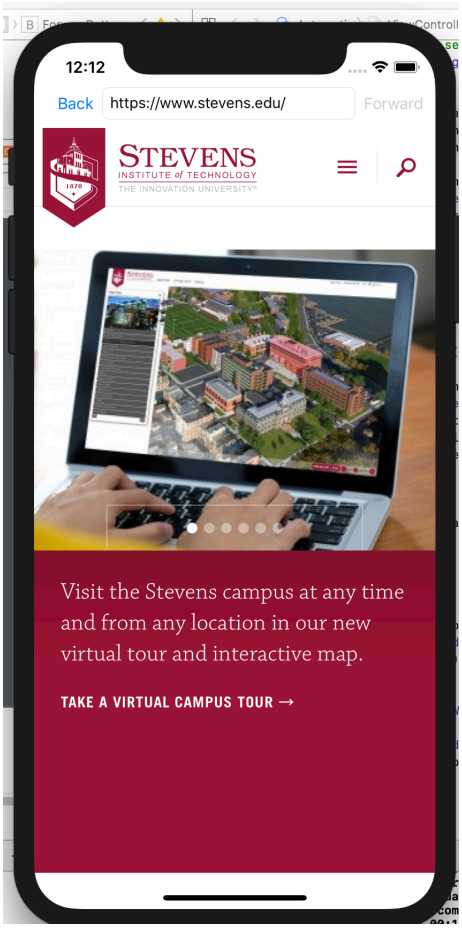
Now in the didFInish method we just need to enable the buttons if there is a history. So we set backButton.isEnabled = webView.canGoBack and the forwardButton.isEnabled = webView.canGoForward and then finally, we set the url textfield text to display the url like so:

```
}
func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
    backButton.isEnabled = webView.canGoBack
    forwardButton.isEnabled = webView.canGoForward
    urlTextField.text = webView.url?.absoluteString
}
```

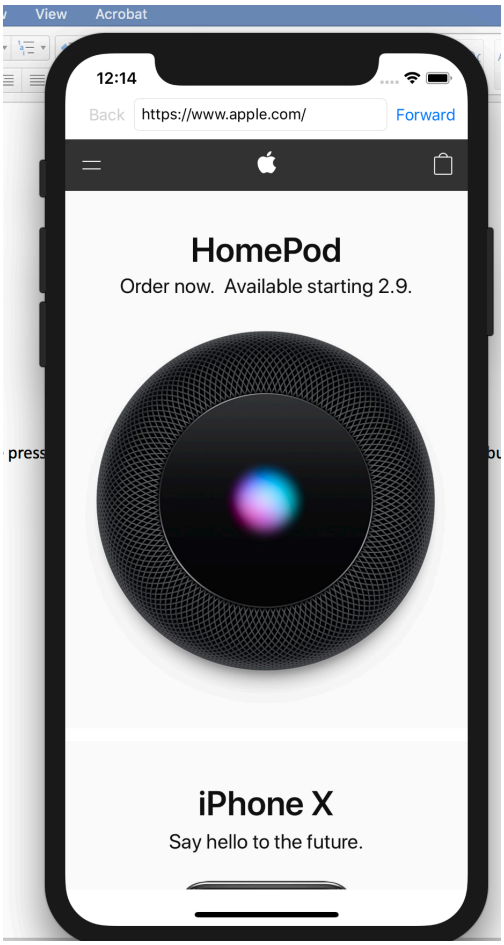
And that is it! Now if we run the application we first see the buttons are disabled:



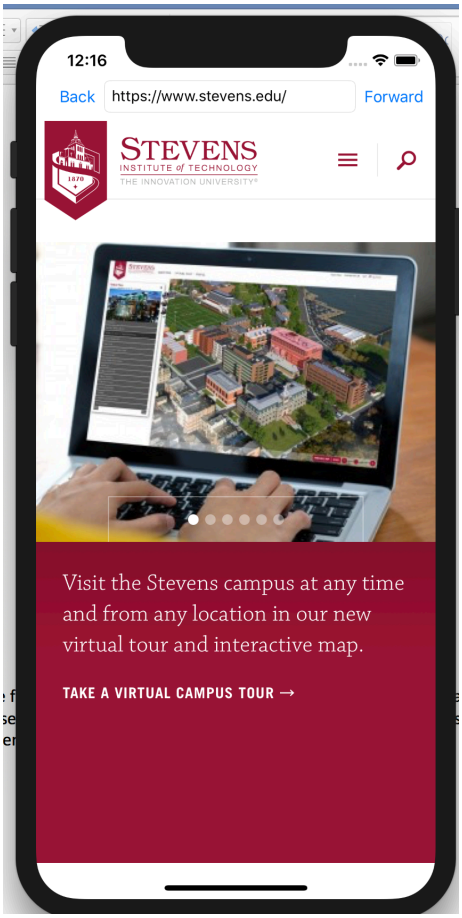
Then if we navigate to stevens.edu we see the back button is now enabled:



And then if we press the back button and go back to apple.com we see the back button will be disabled and the forward button will be enabled:



And if we tap the forward button to go back to the Stevens site, and then click a link on the Stevens Page we see that the new url is displayed in the text field, and if we press back again then both the back buttons and forward buttons are enabled:



Here are a few challenges for you to do to make the app better:

1. Right now if you enter an invalid URL, The app crashes. So the challenge is to modify the app so it checks the input for a valid URL before it sends a request
2. If there is no connection and you try to load a page the app does nothing. So the challenge is to modify the app to check if there is an internet connection before attempting to perform the request, if there is no connection alert the user and tell them to try later
3. If you go to a valid url but the host does not exist, nothing happens, So the challenge is to display an error message if the host does not exist.
4. If you enter a valid URL and add an extra space in it, it crashes the app. Modify the app so it trims the whitespace from the URL
5. When you tap into the textfield make it so the current text is selected (like how other browsers work when you click into the address bar, it selects all the text in the address bar