

<https://github.com/GroundSpeed/BreakingGround>

BREAKING GROUND WITH IOS

August 24, 2014
1:30p - 5:00p

Don Miller
Founder of GroundSpeed™
Co-founder of Seed Coworking
@donmiller



Don Miller

GroundSpeed™
rapid web + mobile software

SCHEDULE

- Introductions
- Discuss Terminology, Xcode Overview, Hello World
- Break
- Command Line and GitHub
- NSDictionary, NSArray, and Objective-C
- Storyboards
- UITableViews (if we have time)

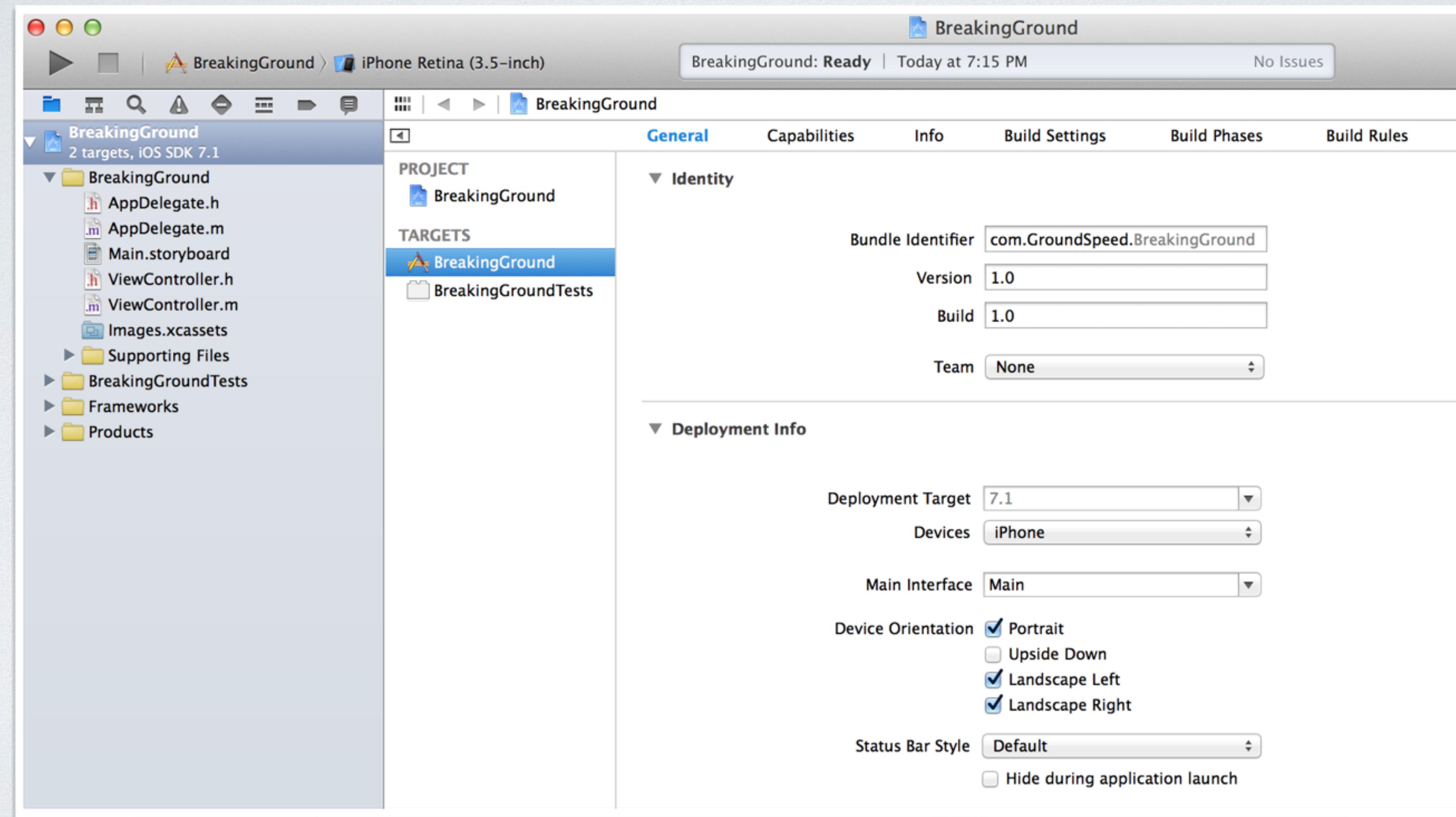


TERMINOLOGY

- ▶ iOS
- ▶ iPhone
- ▶ iPad
- ▶ iPod
- ▶ MacBook
- ▶ iMac
- ▶ SSD
- ▶ SDK
- ▶ IDE
- ▶ Xcode
- ▶ OSX
- ▶ Mavericks
- ▶ Mountain Lion
- ▶ Lion
- ▶ Snow Leopard, etc.
- ▶ Retina



XCODE 5.1.1 & IOS 7.1

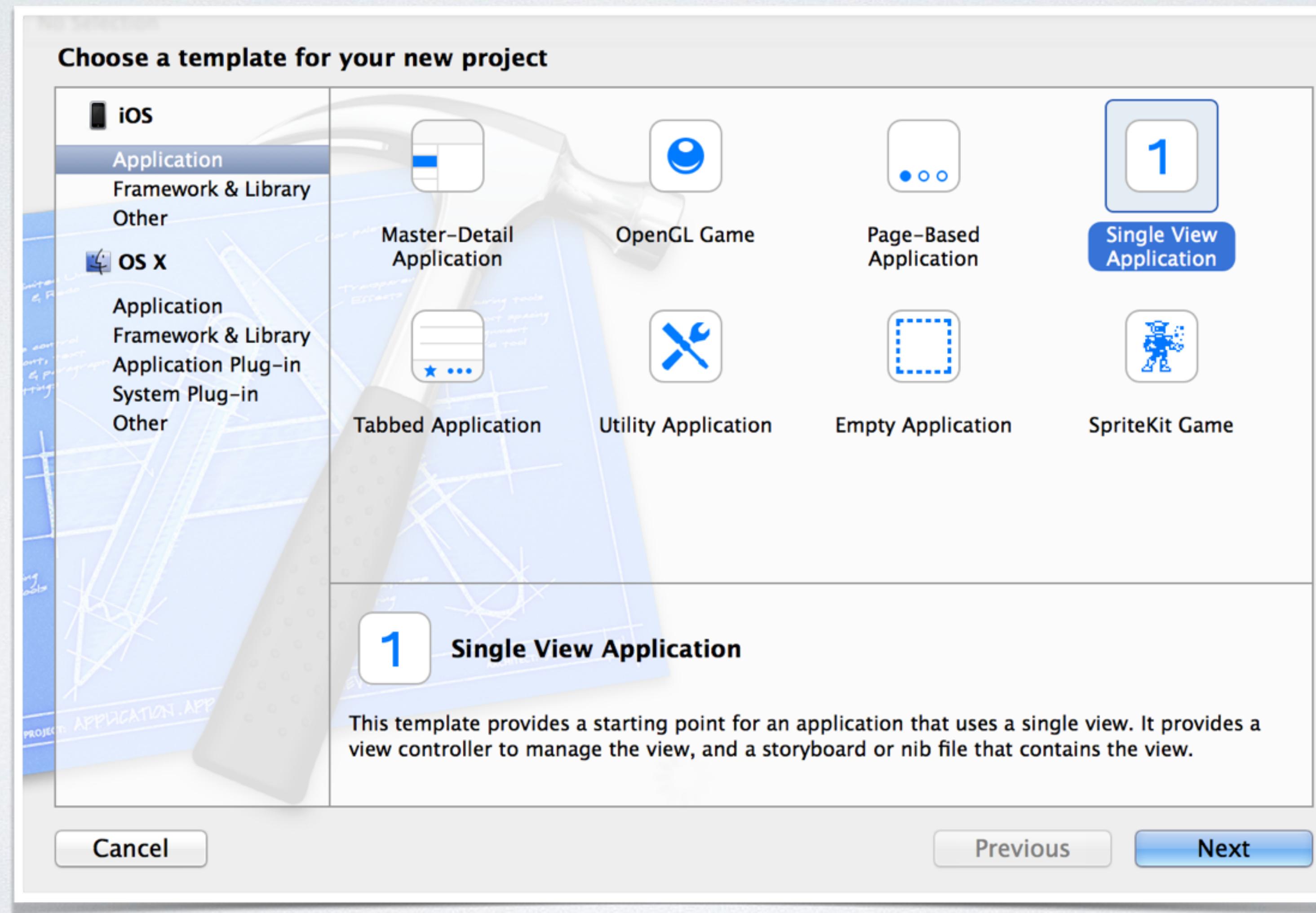


HELLO WORLD!

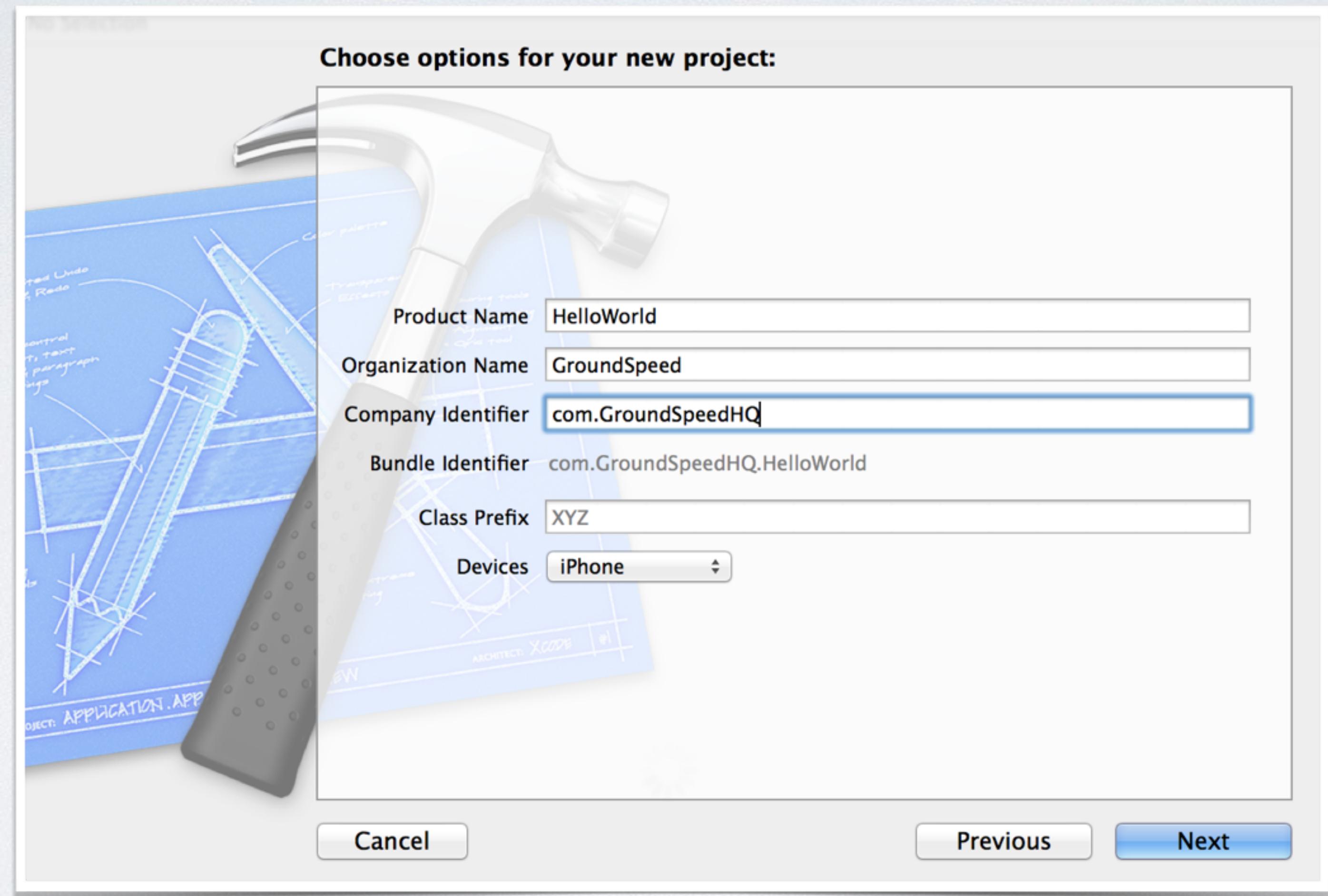


GroundSpeed™
rapid web + mobile software

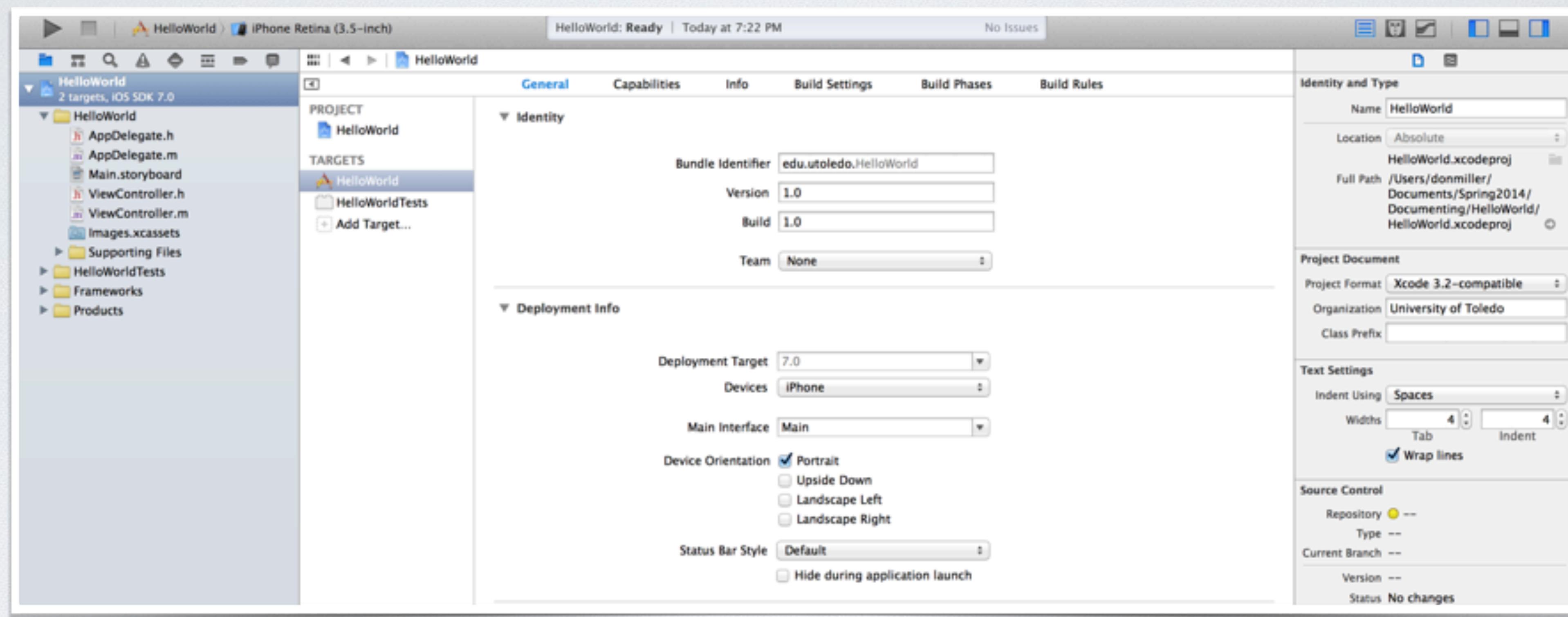
CREATE A NEW SINGLEVIEW APPLICATION



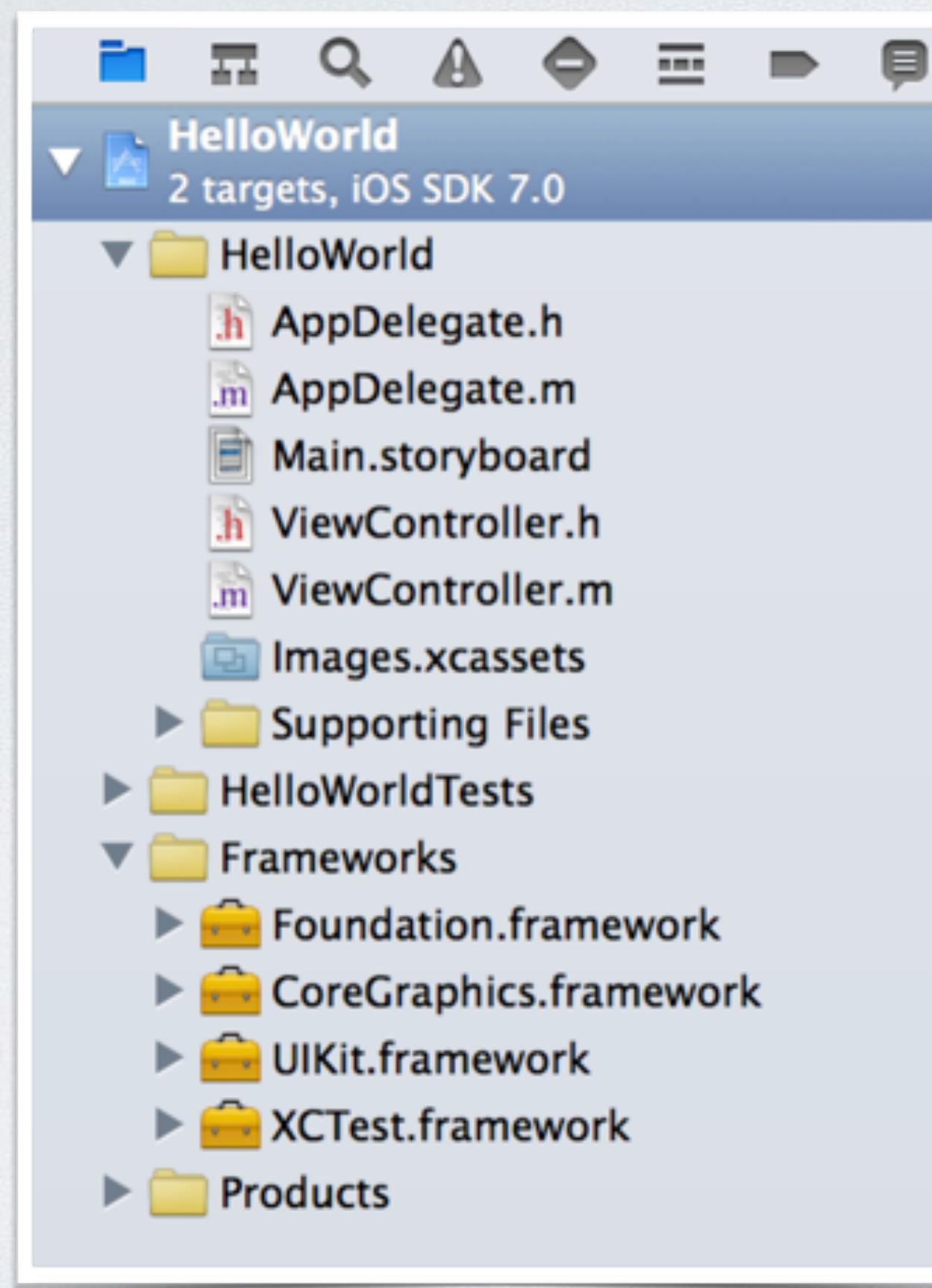
ENTER THE PRODUCT NAME



UPDATE THE GENERAL PROJECT ATTRIBUTES



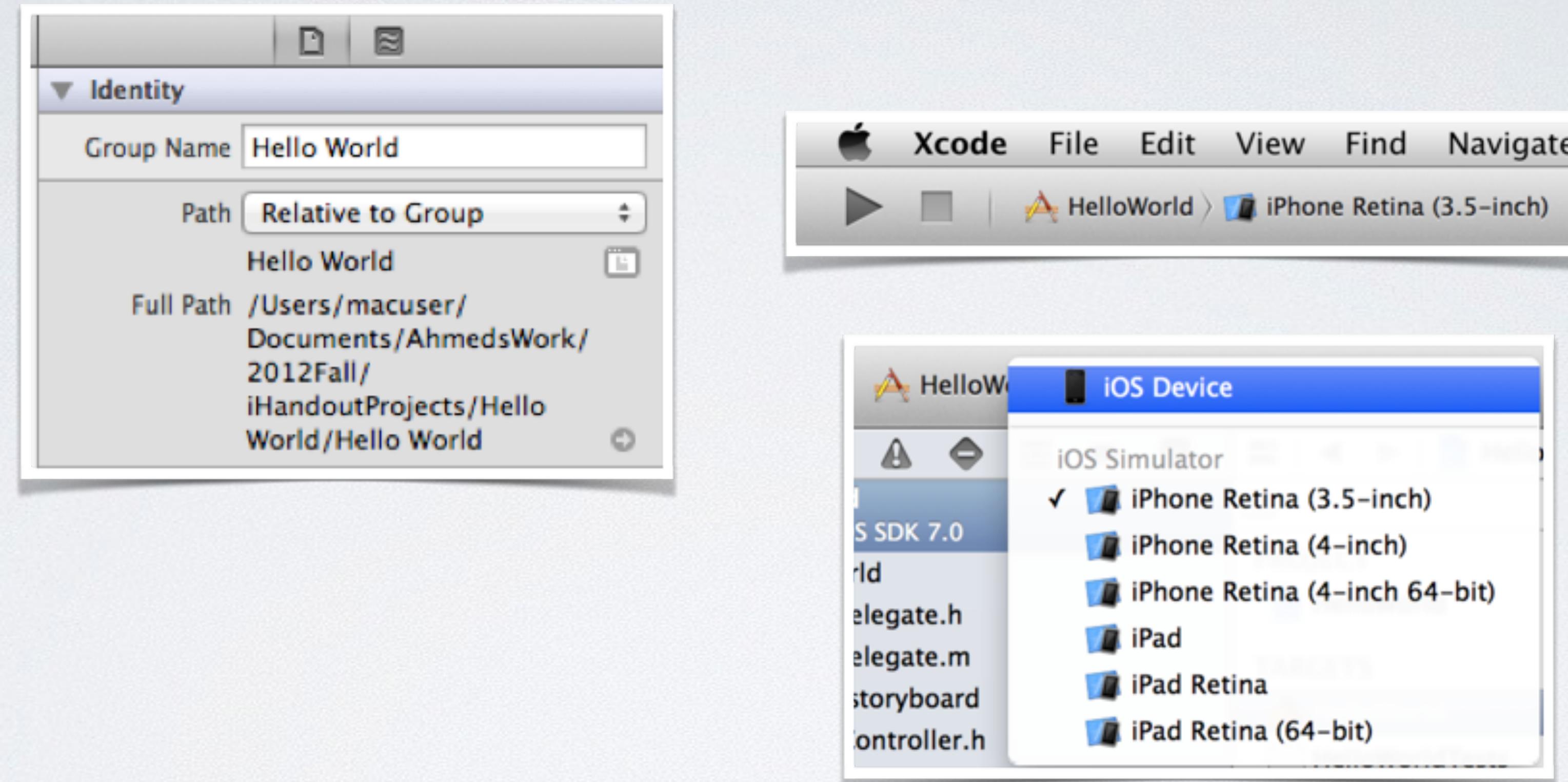
PROJECT STRUCTURE



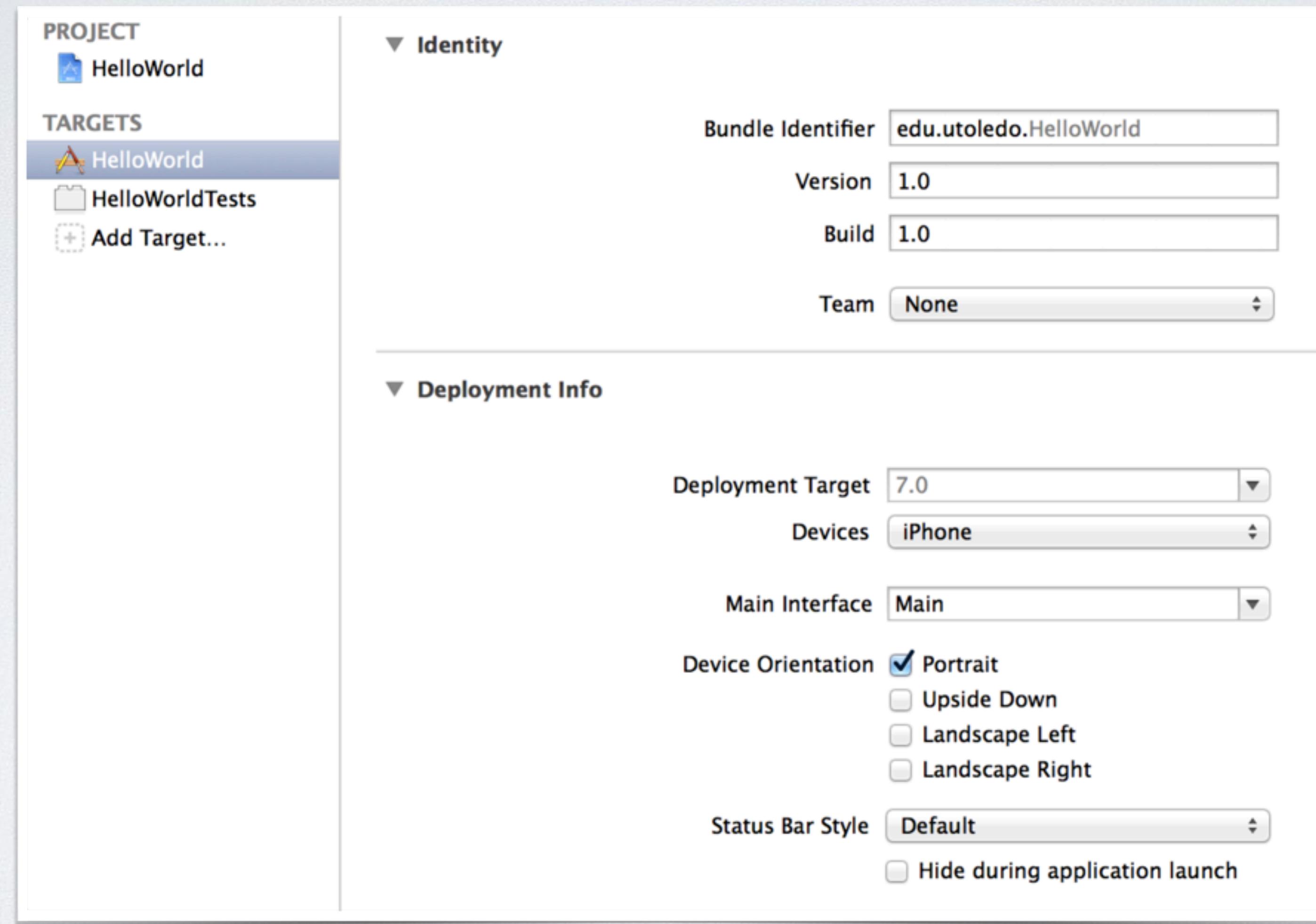
- Folders do not match physical layout on file system.
- Actually you will not find any sub folder named like Supporting Files in your physical project folder.
- The folder hierarchy in the XCode's navigator is simply a logical grouping of the resources.
- At this stage, we would be interested only on the Main.storyboard file.



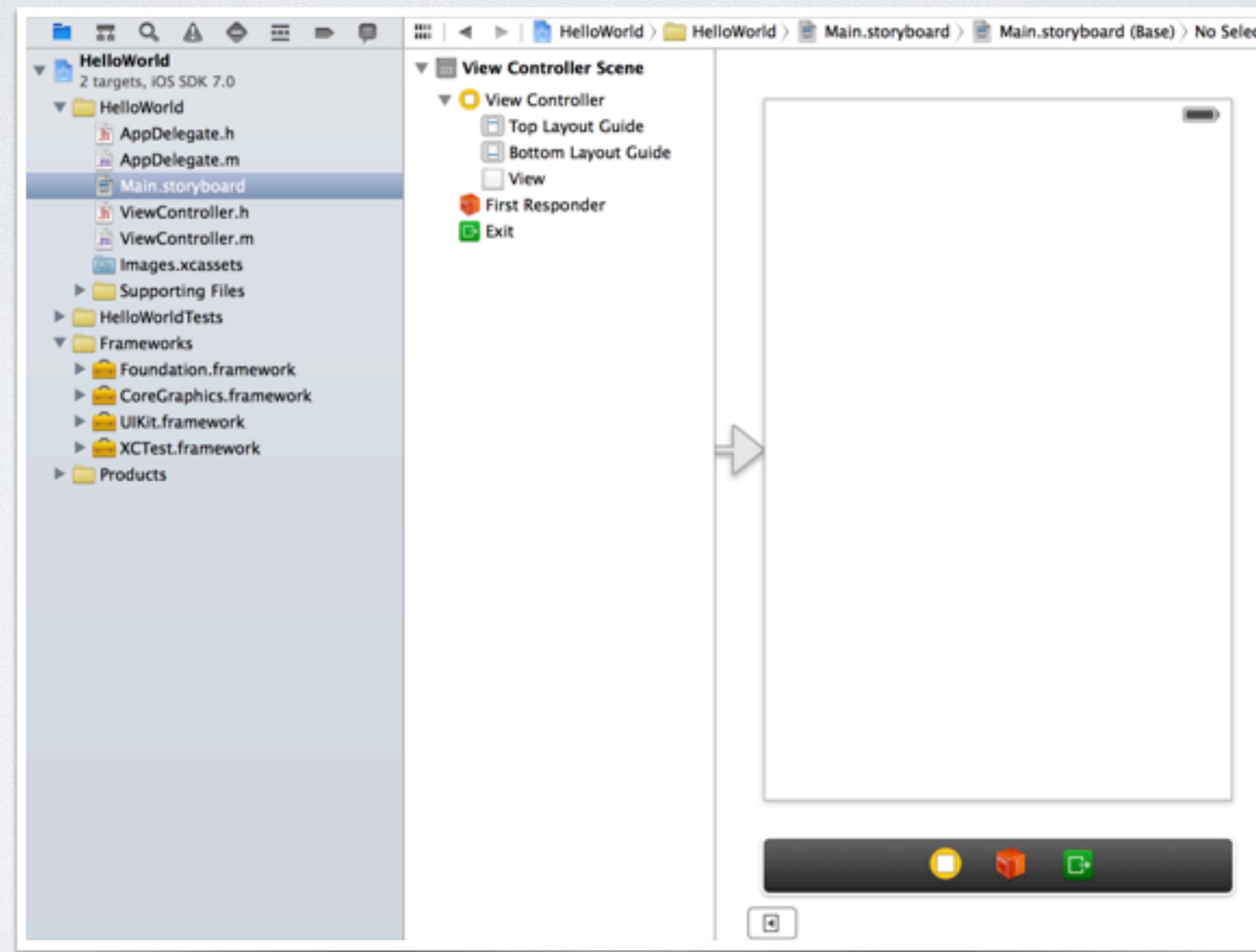
FILE INSPECTOR & SCHEMA SETTINGS



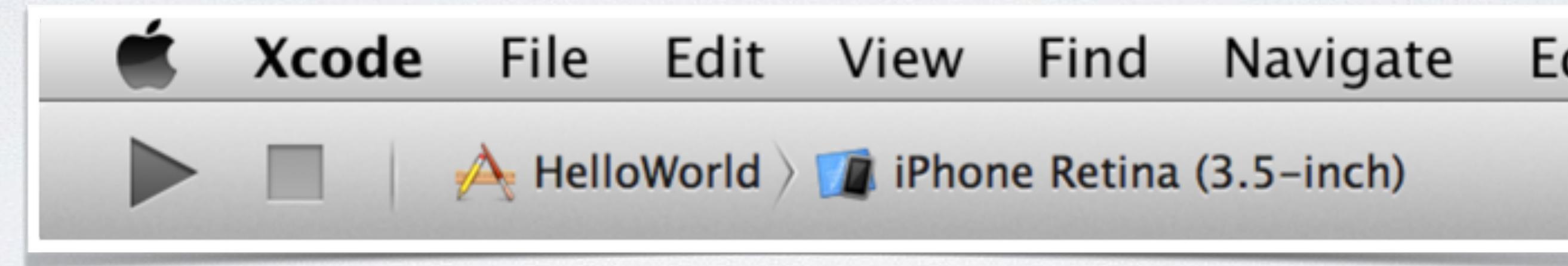
DEVICES & TARGETS



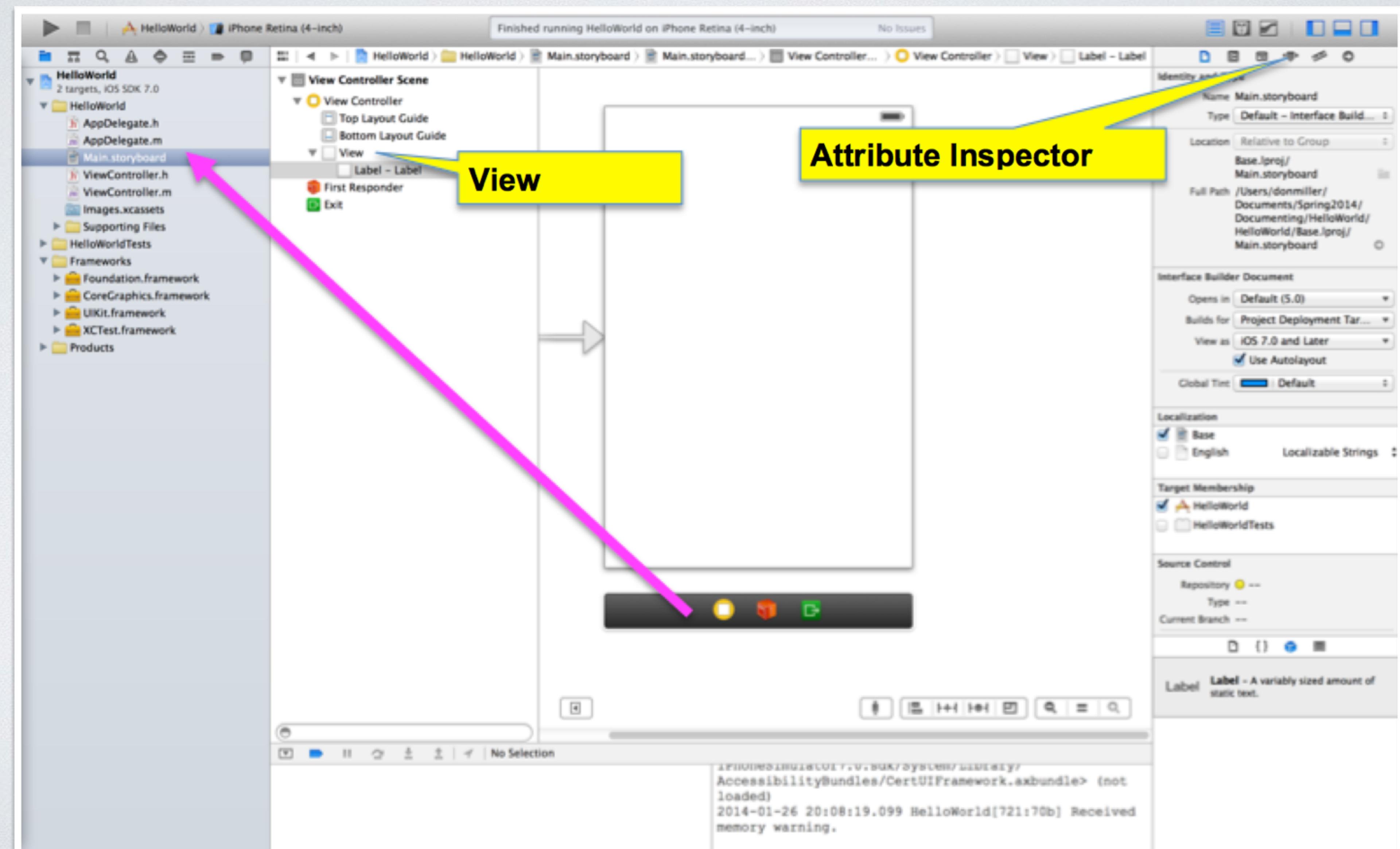
FILE OWNER, VIEW, & FIRST RESPONDER



RUN THE APPLICATION BY PRESSING PLAY

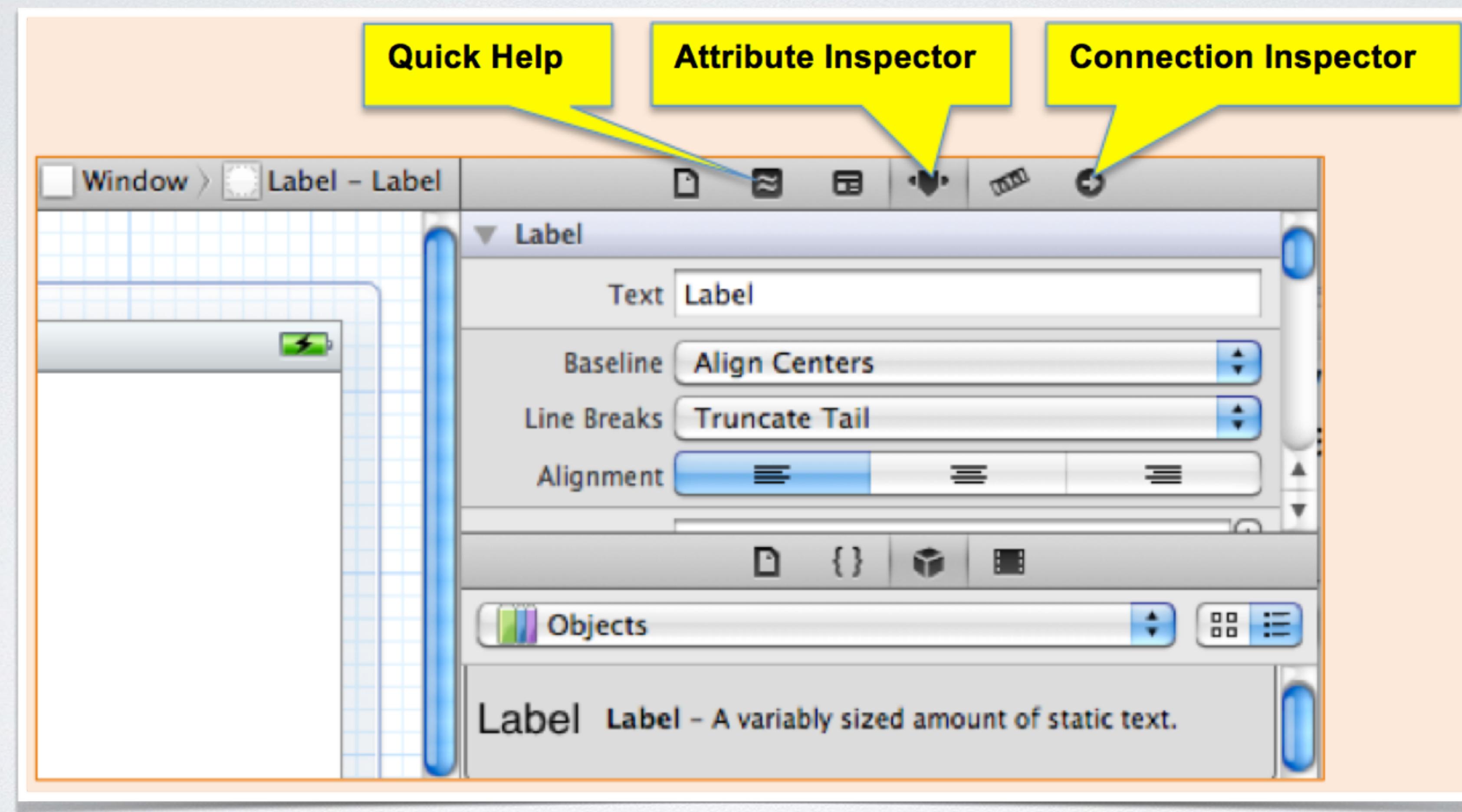


ATTRIBUTE INSPECTOR

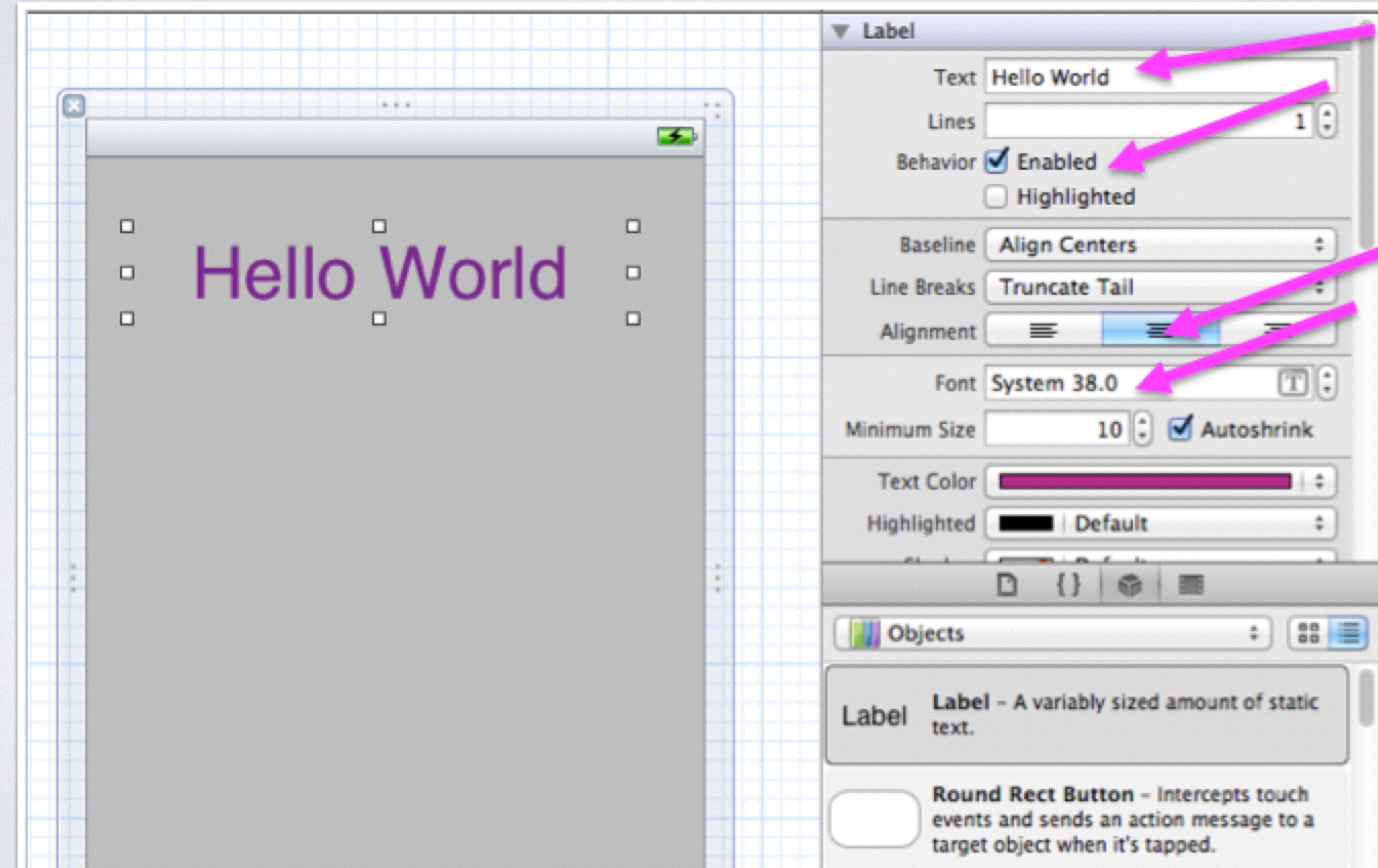


GroundSpeed™
rapid web + mobile software

INSPECTORS



CREATE “HELLO WORLD!” LABEL



Hello World! Demo



GroundSpeed™
rapid web + mobile software

Q. HOW DOES THE SYSTEM KNOW THAT IT NEEDS TO RUN THE VIEW VIA THE MAIN WINDOW?

The screenshot shows the Xcode interface with the main.m file open in the editor. A pink arrow points from the left margin to the AppDelegate.h import statement. Another pink arrow points from the right margin to the UIApplicationMain call, highlighting the class parameter.

```
// main.m
// HelloWorld
//
// Created by Don Miller on 1/26/14.
// Copyright (c) 2014 University of Toledo. All rights reserved.

#import <UIKit/UIKit.h>

#import "AppDelegate.h"

int main(int argc, char * argv[])
{
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil,
                               NSStringFromClass([AppDelegate class]));
    }
}
```



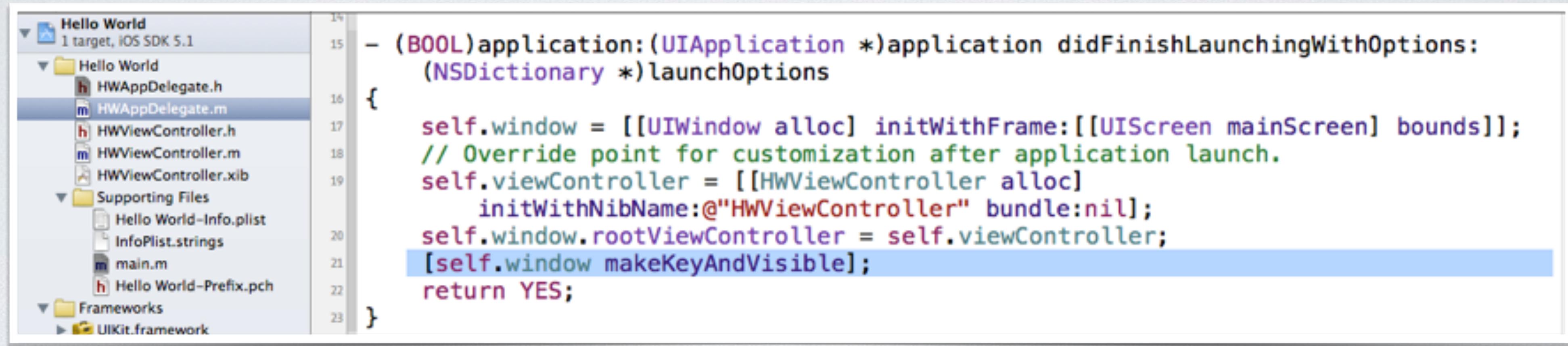
APPDELEGATE.H



```
1 // HWAppDelegate.h
2 // Hello World
3 //
4
5 #import <UIKit/UIKit.h>
6
7 @class HWViewController;
8
9 @interface HWAppDelegate : UIResponder <UIApplicationDelegate>
10
11 @property (strong, nonatomic) UIWindow *window;
12 @property (strong, nonatomic) HWViewController *viewController;
13
14 @end
```



APPDELEGATE.M



The screenshot shows the Xcode interface with the 'Hello World' project selected. The left sidebar displays the project structure:

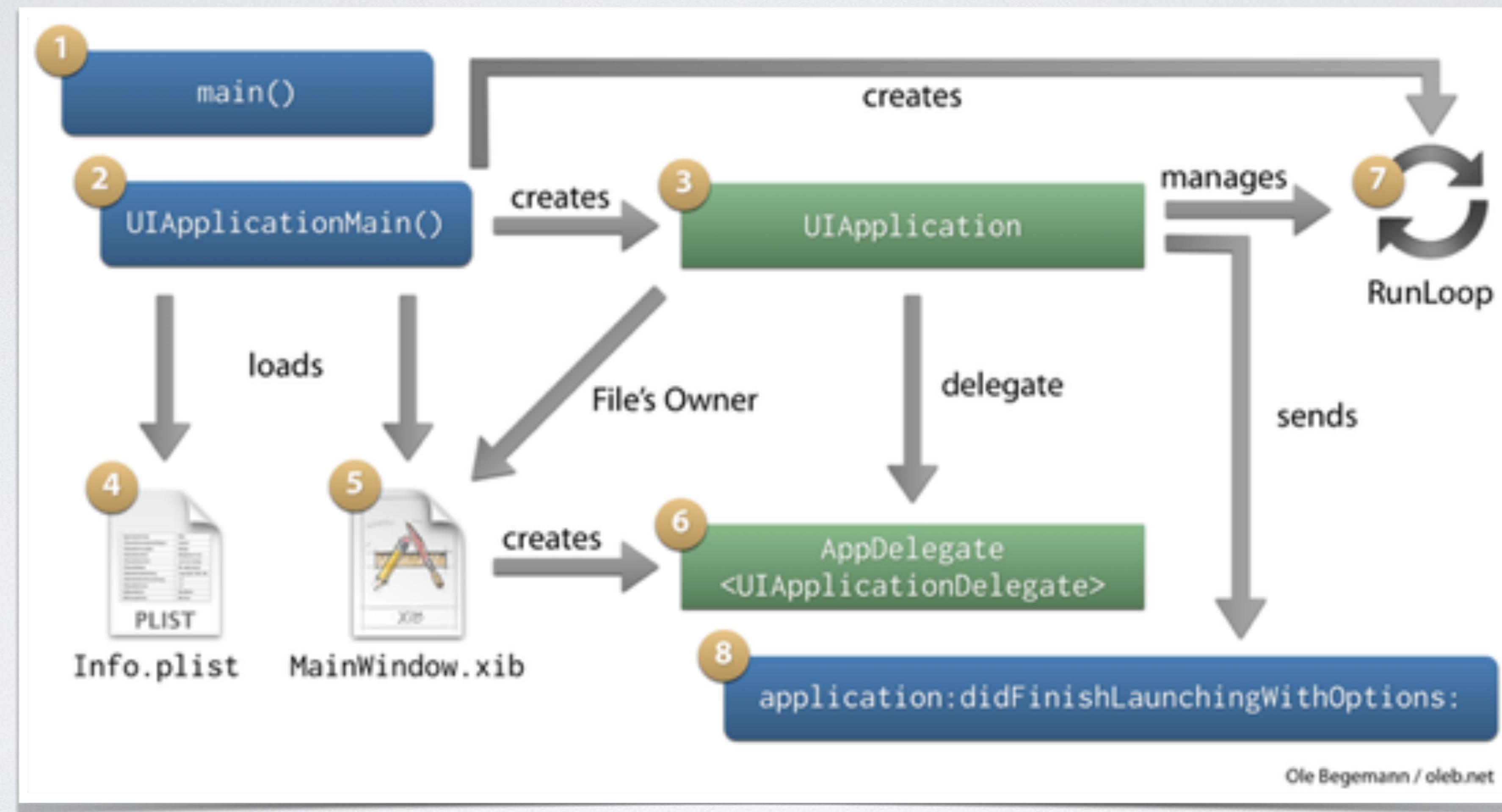
- >Hello World (target, iOS SDK 5.1)
- |- Hello World
 - HWAppDelegate.h
 - HWAppDelegate.m** (selected)
 - HWViewController.h
 - HWViewController.m
 - HWViewController.xib
- |- Supporting Files
 - Hello World-Info.plist
 - InfoPlist.strings
 - main.m
 - Hello World-Prefix.pch
- |- Frameworks
 - UIKit.framework

The right pane shows the code for **HWAppDelegate.m**:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch.
    self.viewController = [[HWViewController alloc]
        initWithNibName:@"HWViewController" bundle:nil];
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
```



IOS APPLICATION RUNTIME CYCLE



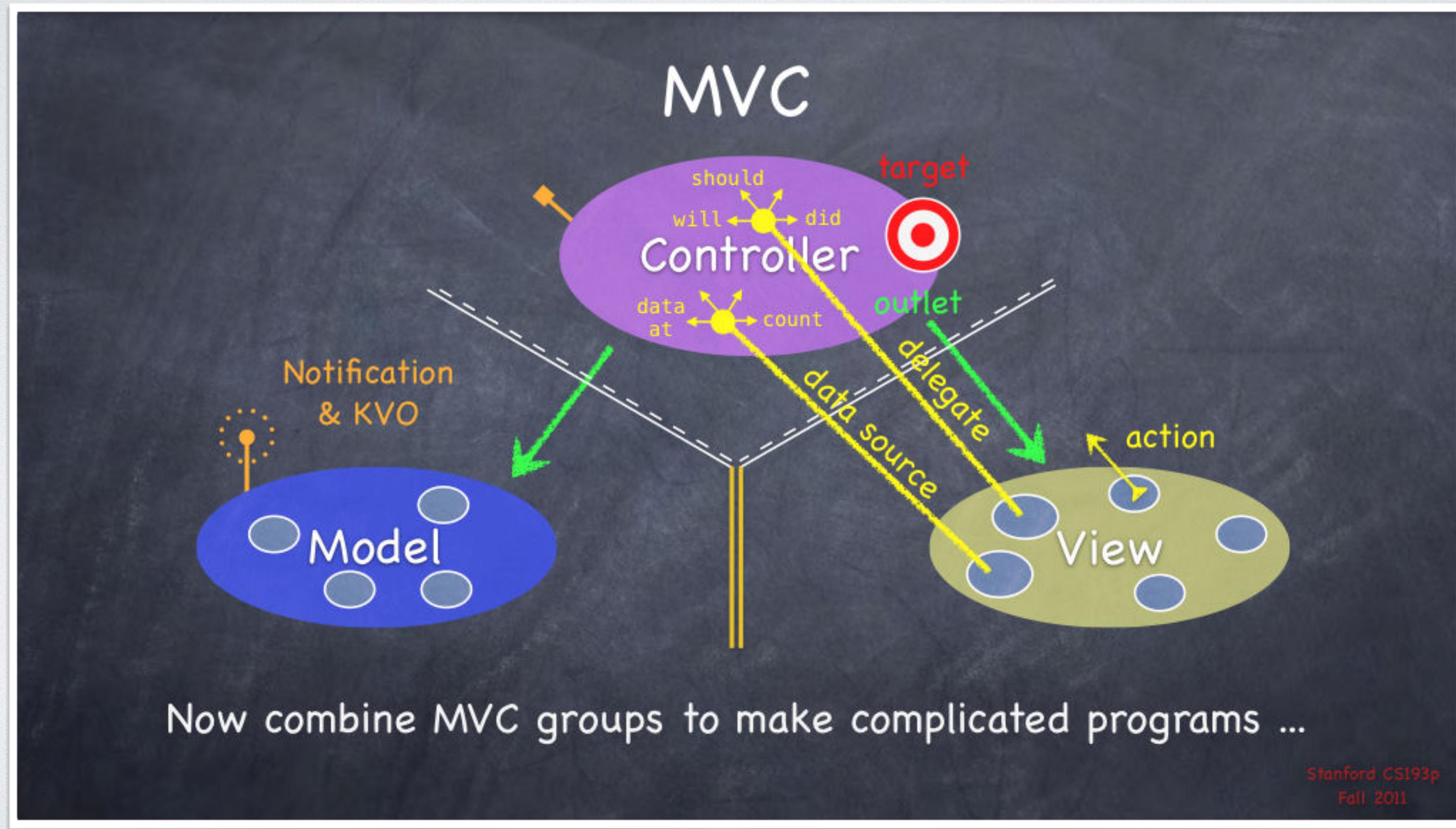
Ole Begemann / oleb.net



GroundSpeed™
rapid web + mobile software

MODEL - VIEW - CONTROLLER

Stanford iTunes University 2011

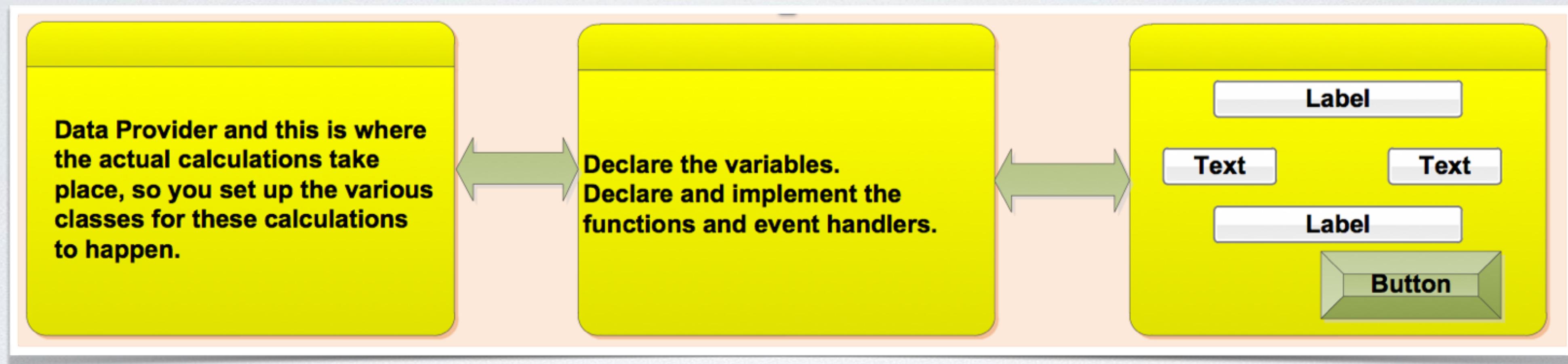


MODEL - VIEW - CONTROLLER

- **The Model:** The M in MVC stands for the Model: For the Model, we will need to develop an independent class (or a set of classes) that will serve the data requirements and standard business procedures.
- **The View:** In this context the View is the UI screen's layout (the canvas). Here, we configure many User Interface items (also known as UI controls on the screen).
- **The Controller:** This is a class that connects the view with the model. In its simplest form, it contains the name of the UIControls (that would be used in the app), and it also includes the signatures and the detail implementations of event handlers.



MODEL - VIEW - CONTROLLER

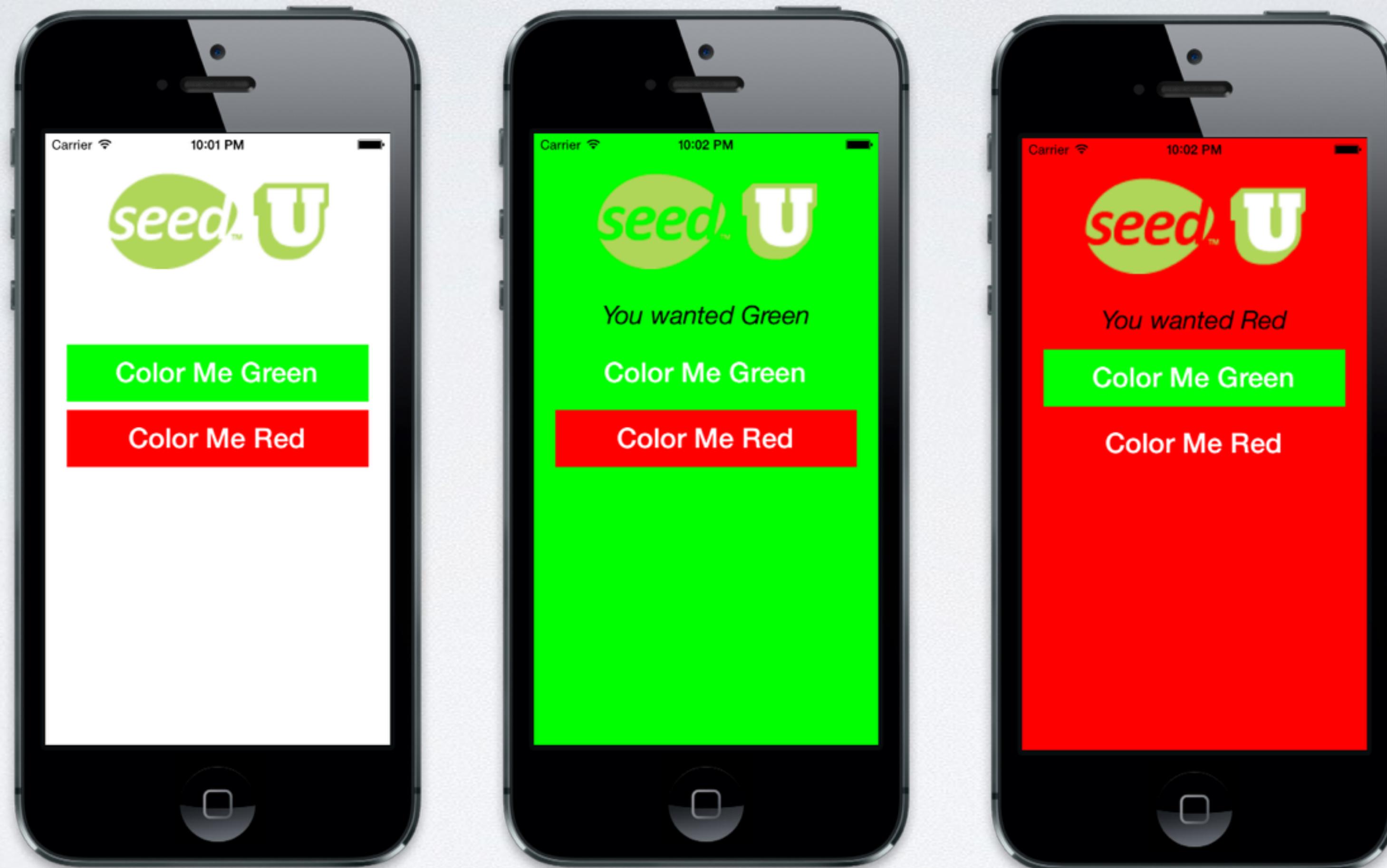


Break Time



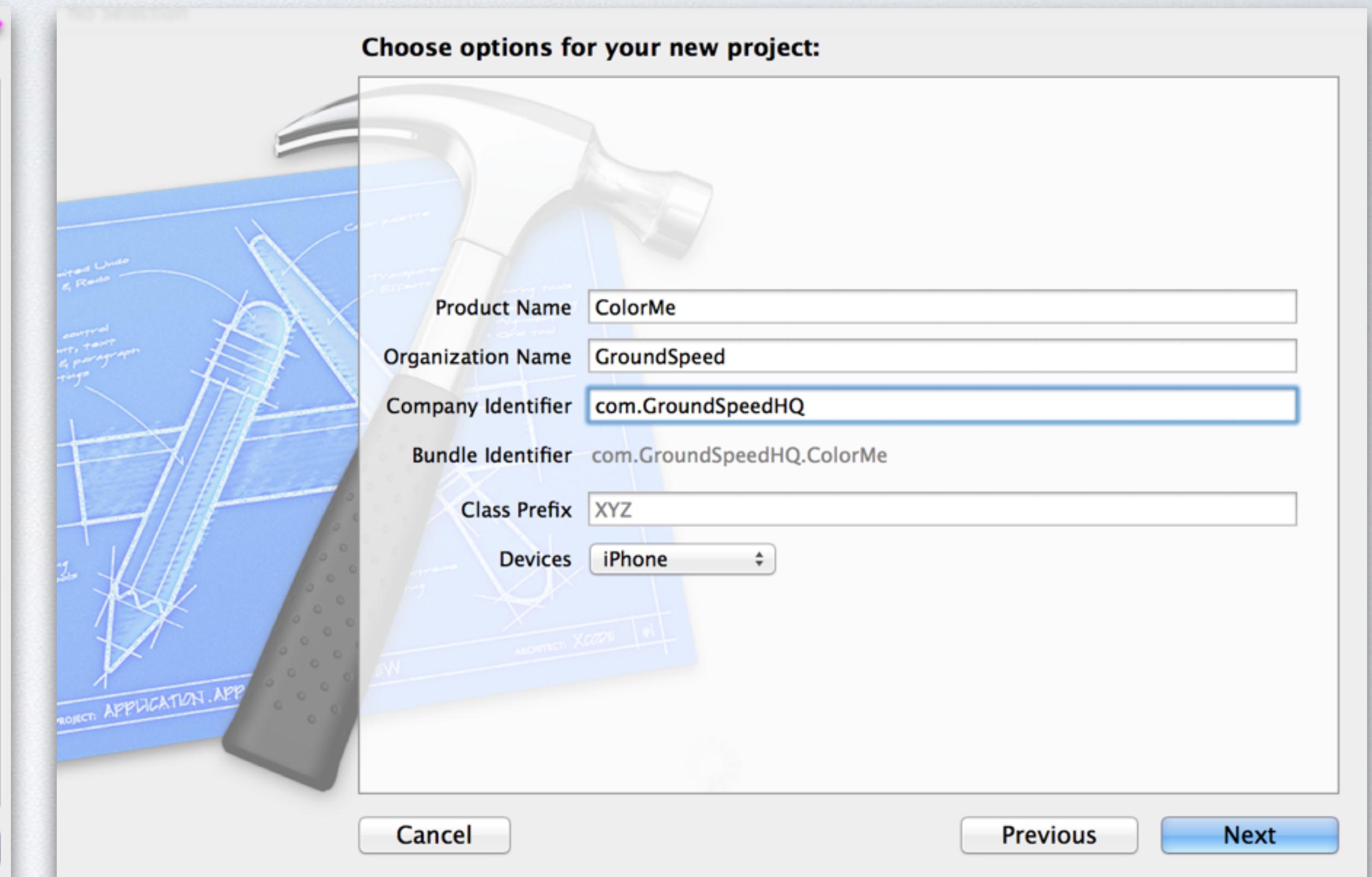
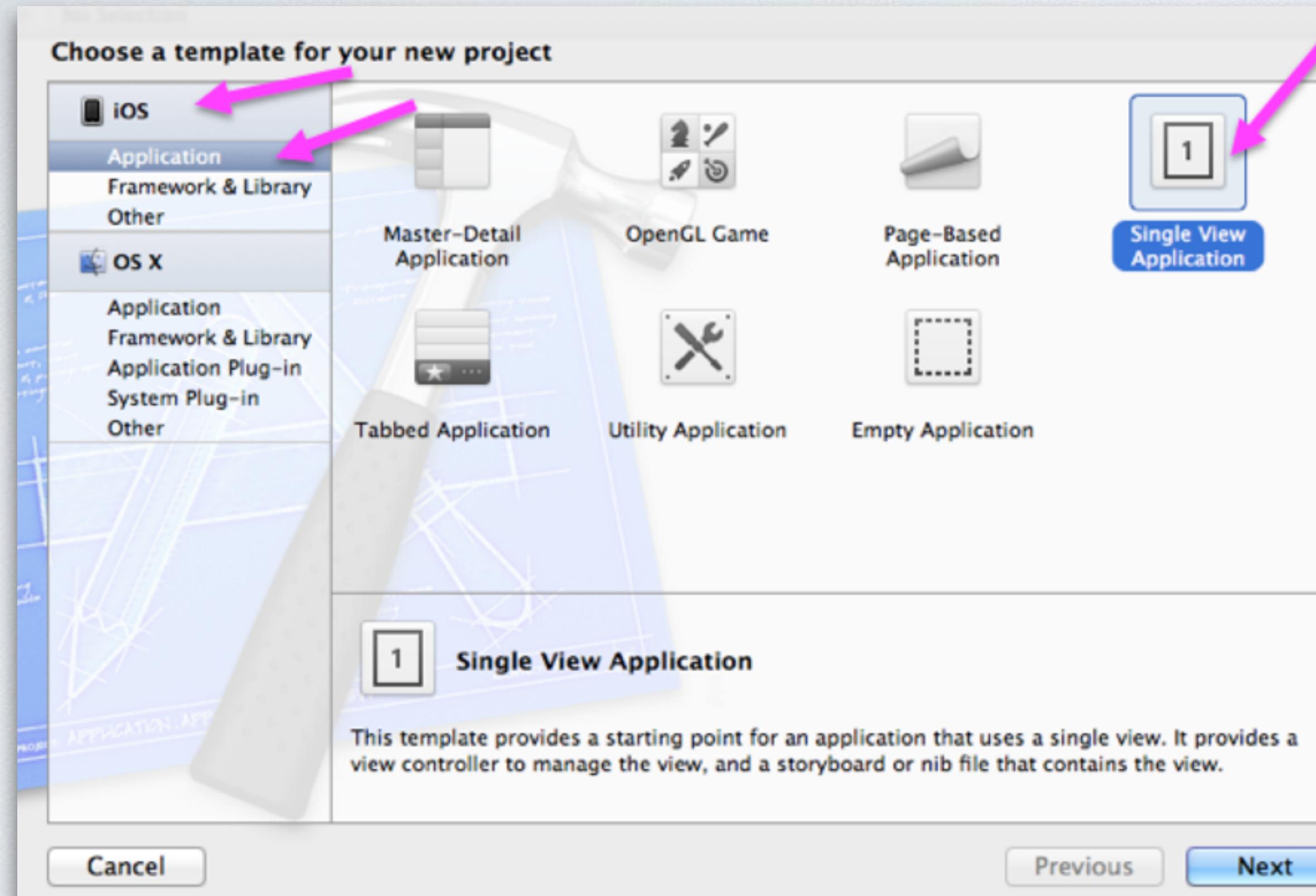
GroundSpeed™
rapid web + mobile software

ANOTHER QUICK DEMO - [COLOR ME]

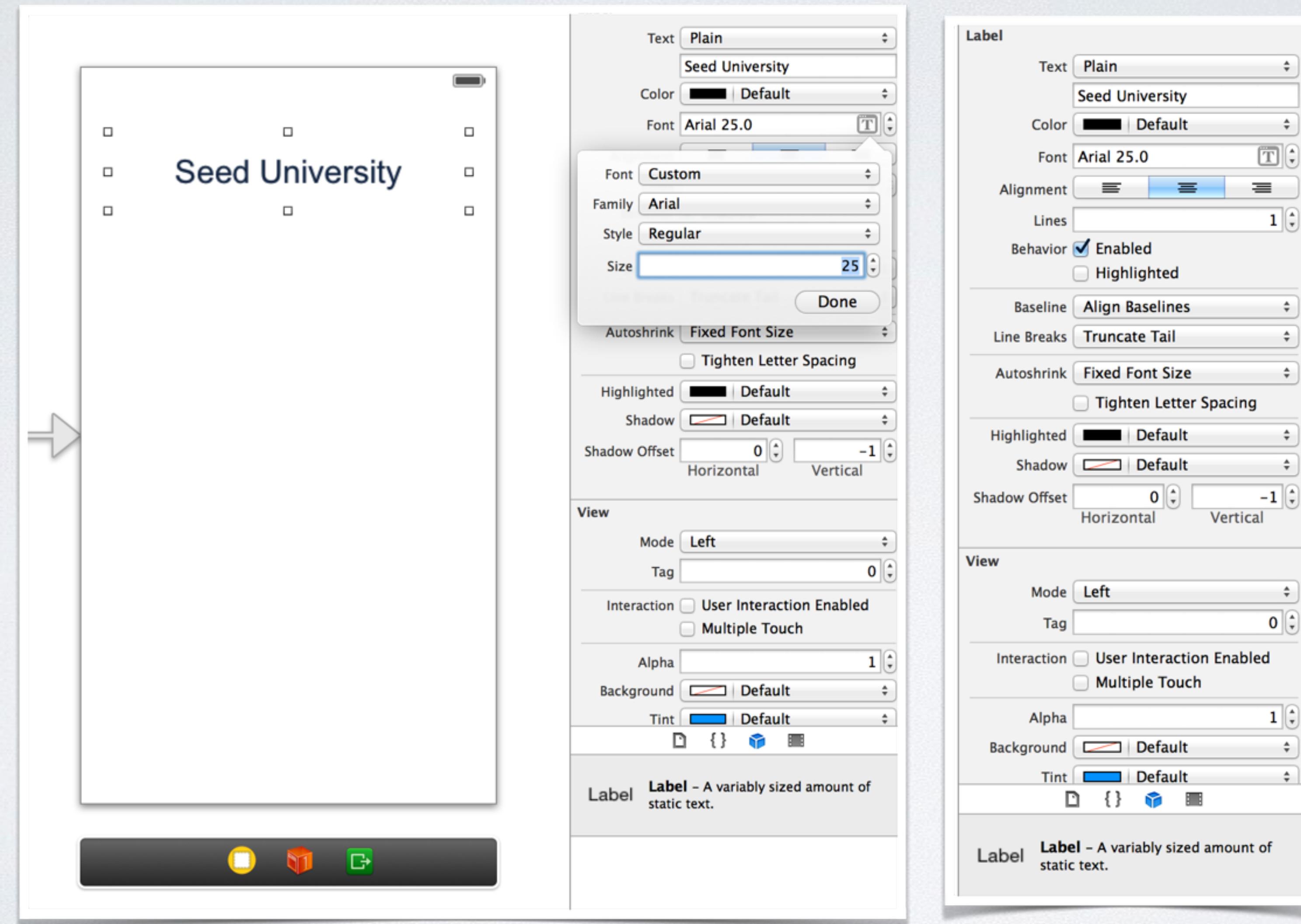


GroundSpeed™
rapid web + mobile software

CREATE A SINGLEVIEW APPLICATION



DRAG A LABEL TO THE STORYBOARD



CREATE A BLANK LABEL FOR OUTPUT

The screenshot shows a storyboard scene titled "Seed University". Inside the scene, there is a single "Label" object. The label has the text "Label" displayed. The "Label" inspector is open to the right, showing the following settings:

- Text:** Plain | Label
- Color:** Black Color
- Font:** System Italic 24.0
- Alignment:** Center
- Lines:** 1
- Behavior:** Enabled (checkbox checked), Highlighted (checkbox unselected)
- Baseline:** Align Baselines
- Line Breaks:** Truncate Tail
- Autoshrink:** Fixed Font Size (checkbox unselected), Tighten Letter Spacing (checkbox unselected)
- Highlighted:** Default
- Shadow:** Default
- Shadow Offset:** 0 (Horizontal), -1 (Vertical)
- View:** Mode: Left, Tag: 0
- Interaction:** User Interaction Enabled (checkbox unselected), Multiple Touch (checkbox unselected)
- Alpha:** 1
- Background:** Default
- Tint:** Default

Label **Label** - A variably sized amount of static text.



CREATE TWO BUTTONS

The image shows a mobile application interface builder. On the left, a screen titled "Seed University" is displayed with a label "Label" and two buttons: a green button labeled "Color Me Green" and a red button labeled "Color Me Red". On the right, a detailed configuration panel for the green button is shown:

Button

- Type: System
- State Config: Default
- Title: Plain
- Color Me Green
- Font: System Bold 26.0
- Text Color: (Color swatch)
- Shadow Color: (Color swatch) | Default
- Image: Default Image
- Background: Default Background Image
- Shadow Offset: 0.0 (Width), 0.0 (Height)
 - Reverses On Highlight
 - Shows Touch On Highlight
 - Highlighted Adjusts Image
 - Disabled Adjusts Image
- Line Break: Truncate Middle
- Edge: Content
- Inset:
 - Top: 0
 - Bottom: 0
 - Left: 0
 - Right: 0

Control

- Alignment:
 - Horizontal
 - Vertical
- { } { } { }

Button – Intercepts touch events and sends an action message to a target object when it's tapped.



UPDATE THE VIEWCONTROLLER.H FILE



The screenshot shows the Xcode interface with the project navigation bar at the top. Below it is a tree view of the project structure:

- ColorMe (target, iOS SDK 7.1)
- ColorMe (group)
 - AppDelegate.h
 - AppDelegate.m
 - Main.storyboard
 - ViewController.h (selected)
 - ViewController.m
 - Images.xcassets
- Supporting Files
 - logo.png
 - ColorMe-Info.plist
 - InfoPlist.strings
 - main.m
 - ColorMe-Prefix.pch
- ColorMeTests
- Frameworks
- Products

To the right of the tree view is the code editor window. The current file is `ViewController.h`. The code is as follows:

```
// ViewController.h
// ColorMe
//
// Created by Don Miller on 1/15/14.
// Copyright (c) 2014 University of Toledo. All rights reserved.

brgr-CMViewController.h

@property (nonatomic, retain) IBOutlet UILabel *lblMsg;

// These will contain the code for our event handlers
- (IBAction)displayGreen;
- (IBAction)displayRed;

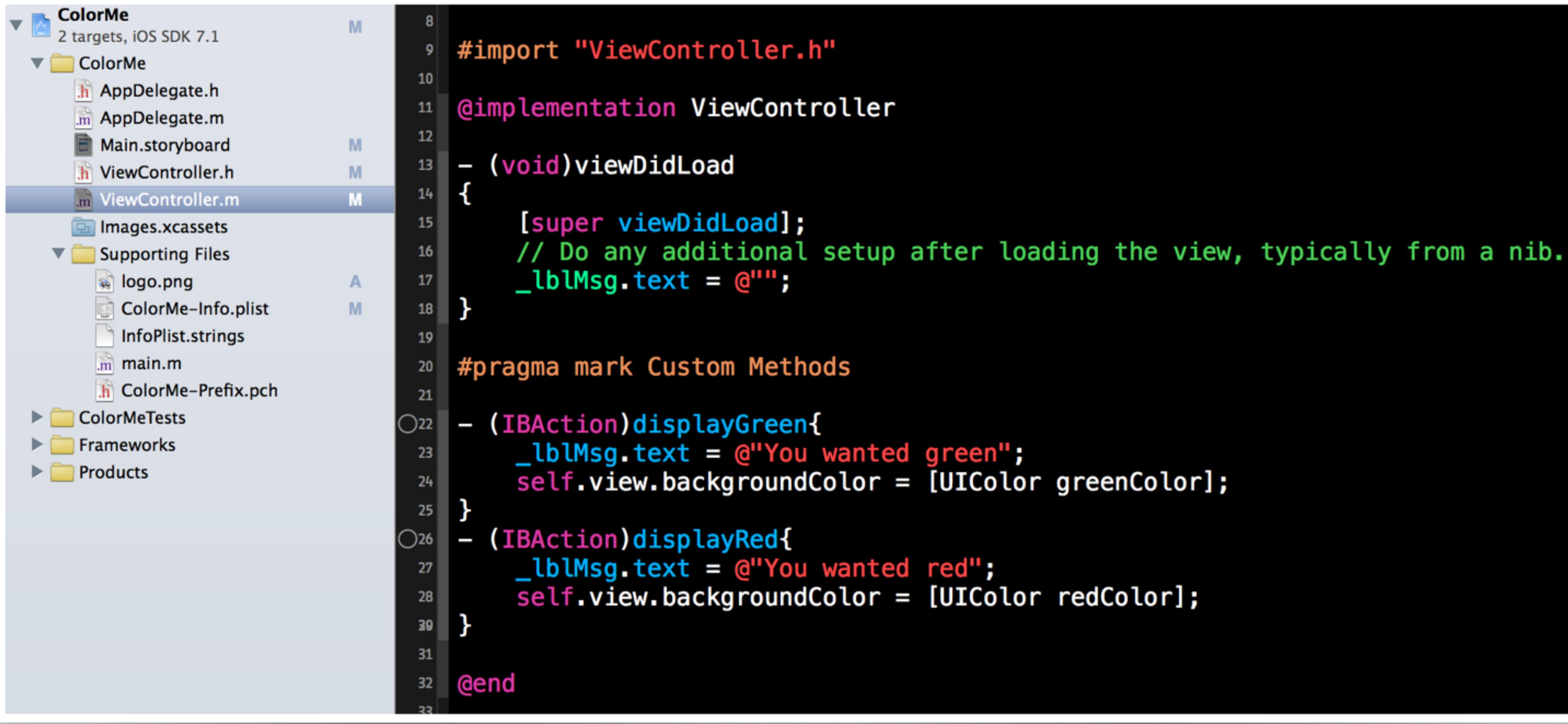
@end
```

[Link to Gist](#)



GroundSpeed™
rapid web + mobile software

UPDATE THE VIEWCONTROLLER.M FILE



The screenshot shows the Xcode interface with the project navigation bar at the top. Below it is a tree view of the project structure:

- ColorMe (target)
 - ColorMe
 - AppDelegate.h
 - AppDelegate.m
 - Main.storyboard
 - ViewController.h
 - ViewController.m (selected)
 - Images.xcassets
 - Supporting Files
 - logo.png
 - ColorMe-Info.plist
 - InfoPlist.strings
 - main.m
 - ColorMe-Prefix.pch
 - ColorMeTests
 - Frameworks
 - Products

The ViewController.m file is open in the main editor window. The code is as follows:

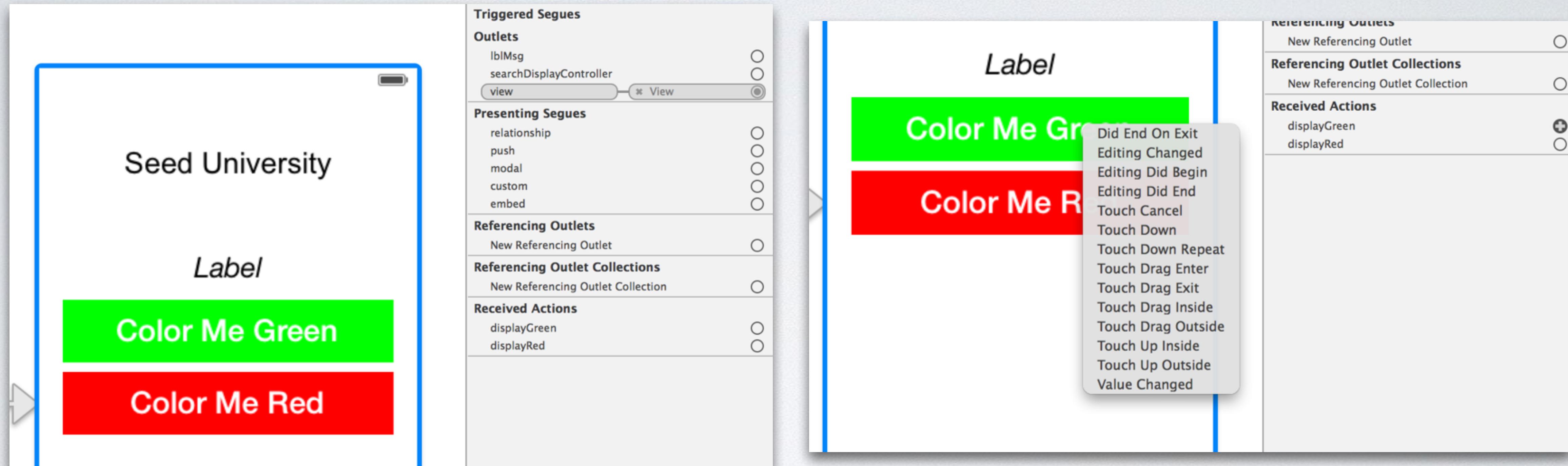
```
8 #import "ViewController.h"
9
10 @implementation ViewController
11
12 - (void)viewDidLoad
13 {
14     [super viewDidLoad];
15     // Do any additional setup after loading the view, typically from a nib.
16     _lblMsg.text = @"";
17 }
18
19 #pragma mark Custom Methods
20
21 - (IBAction)displayGreen{
22     _lblMsg.text = @"You wanted green";
23     self.view.backgroundColor = [UIColor greenColor];
24 }
25
26 - (IBAction)displayRed{
27     _lblMsg.text = @"You wanted red";
28     self.view.backgroundColor = [UIColor redColor];
29 }
30
31
32 @end
33
```

[Link to Gist](#)



GroundSpeed™
rapid web + mobile software

CONNECTING THE OUTLETS AND ACTIONS

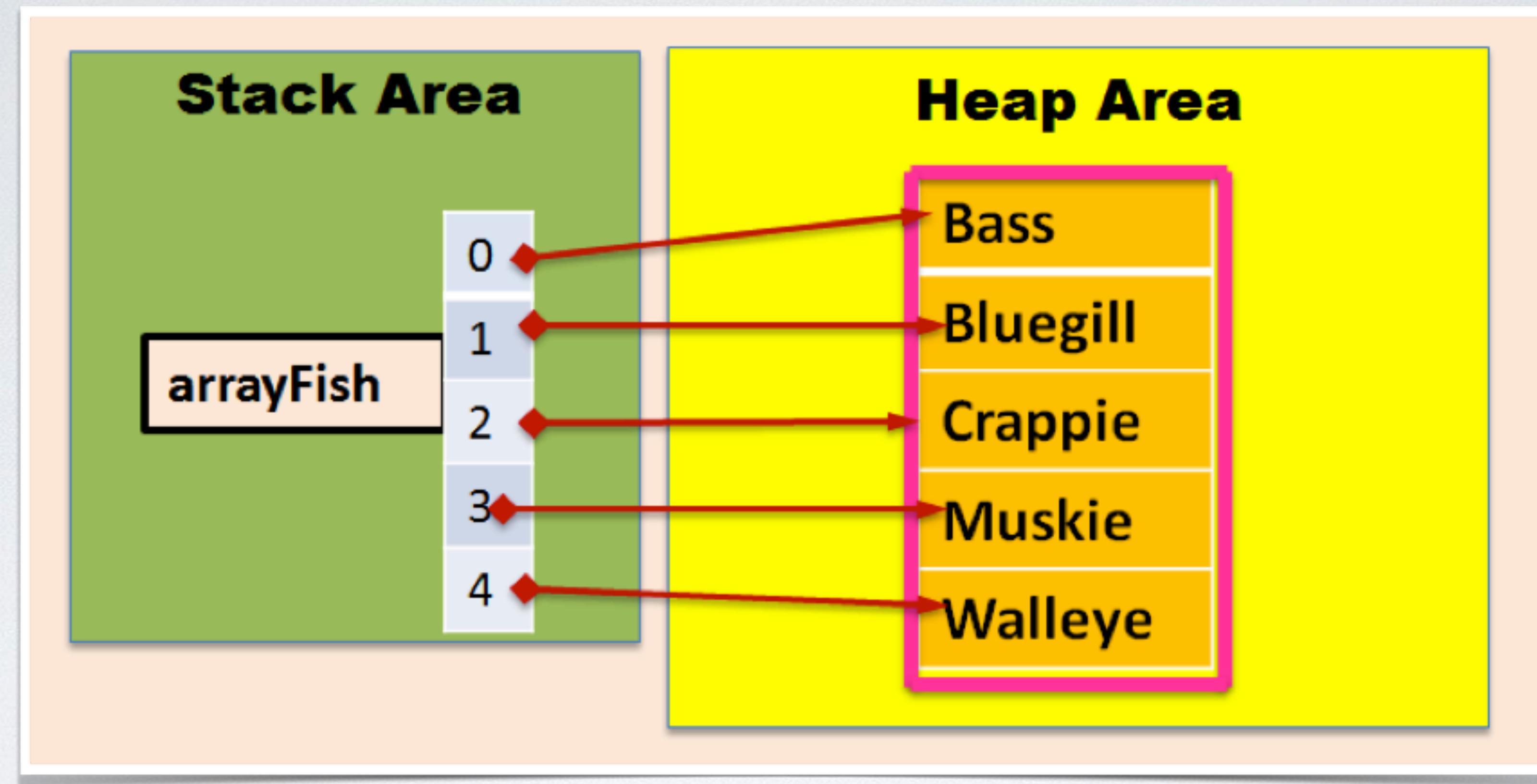


Color Me Demo



GroundSpeed™
rapid web + mobile software

ARRAYS



- `[arrayFish count]` will return the integer 5.
- `[arrayFish objectAtIndex:3]` will return the string Muskie
- `[arrayFish objectAtIndex:5]` will raise an Error

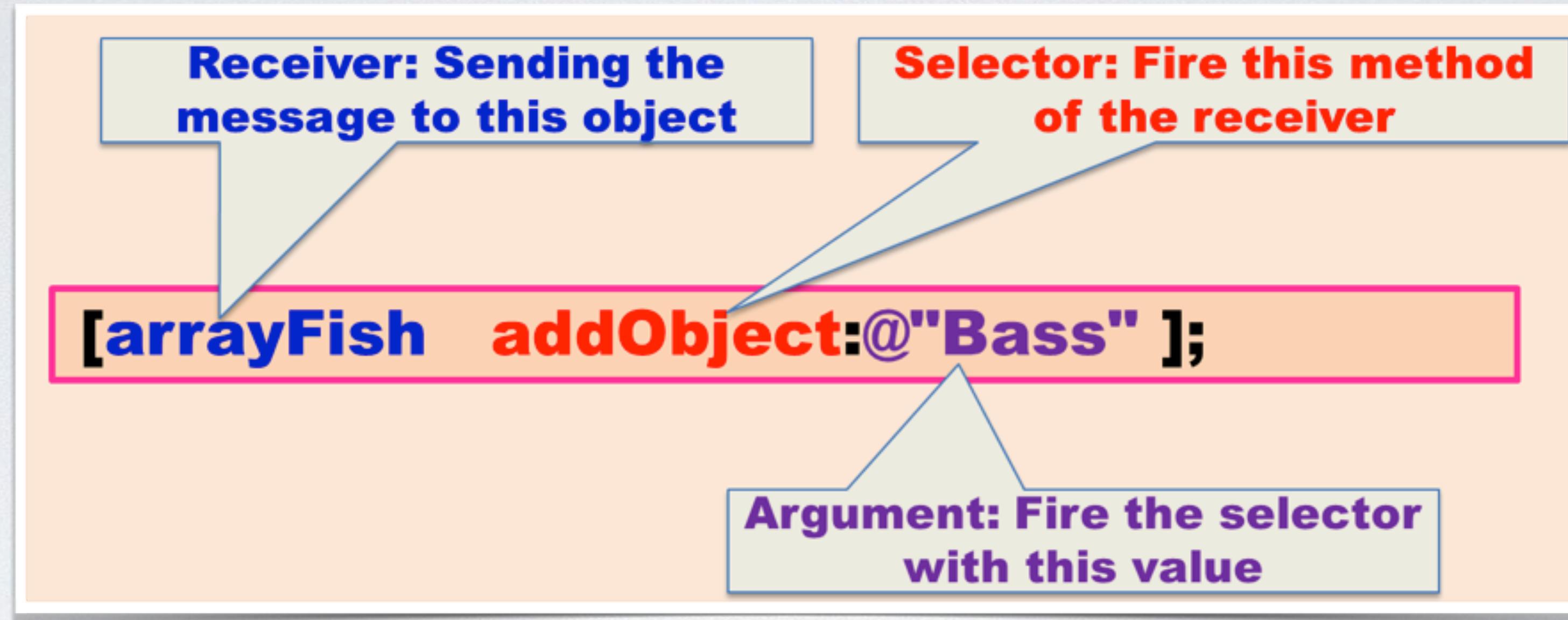


CREATING AN ARRAY WITH OBJECTIVE-C

```
1 // allocate and initialize an NSMutableArray object
2 NSMutableArray *arrayFish = [[NSMutableArray alloc] init];
3
4 [arrayFish addObject:@"Bass" ];
5 [arrayFish addObject:@"Bluegill" ];
6 [arrayFish addObject:@"Crappie" ];
7 [arrayFish addObject:@"Muskie" ];
8 [arrayFish addObject:@"Walleye" ];
```



SENDING MESSAGES



- In this context, let us get introduced to method firing traits in Objective-C. The method firing phenomena is also known as Sending Messages.
 - In Objective -C, usually a method is called in a pair of square brackets.
 - A message has three parts:
 - receiver: to whom are you sending the message
 - selector: which method of the receiver would you want to fire
 - arguments: the values required by the selector, if any



ABOUT MUTABLE AND IMMUTABLE OBJECTS

**Immutable
Object**

Once it is instantiated, we cannot add,
edit or remove its contents any further

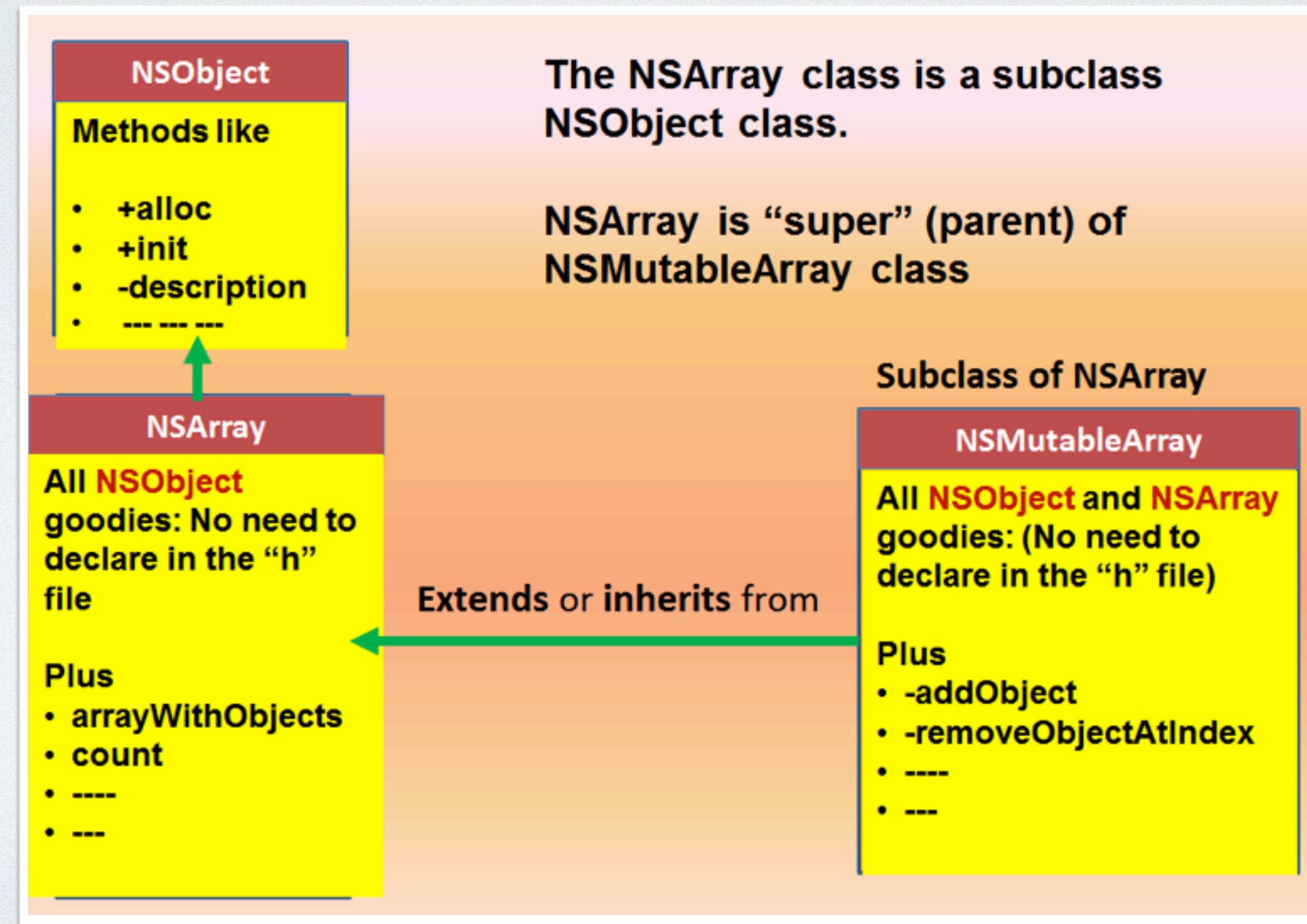
**Mutable
Object**

Once it is instantiated, we can add,
edit or remove its contents later

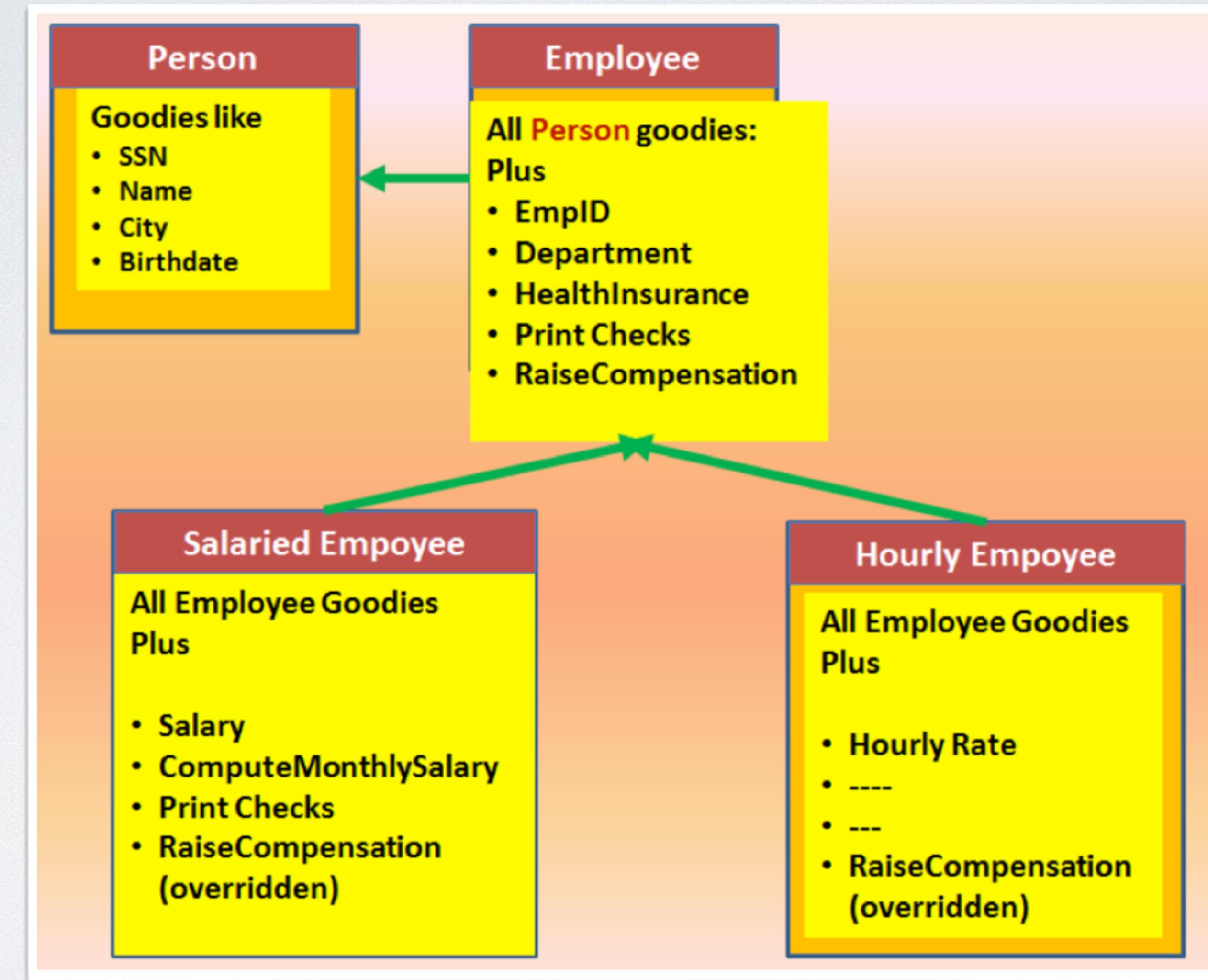
- **NSArray** is **immutable**, and its **mutable subclass** is **NSMutableArray**.
- **Immutable objects take less memory and execute faster than the mutable versions.**
- **All collection classes contain objects. Thus you cannot create an array of int, double or floats (the primitive data types). In that case you will need to populate an array with instances of NSNumber class.**
- **Powerful Feature:** An Objective C collection object can accommodate any mix of objects as its elements. For example an *arrayPeople* may contain some faculty objects, some textbook objects, and/or some animal objects.
- **By the way all strings in Objective-C are actually objects of NSString class (this is the immutable version). There is also a mutable String class named NSMutableString.**



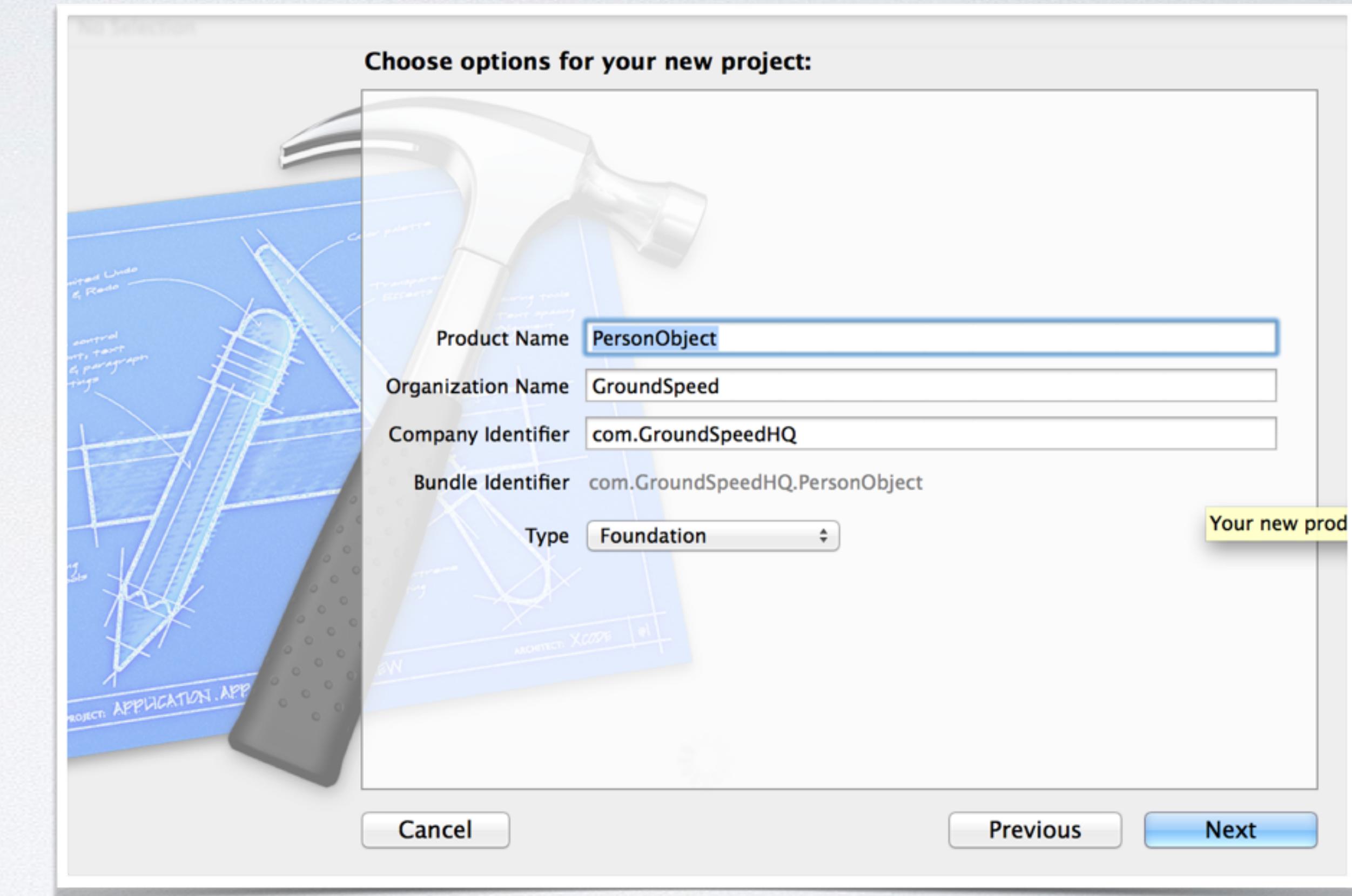
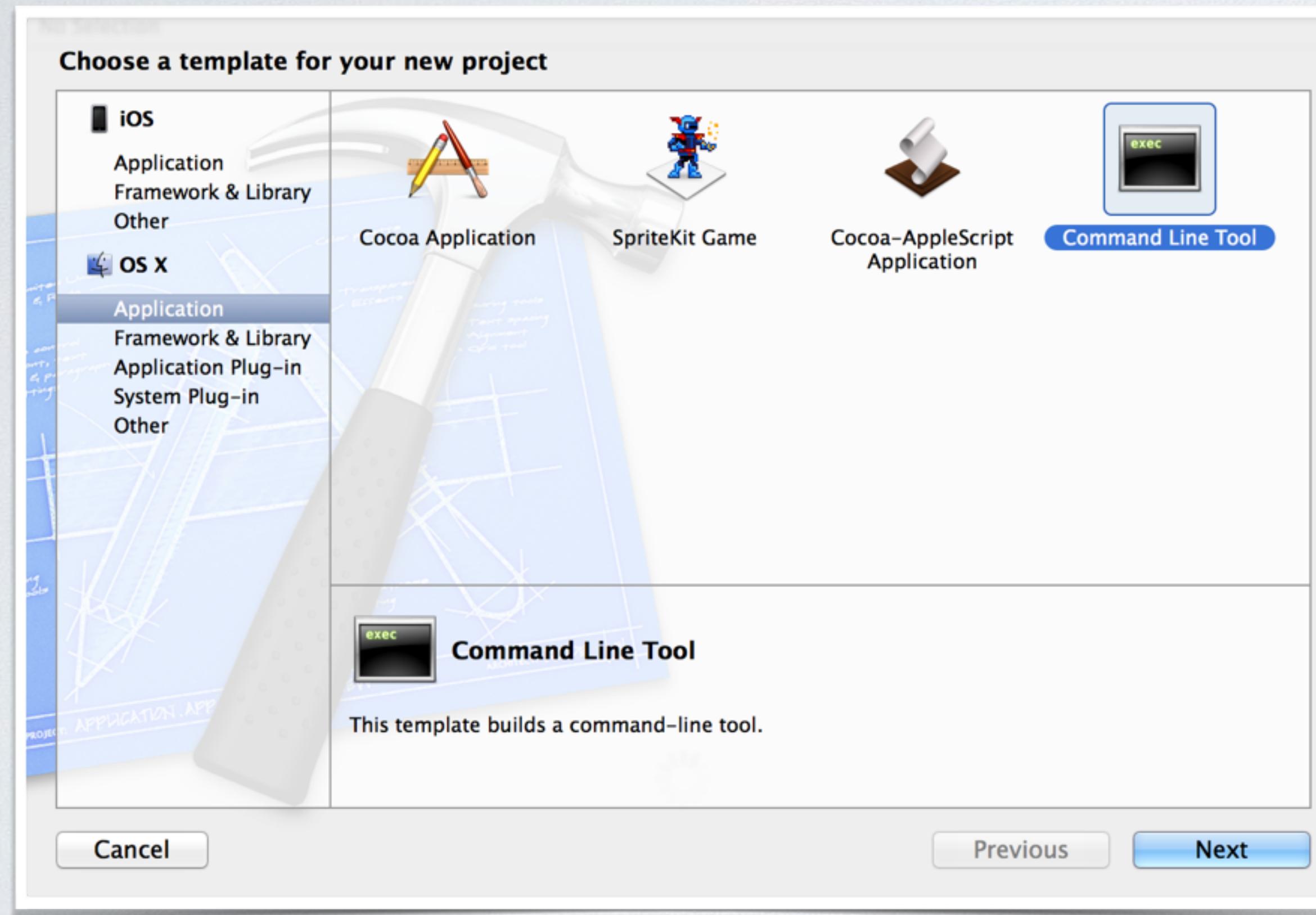
SUBCLASSES AND NSMUTABLEARRAY GENEALOGY



AN EXAMPLE



EXAMPLE: CREATE A COMMAND LINE PROGRAM



CLASS ILLUSTRATIONS

```
9 #import <Foundation/Foundation.h>
10
11 @interface Person : NSObject
12
13 @property (nonatomic, strong) NSString *ssn;
14 @property (nonatomic, strong) NSString *name;
15 @property (nonatomic, strong) NSString *city;
16 @property (nonatomic, strong) NSString *state;
17
18 @end
19
```

```
9 #import <Foundation/Foundation.h>
10 #import "Employee.h"
11
12 @interface SalariedEmployee : Employee
13
14 @property (nonatomic, strong) NSString *salary;
15
16 -(void)computeMonthlySalary;
17 -(void)printChecks;
18 -(void)raiseCompetition;
19
20 @end
21
```

```
9 #import <Foundation/Foundation.h>
10 #import "Person.h"
11
12 @interface Employee : Person
13
14 @property (nonatomic, strong) NSString *empId;
15 @property (nonatomic, strong) NSString *department;
16 @property (nonatomic, strong) NSString *healthInsurance;
17
18 -(void)printChecks;
19 -(void)raiseCompetition;
20
21 @end
22
```

```
9 #import <Foundation/Foundation.h>
10 #import "Employee.h"
11
12 @interface HourlyEmployee : Employee
13
14 @property (nonatomic, strong) NSString *hourlyRate;
15
16 -(void)raiseCompetition;
17
18 @end
19
```



CLASS CONSUMPTION

```
9 #import <Foundation/Foundation.h>
10 #import "HourlyEmployee.h"
11
12 int main(int argc, const char * argv[])
13 {
14
15     @autoreleasepool {
16
17         // insert code here...
18         HourlyEmployee *hourlyEmployee = [[HourlyEmployee alloc] init];
19
20         hourlyEmployee.name = @"Don Miller";
21         hourlyEmployee.empId = @"1234";
22         hourlyEmployee.salary = @"9,999"; ! Property 'salary' not found on object of type 'HourlyEmployee *'
23         hourlyEmployee.hourlyRate = @"9.99";
24         [hourlyEmployee raiseCompetition];
25     }
26     return 0;
27 }
```



STORYBOARD OVERVIEW

- Storyboards are graphic organizers displayed in sequence for the purpose of pre-visualizing a motion picture or animation. The storyboarding process, in the form it is known today, is used by developers to give the designer, developer, and end user an opportunity to see what the application or web site will look like at a high level.
- In iOS, a Storyboard is a container of screens that may have embedded view controllers, navigation controllers, and tab bar controllers. It allows the user to graphically connect screens to one another through embedded objects on the screen. These connections are called segues (pronounced seg-wey). The storyboard eliminates the need for xib files and has one file that contains all of your screens and transitions. It is possible to connect two storyboard files with one another; however, unlikely with smaller applications.



PROS OF STORYBOARDS

Pros of using Storyboards:

- With a storyboard you have a better conceptual overview of all the screens in your app and the connections between them. It's easier to keep track of everything because the entire design is in a single file, rather than spread out over many separate nibs.
- The storyboard describes the transitions between the various screens. These transitions are called “segues” and you create them by simply ctrl-dragging from one view controller to the next. Thanks to segues you need less code to take care of your UI.
- Storyboards make working with table views a lot easier with the new prototype cells and static cells features. You can design your table views almost completely in the storyboard editor, something else that cuts down on the amount of code you have to write.



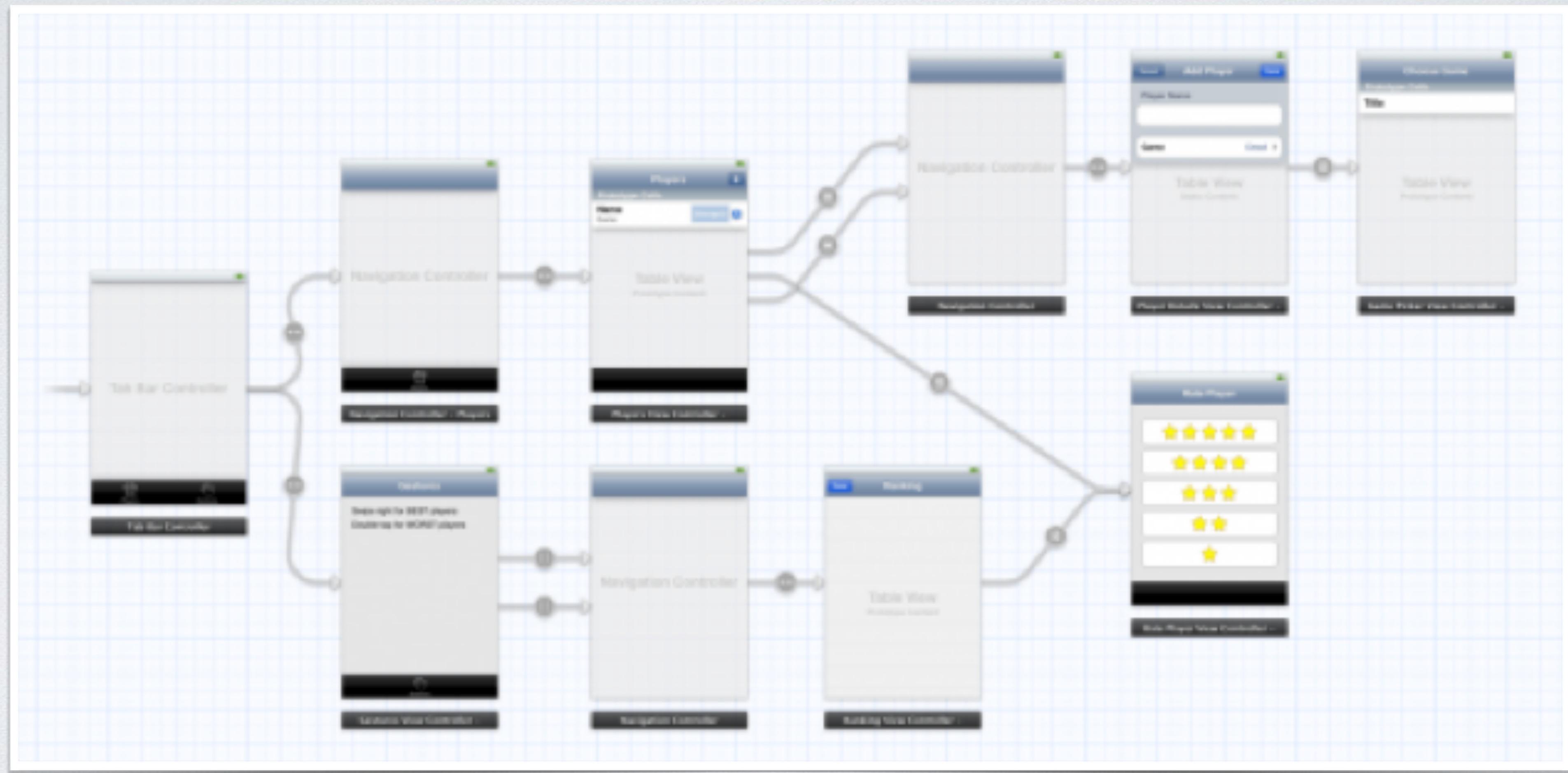
CONS OF STORYBOARDS

Cons of using Storyboards:

- It's not easy to work with storyboards in a team, since only one participant can work on the storyboard at once (because it's one binary file).
- If you need to do things storyboard doesn't offer, it's not quite easy to get storyboard mixed with programmatically created views)
- Since all objects are in one file, a larger project could cause the storyboard to be very slow and unresponsive. It loads the entire thing into memory. It loads the entire item into memory.

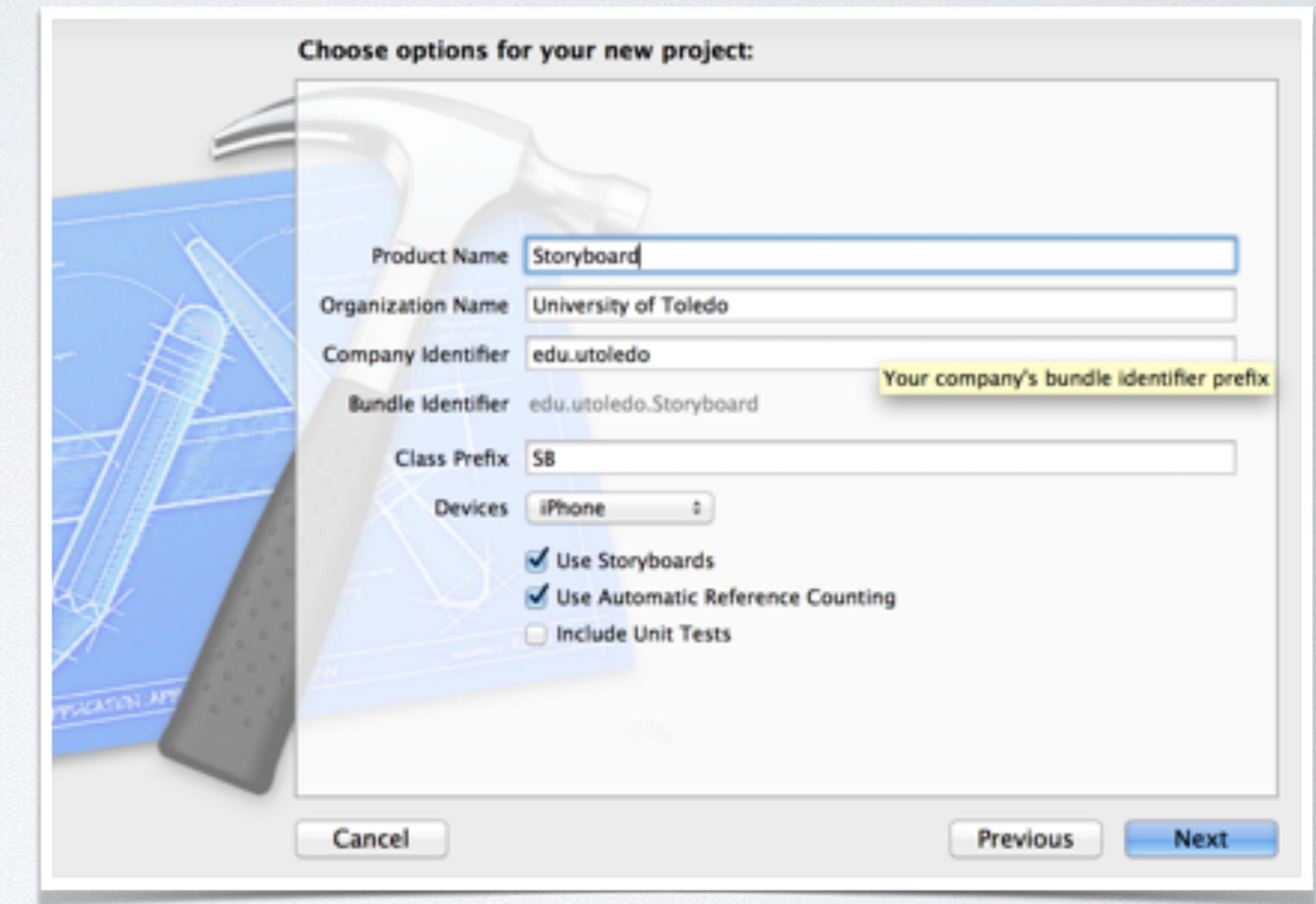
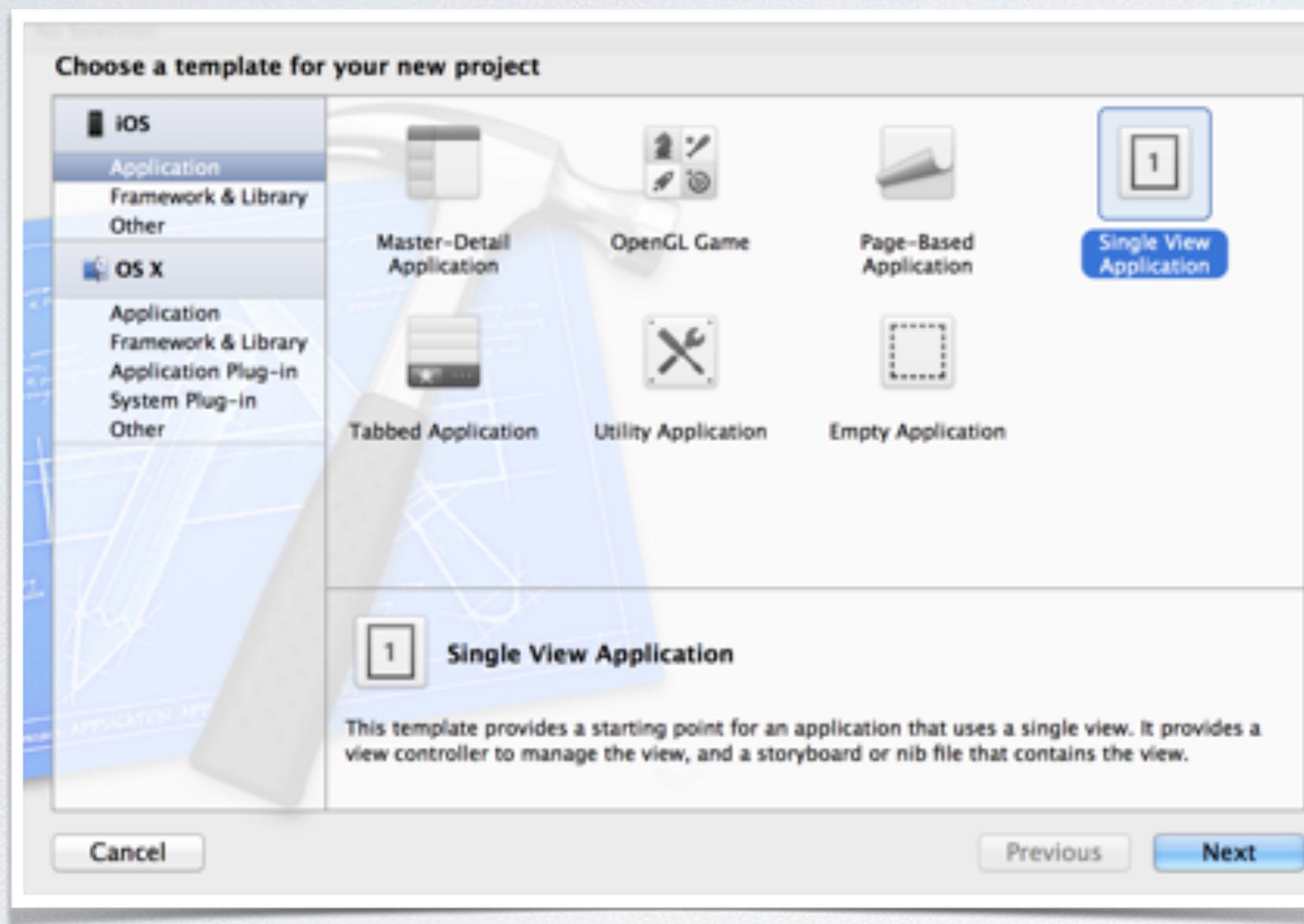


STORYBOARD OVERVIEW



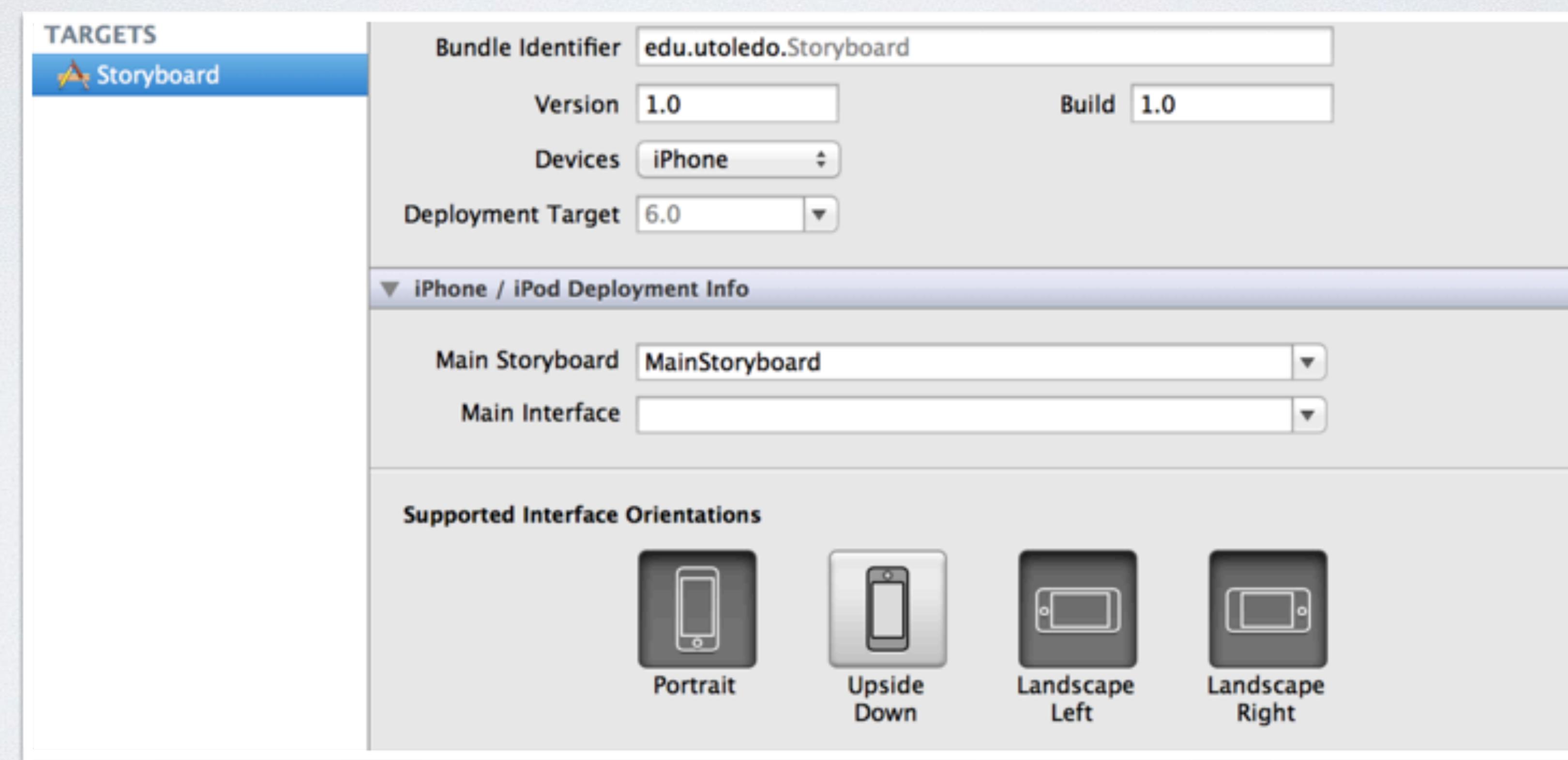
STORYBOARD DEMO

STEP I: CREATE A NEW SINGLE VIEW APPLICATION



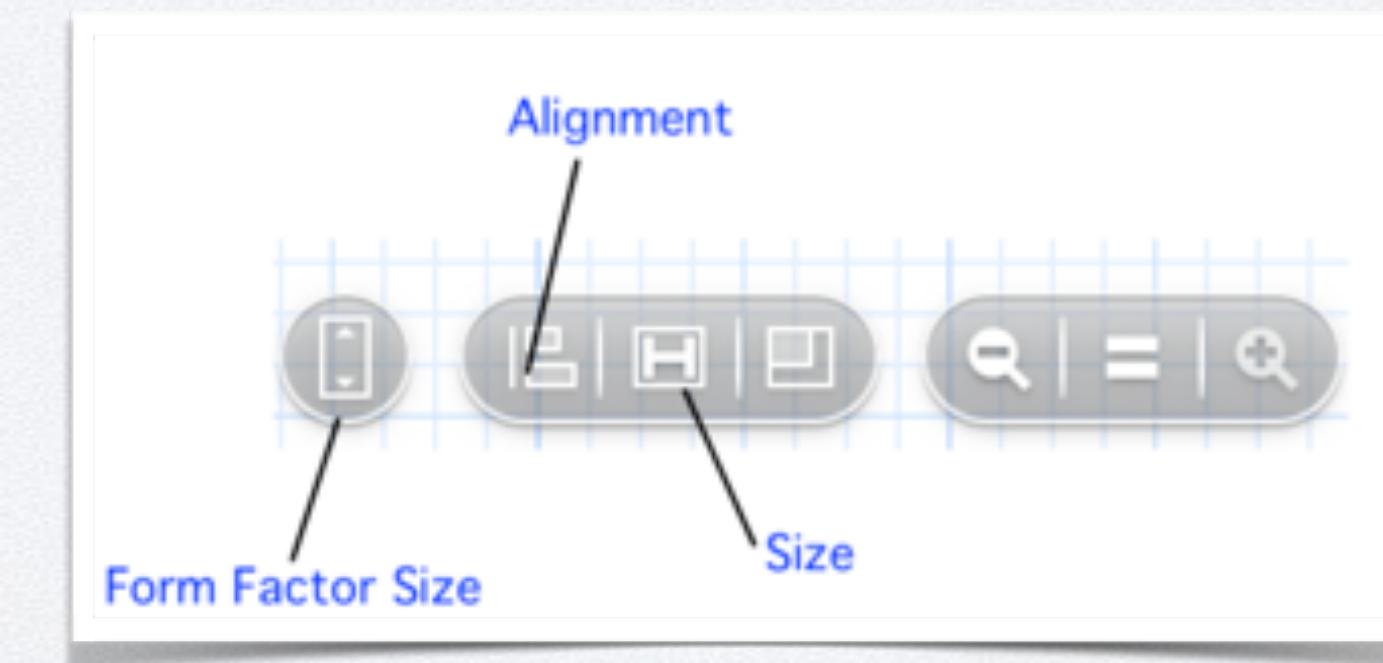
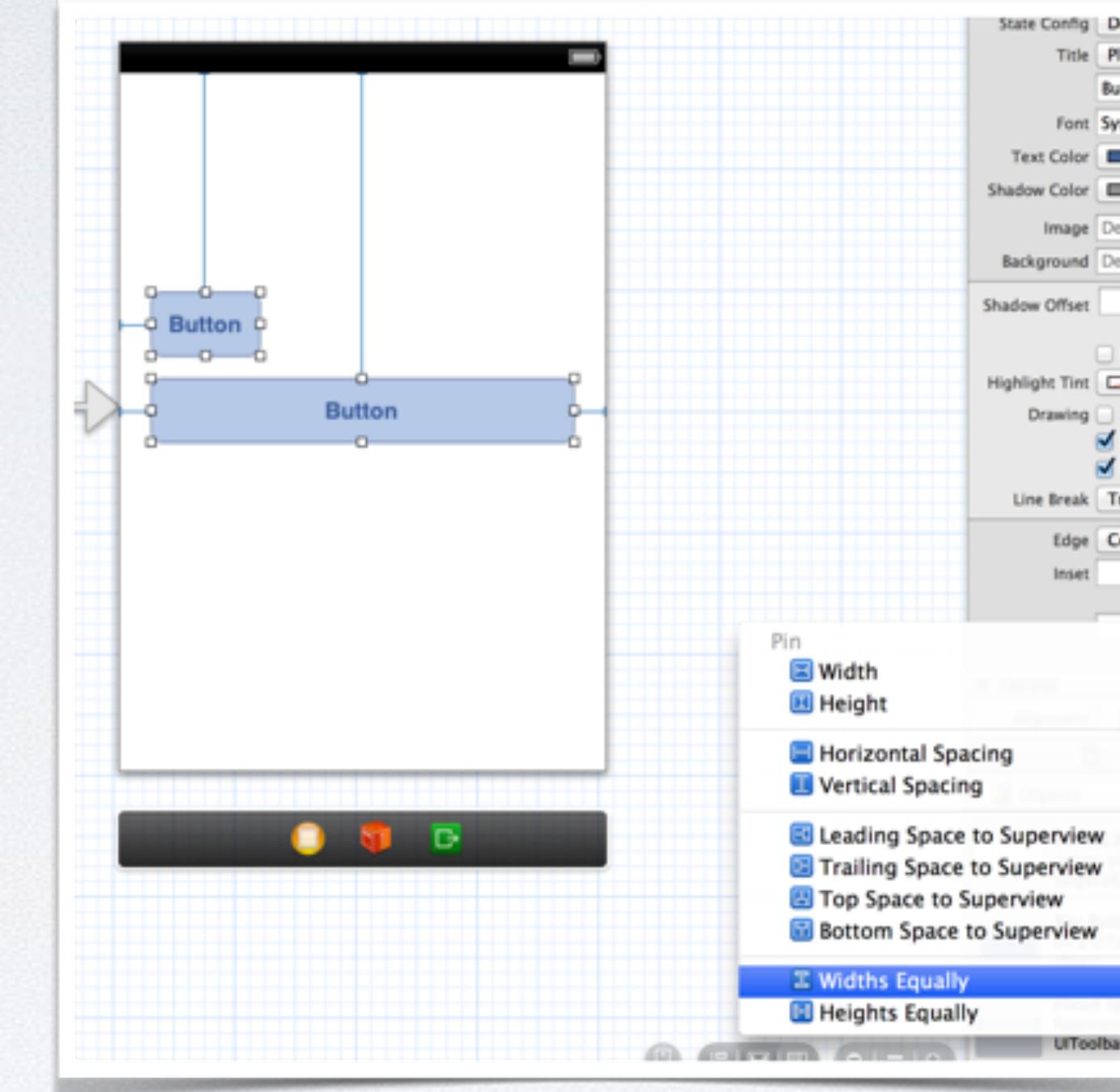
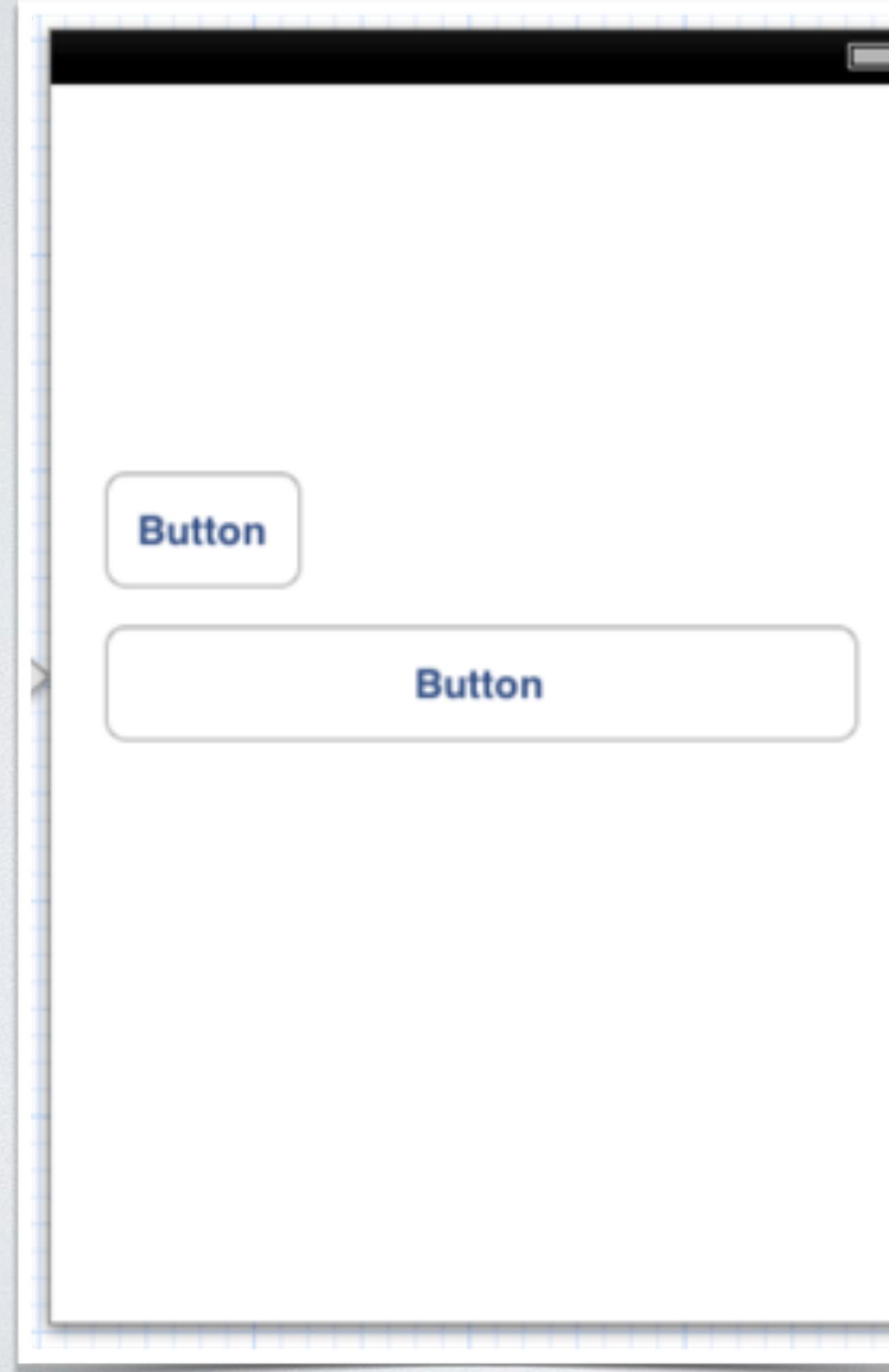
STEP 2: OBSERVE THE SUMMARY PAGE

The Main Storyboard will default to the newly created storyboard.
We can change it here instead of coding in the AppDelegate.

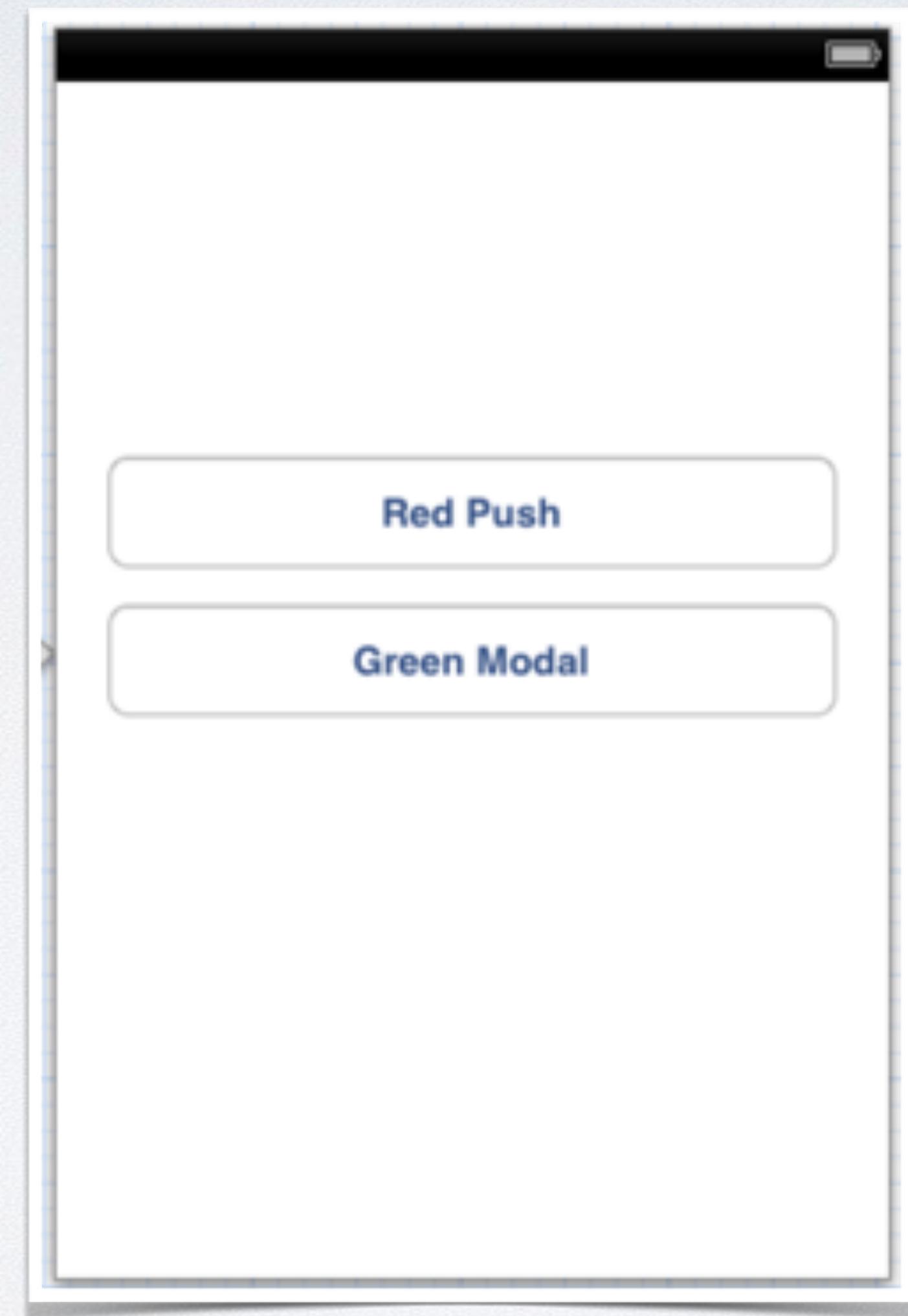


STEP 3: ADD 2 BUTTONS TO THE VIEW CONTROLLER

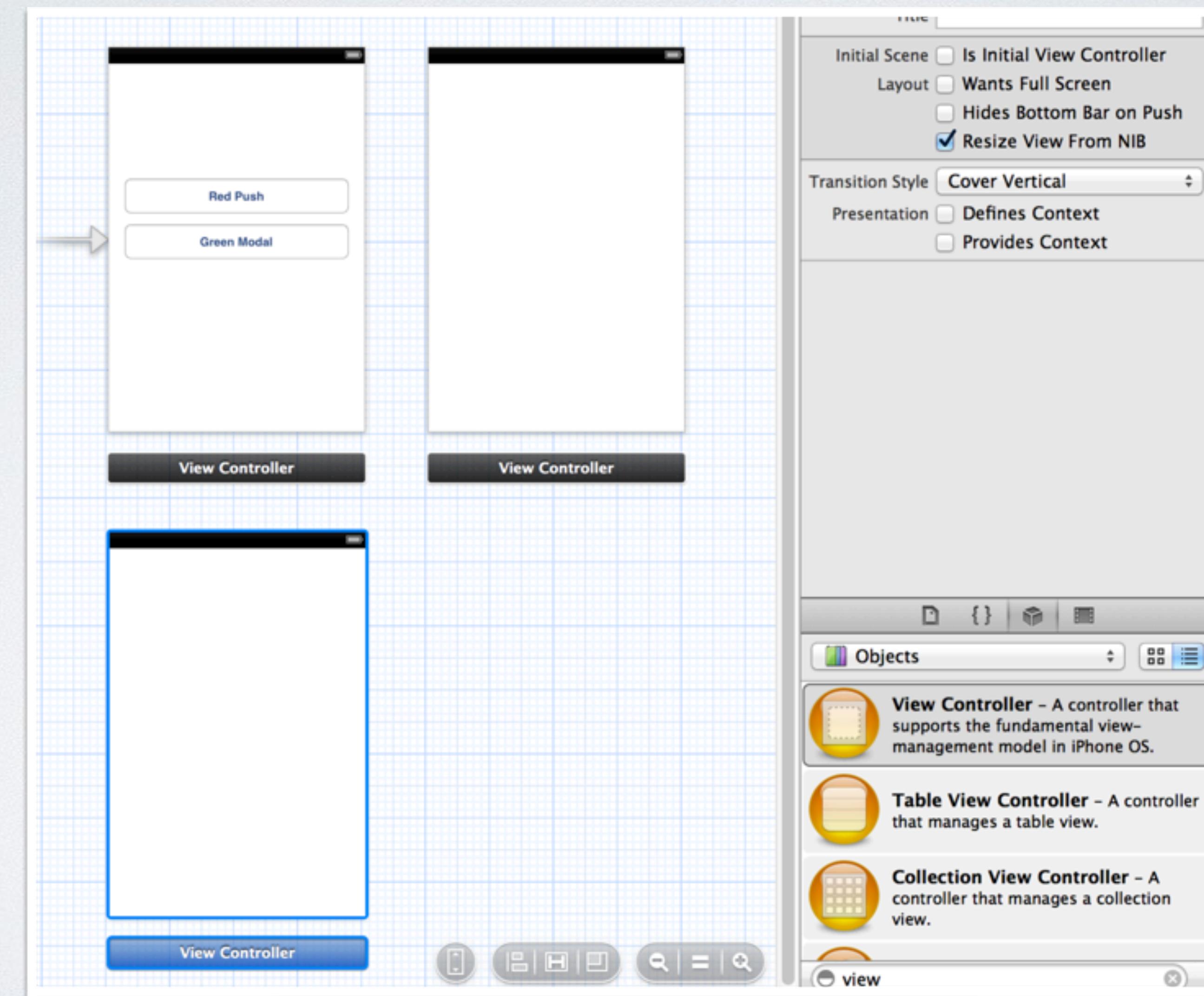
STEP 4: LET'S MAKE THEM THE SAME WIDTH



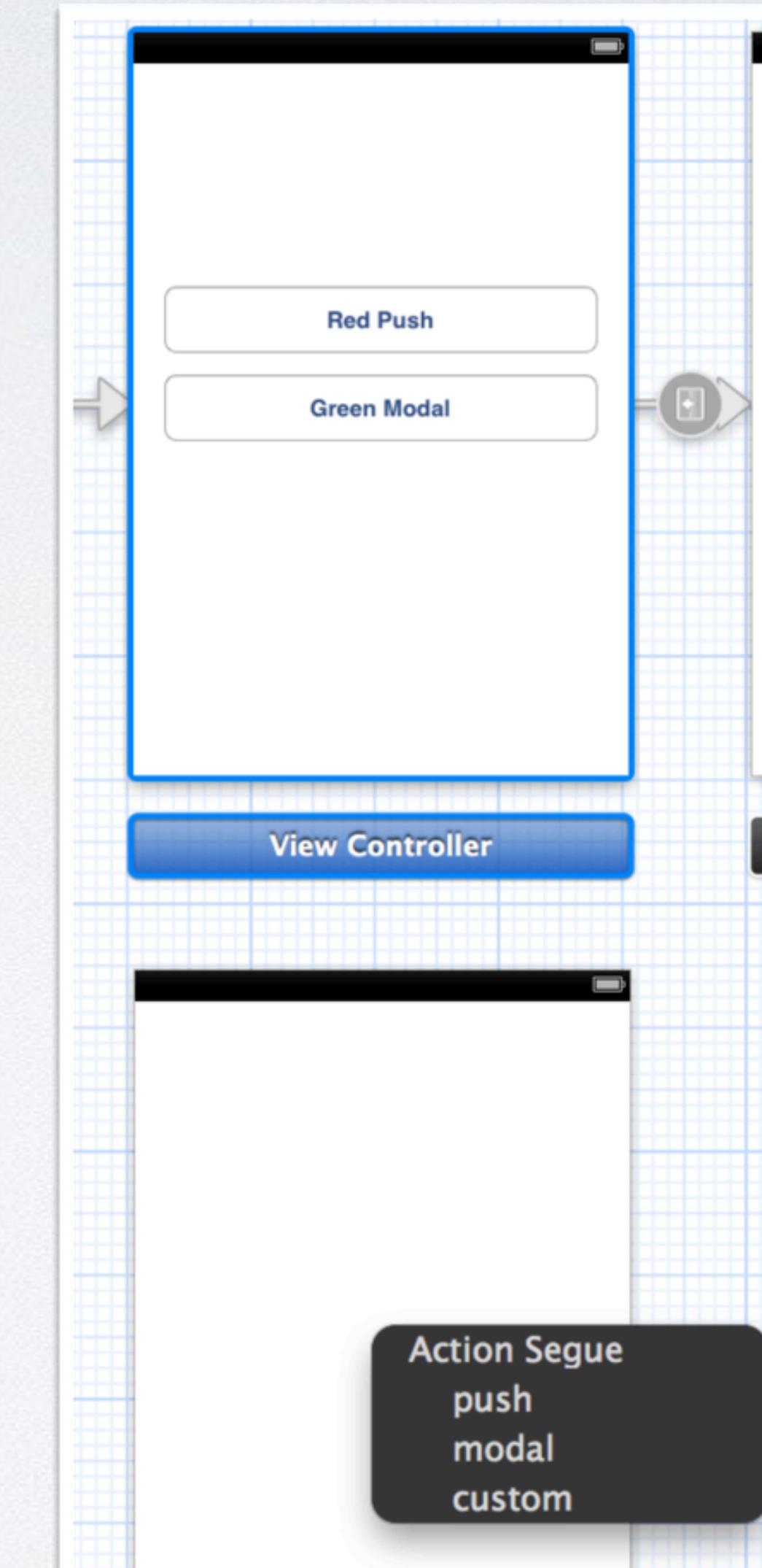
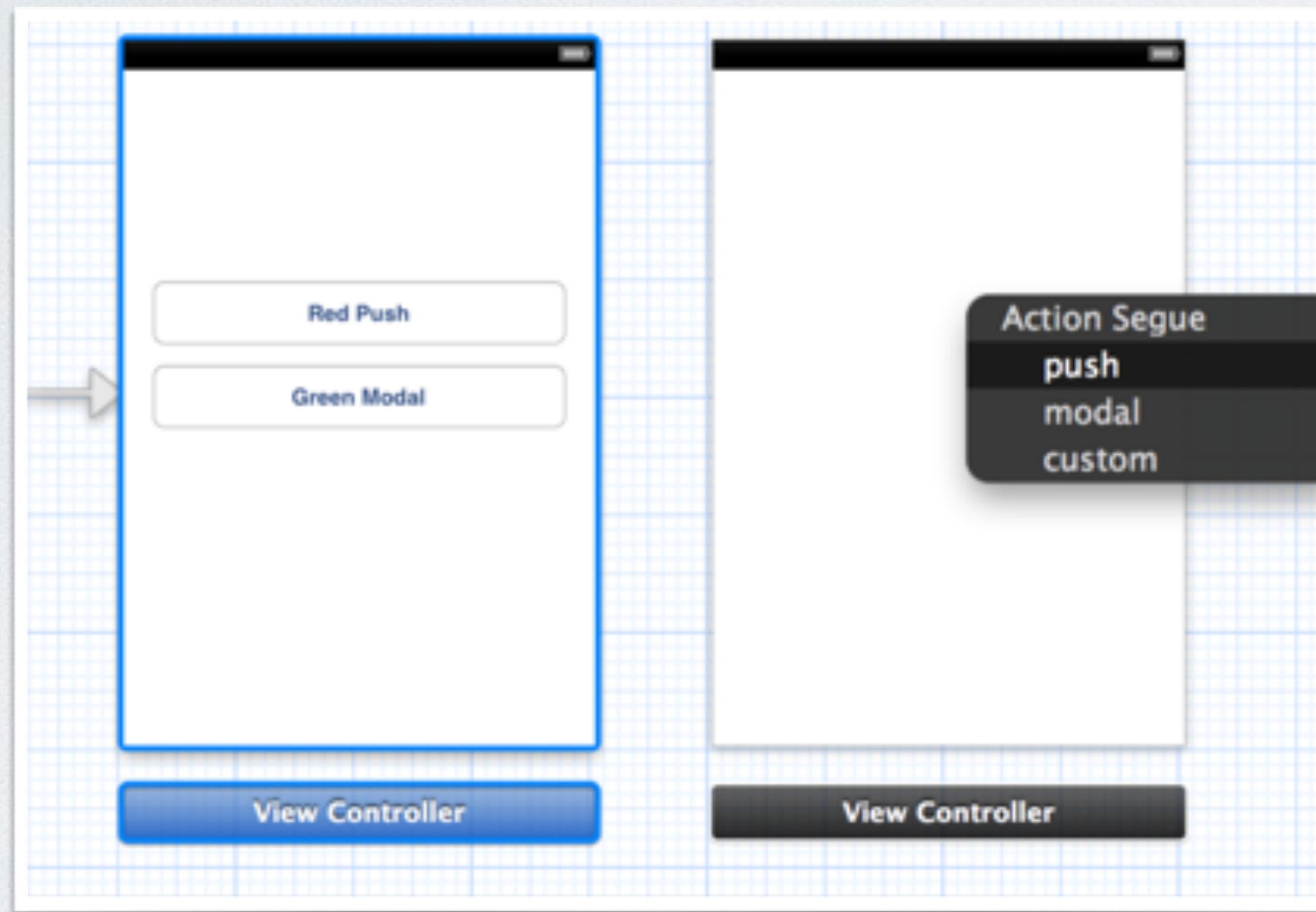
STEP 5: CHANGE THE BUTTON LABELS TO RED AND GREEN



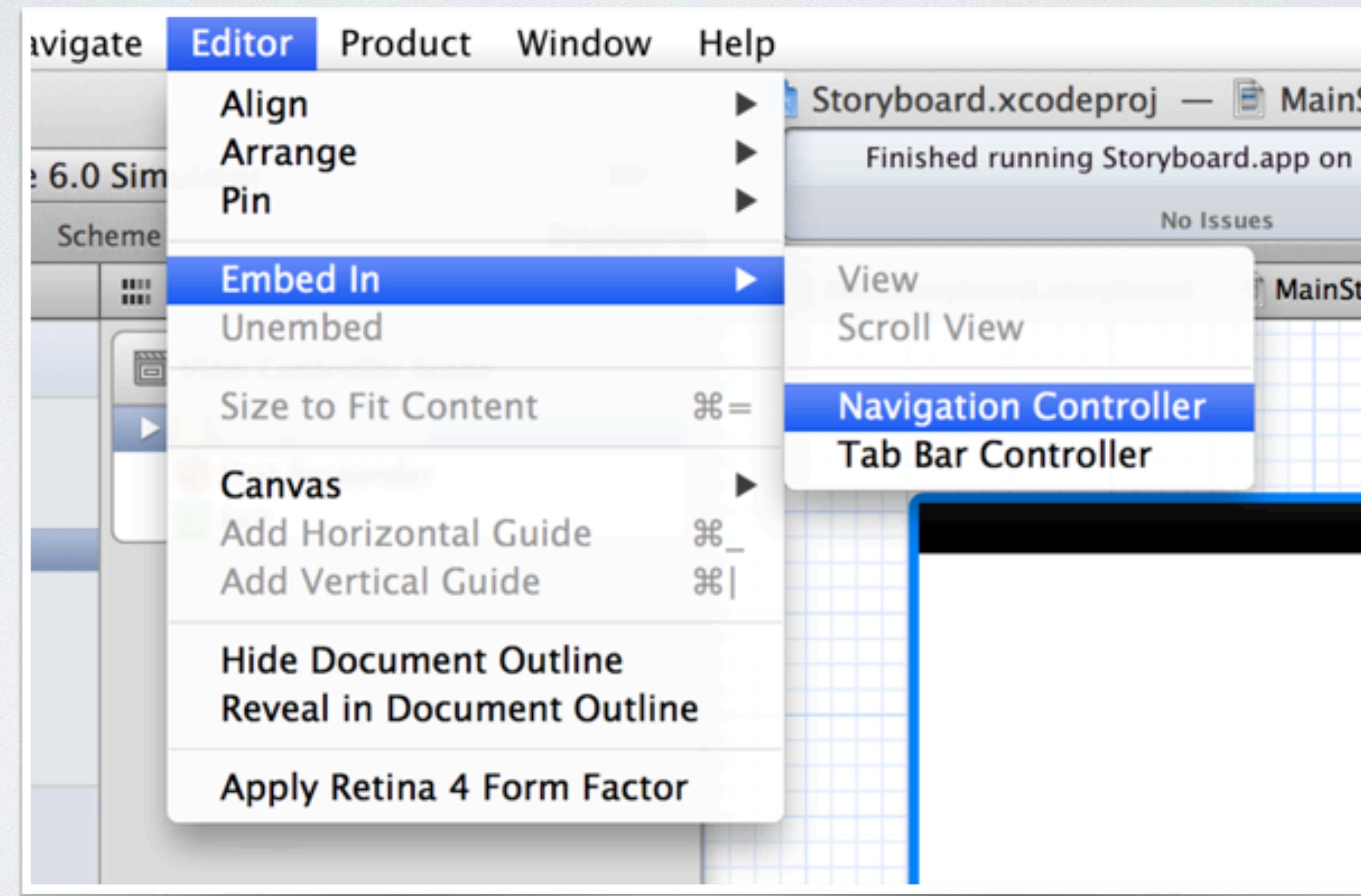
STEP 6: ADD TWO VIEW CONTROLLERS TO THE STORYBOARD BY DRAGGING THEM OVER



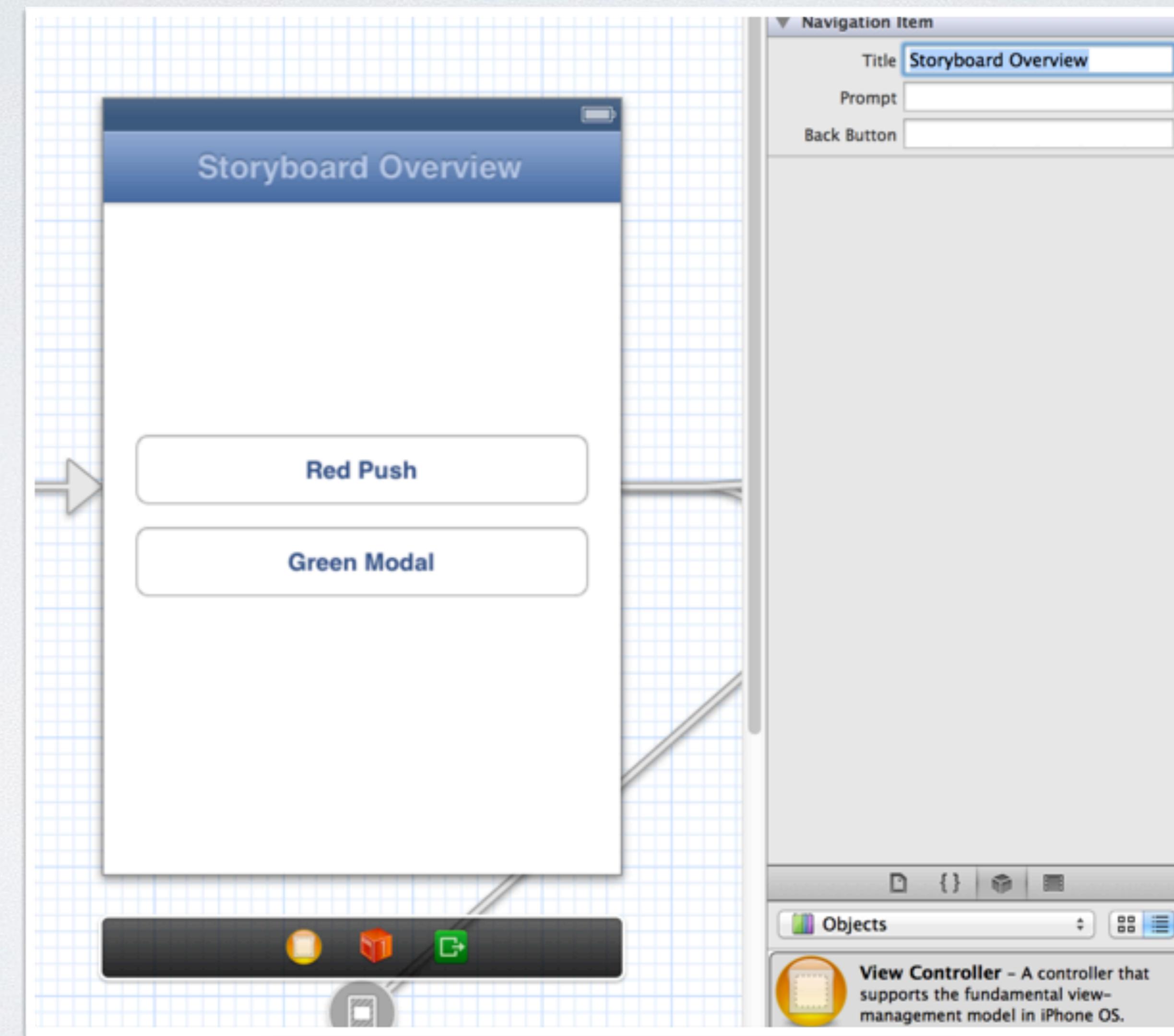
STEP 7: CONNECT THE BUTTONS TO THE NEWVIEW CONTROLLERS BY HOLDING DOWN THE [CONTROL] KEY AND DRAGGING FROM THE BUTTON TO THE NEWVIEW CONTROLLER



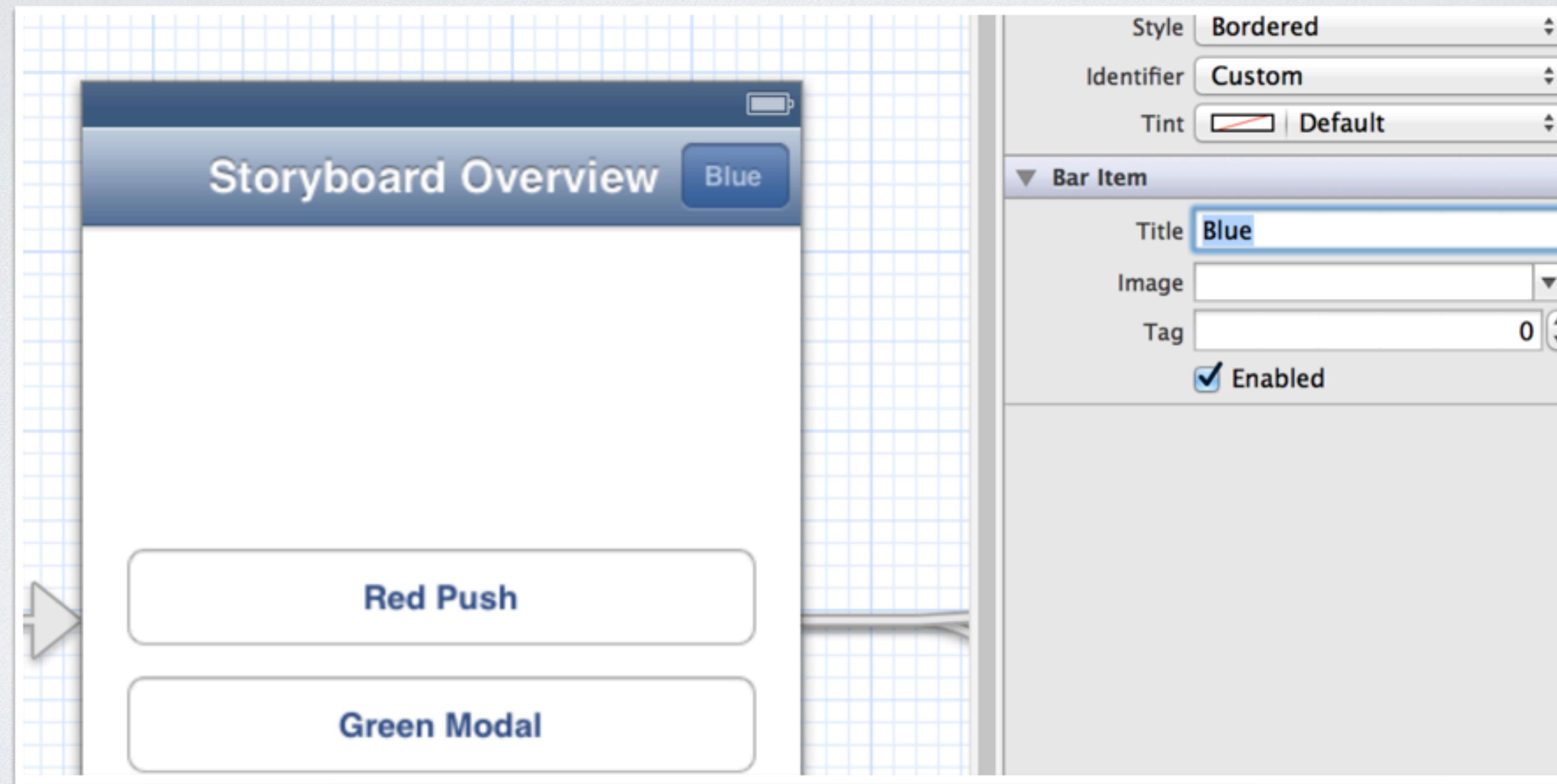
STEP 8: EMBED A NAVIGATION CONTROLLER ON YOUR PRIMARY VIEW



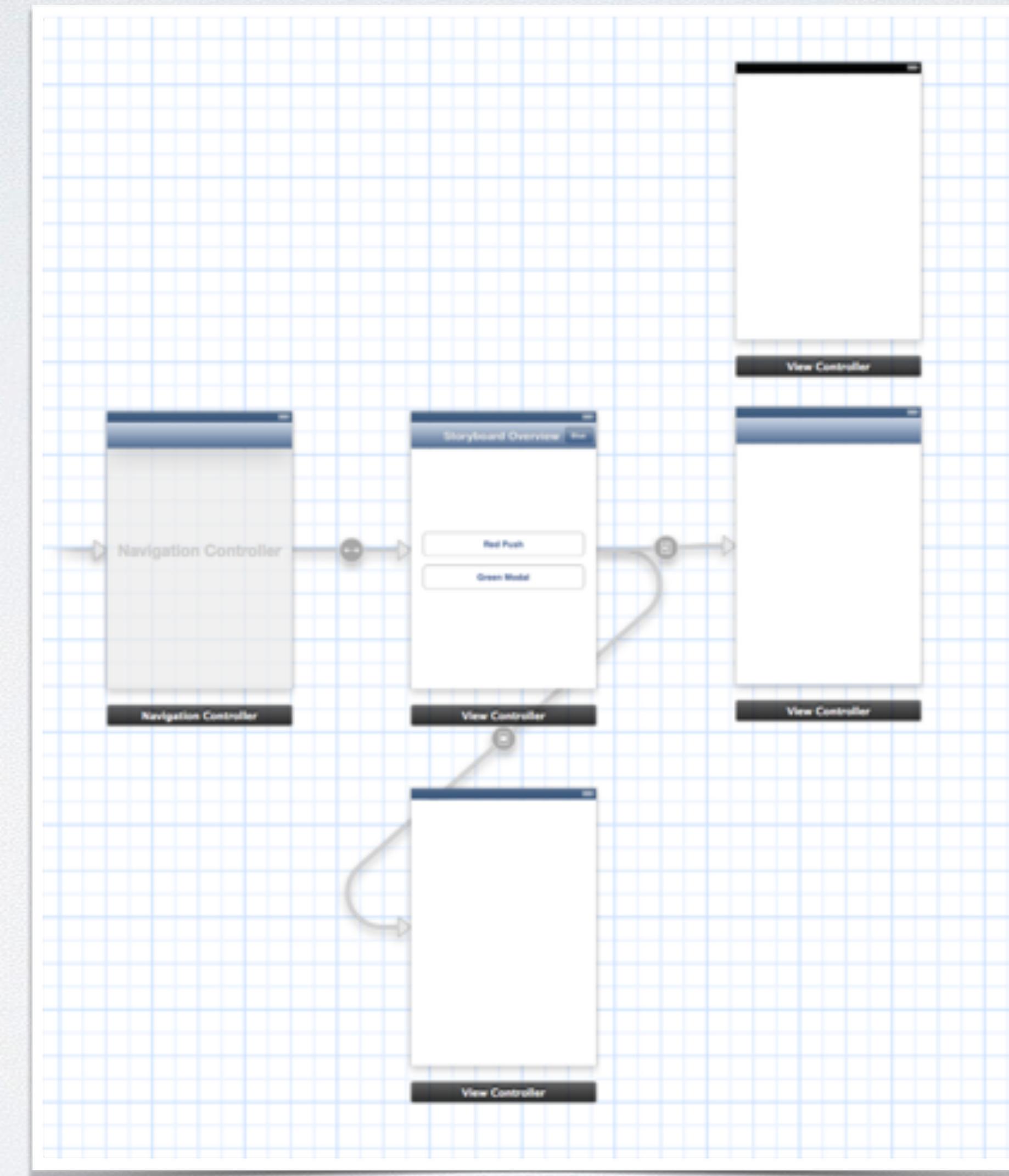
STEP 9: CHANGE THE TITLE OF THE NAVIGATION BAR



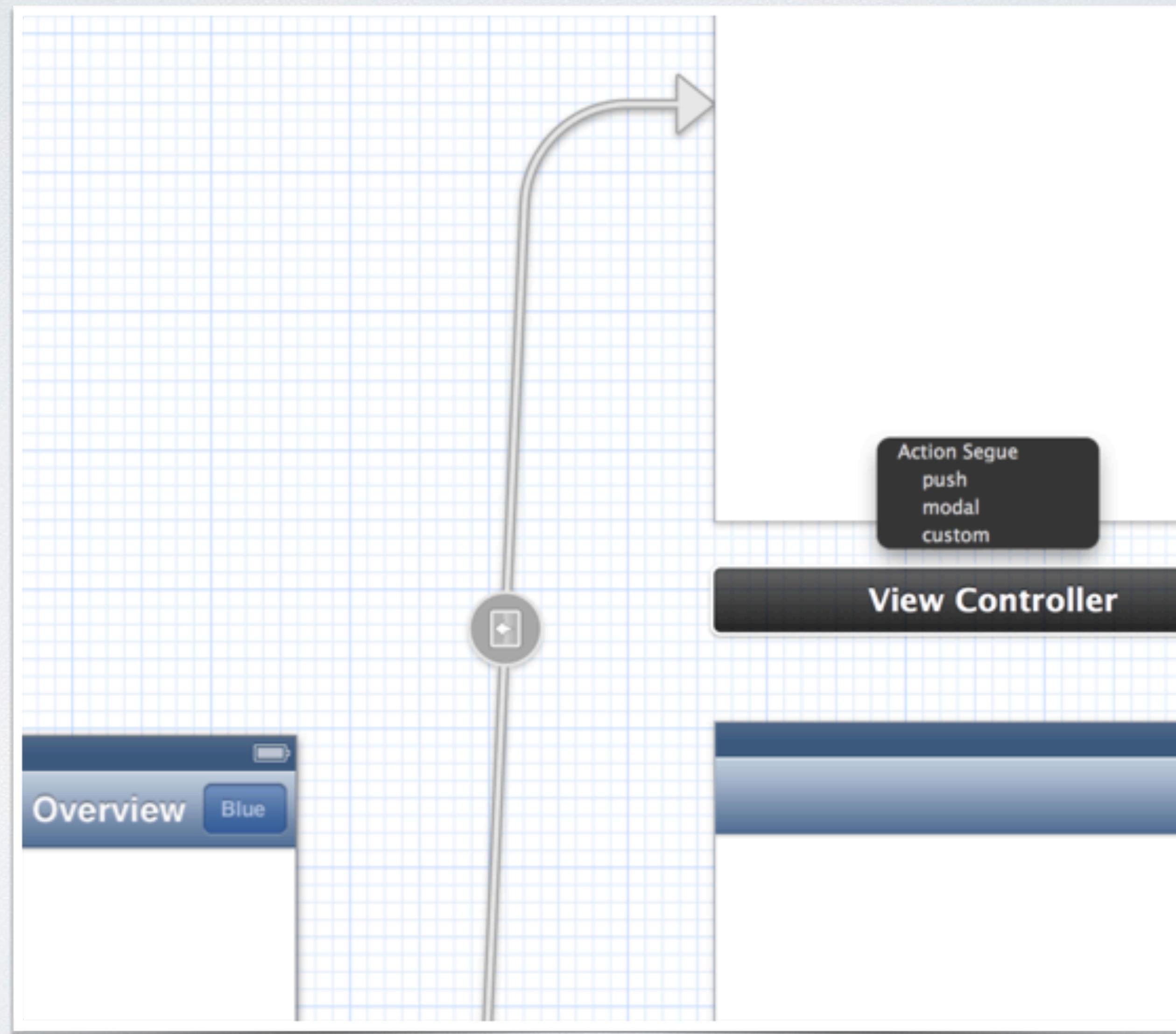
STEP 10: ADD A BAR BUTTON ITEM TO THE NAVIGATION BAR AND CHANGE THE TITLE TO BLUE



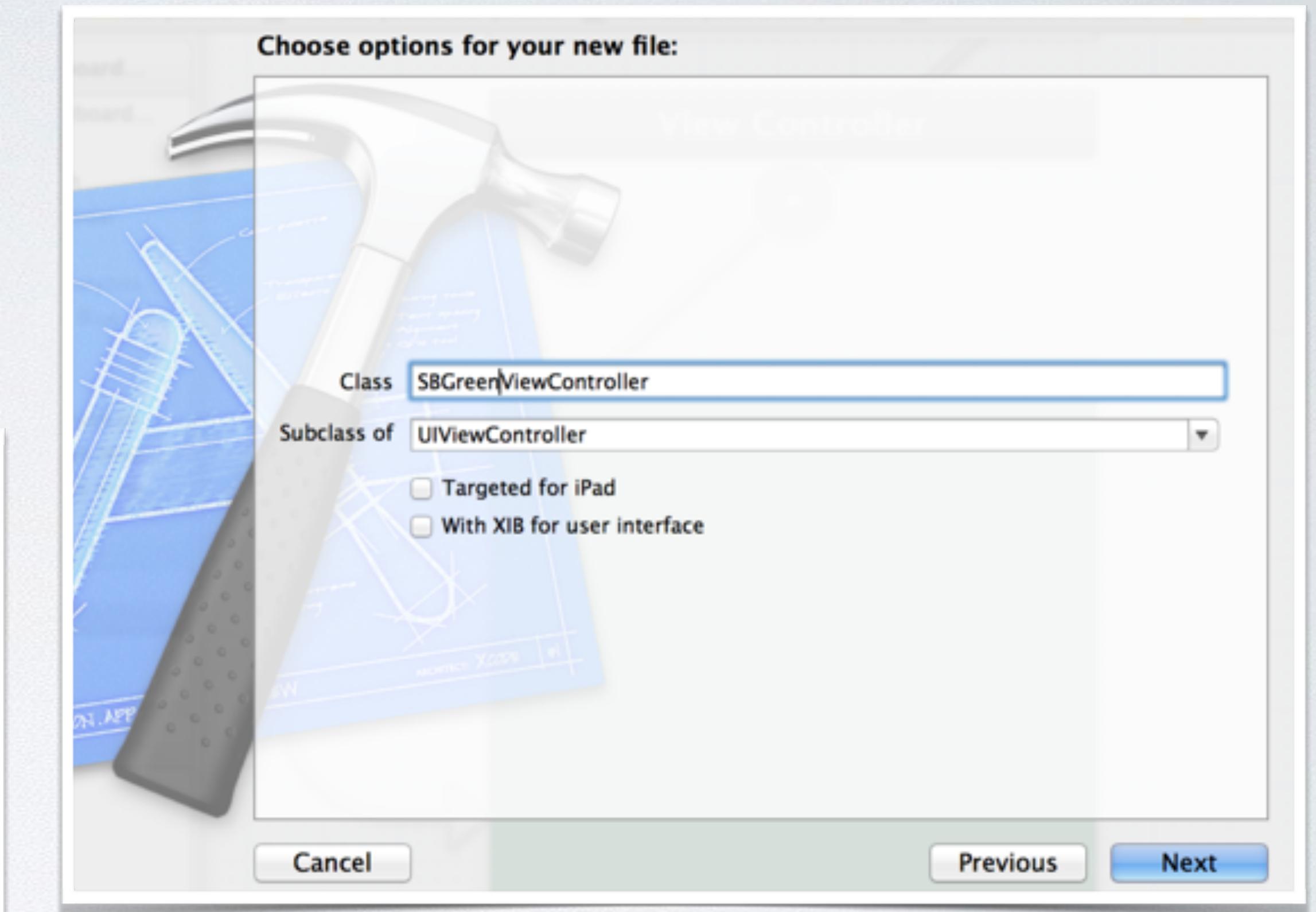
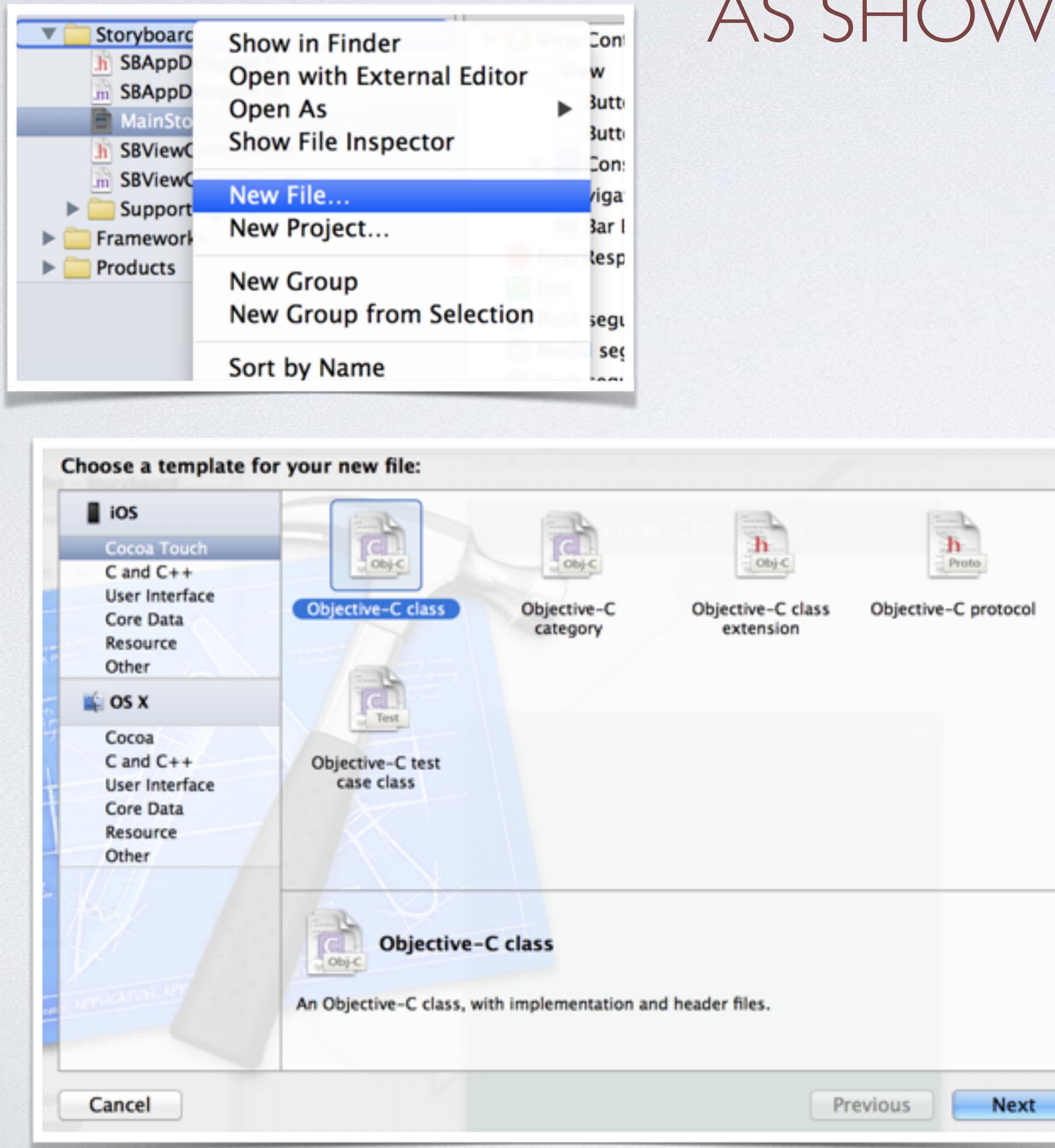
STEP III: ADD OUR THIRD AND FINAL VIEW CONTROLLER TO THE STORYBOARD



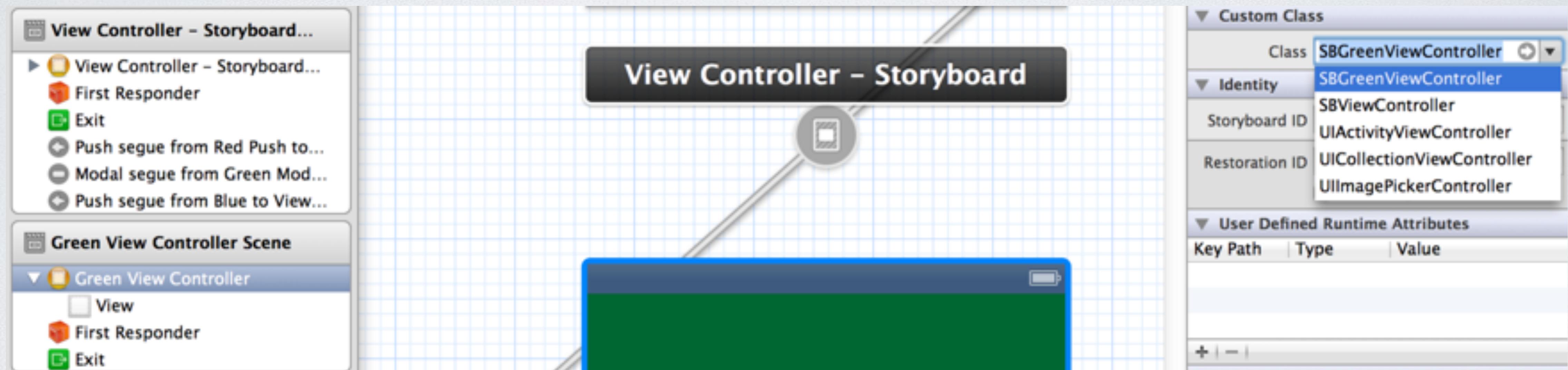
STEP 12: CONNECT OUR BAR BUTTON ITEM TO THE NEW VIEW CONTROLLER AND SELECT PUSH. REMEMBER TO HOLD DOWN THE [CONTROL] BUTTON WHILE CLICKING AND DRAGGING



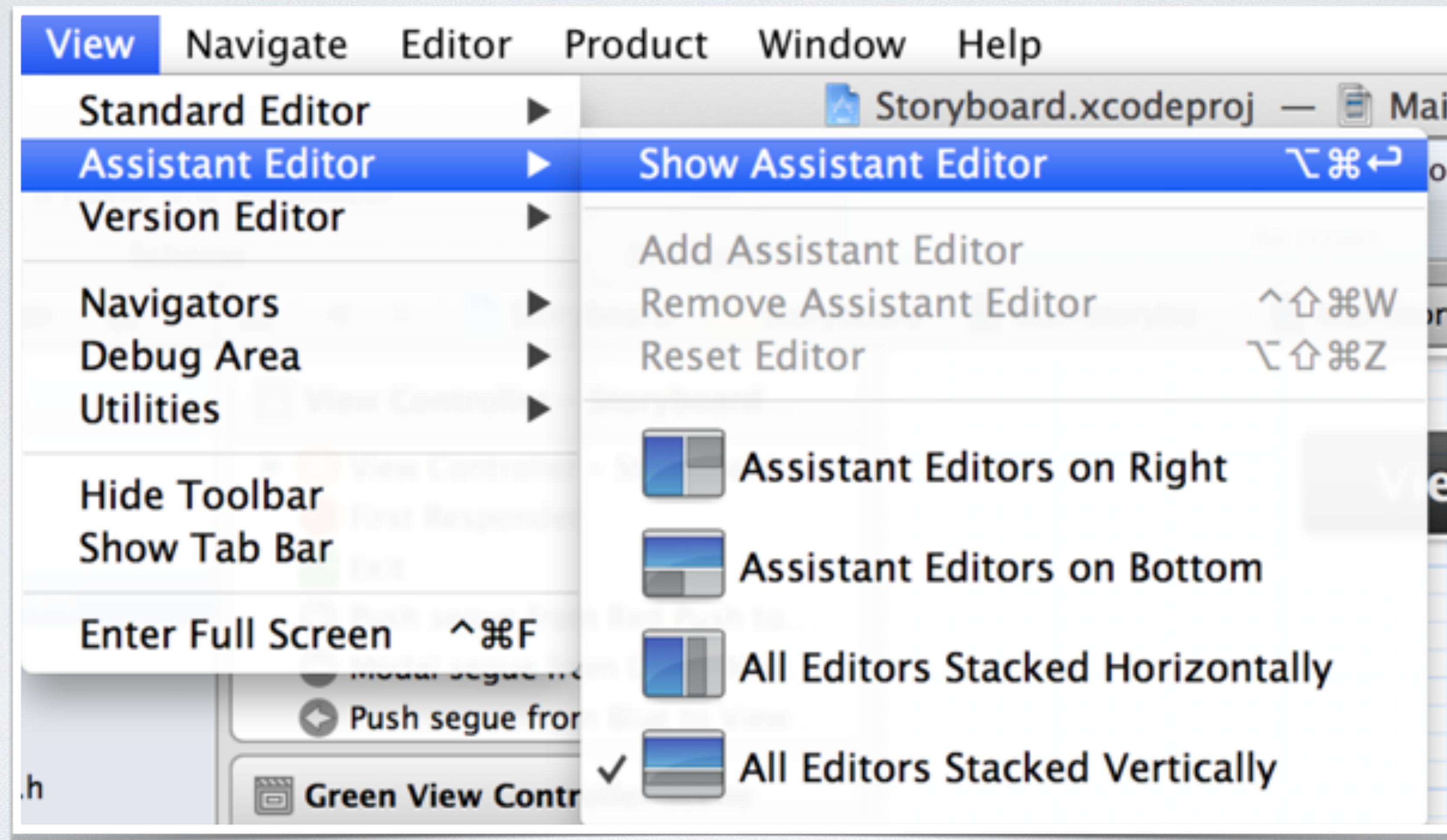
STEP 13: NOW WE NEED TO DO A LITTLE PROGRAMMING. ADD A CONTROLLER OBJECT AND NAME IT “SBGREENVIEWCONTROLLER” AS SHOWN BELOW



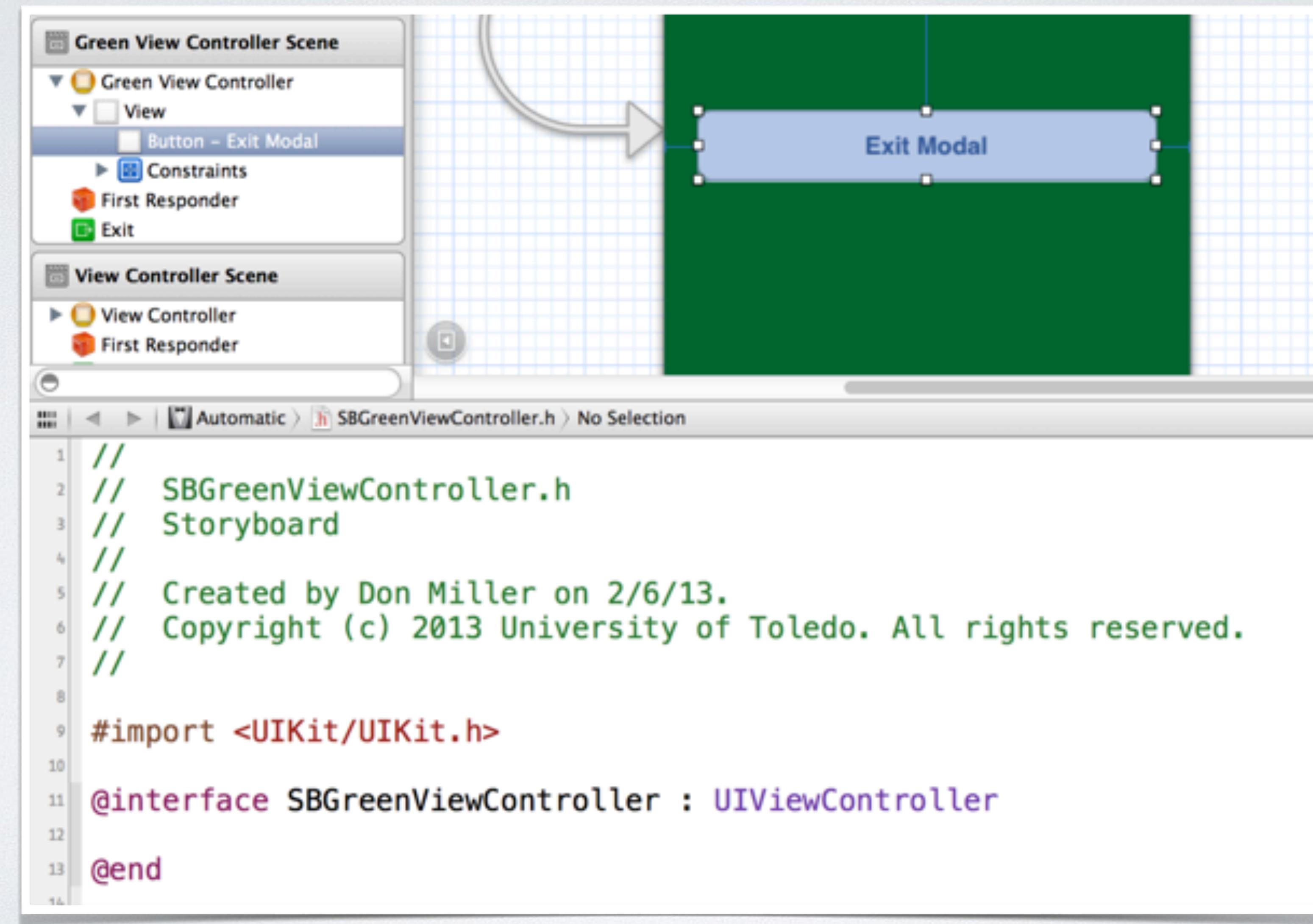
STEP 14: ASSIGN YOUR VIEW IN THE STORYBOARD TO THE NEW OBJECTVIEW CONTROLLER YOU JUST CREATED



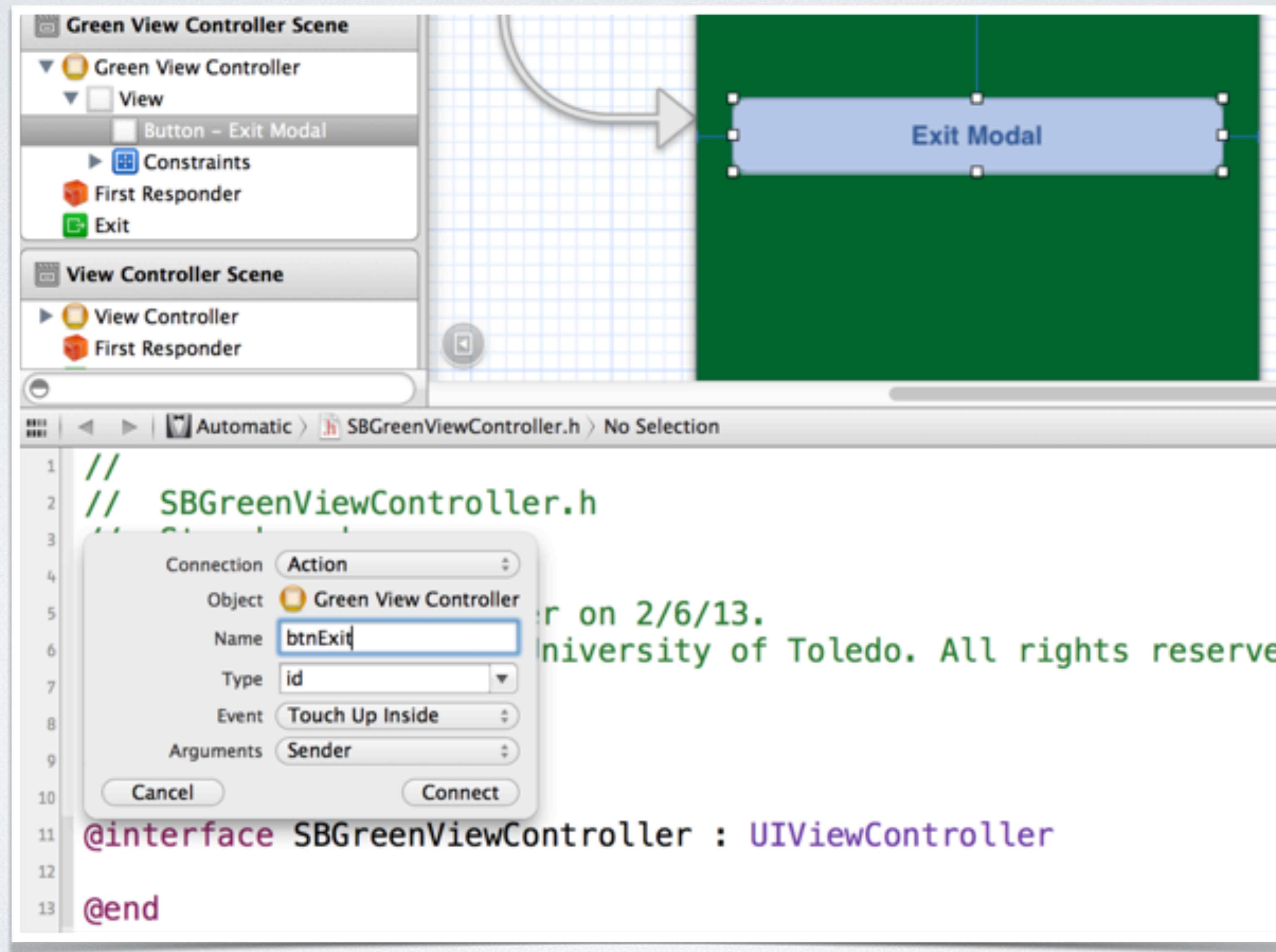
STEP 15: SHOW THE ASSISTANT EDITOR TO HAVE OUR CODE GENERATED FOR US BY CLICKING



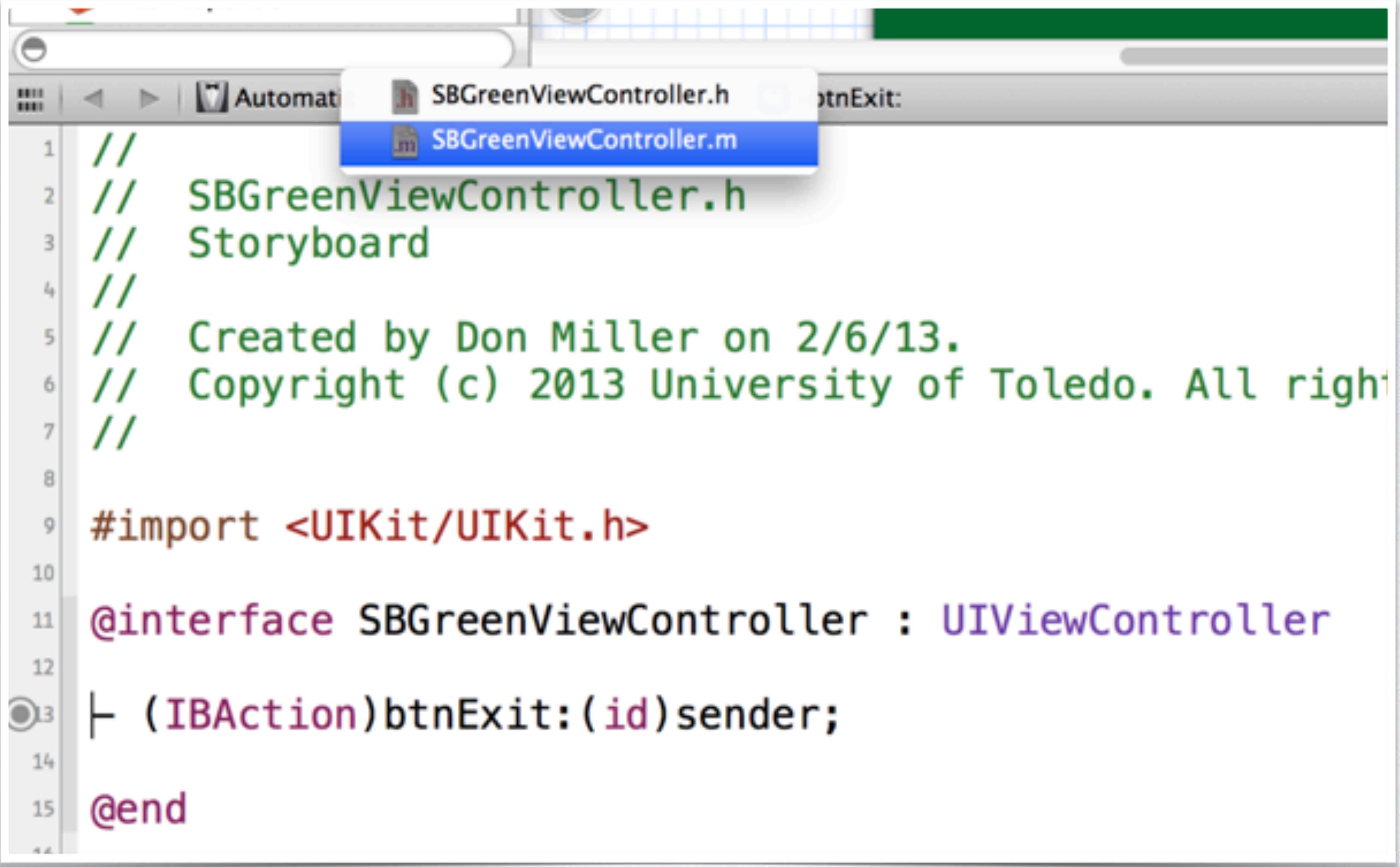
STEP 16: ADD A BUTTON TO THE GREEN VIEW



STEP 17: HOLD DOWN THE CONTROL WHEN CLICKING FROM THE BUTTON TO YOU HEADER FILE AND DRAG IT UNDER THE INTERFACE LINE. CHANGE THE CONNECTION TO ACTION AND THE NAME TO BTNEXIT



STEP 18: CHANGE TO YOUR IMPLEMENTATION FILE BY CLICKING THE SBGREENVIEWCONTROLLER.H TITLE AT THE TOP OF THE ASSISTANT EDITOR



The screenshot shows the Xcode interface with the Assistant Editor open. The top bar displays the file names "Automat" and "SBGreenViewController.h" on the left, and "btnExit:" on the right. Below the bar, the "SBGreenViewController.m" file is selected, indicated by a blue highlight. The code editor shows the implementation file's content:

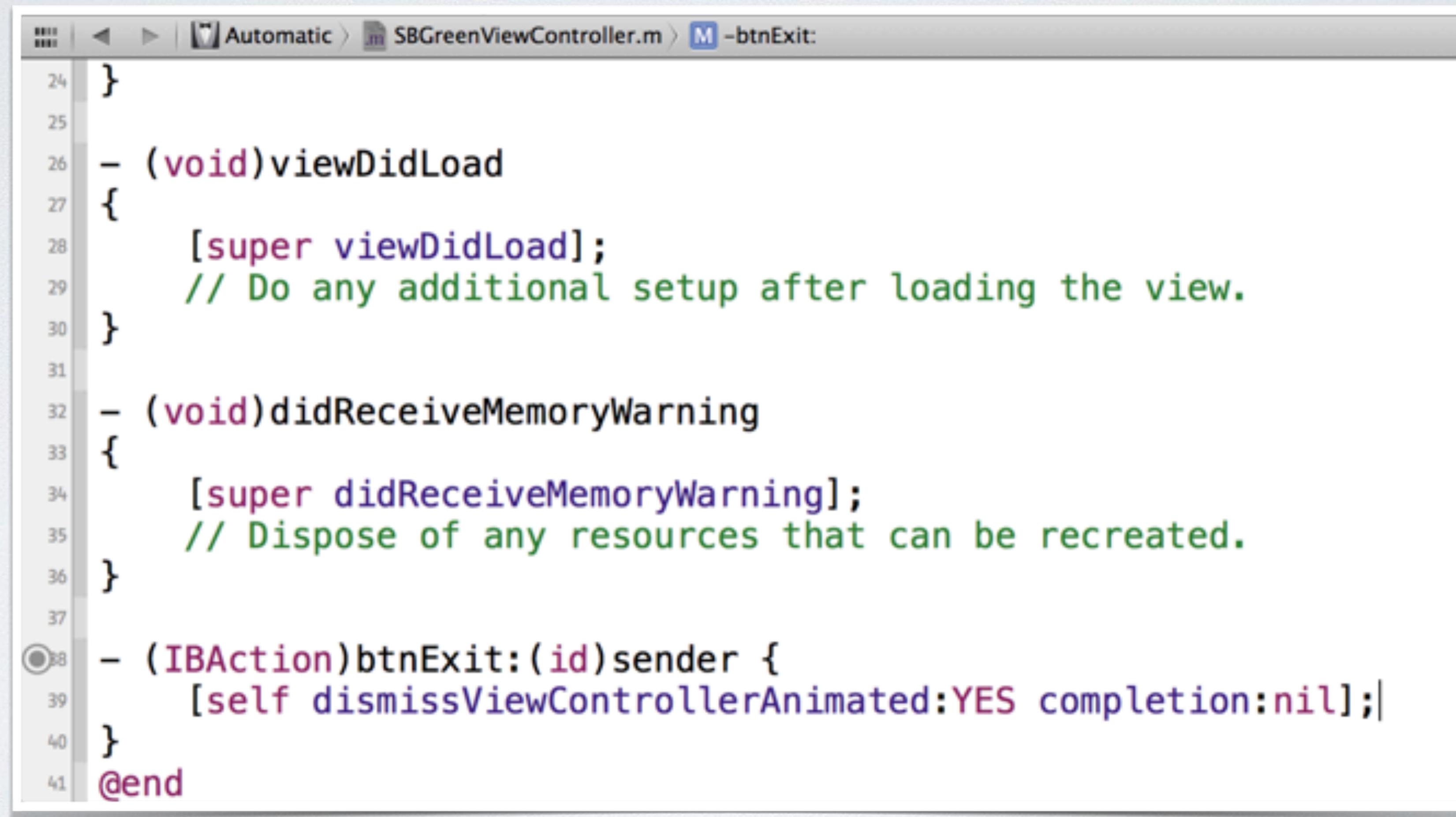
```
// SBGreenViewController.h
// Storyboard
//
// Created by Don Miller on 2/6/13.
// Copyright (c) 2013 University of Toledo. All rights reserved.

#import <UIKit/UIKit.h>

@interface SBGreenViewController : UIViewController
    -(IBAction)btnExit:(id)sender;
@end
```



STEP 19: ADD THE FOLLOWING LINE OF CODE IN THE BTNEXIT METHOD TO DISMISS THE MODAL WINDOW



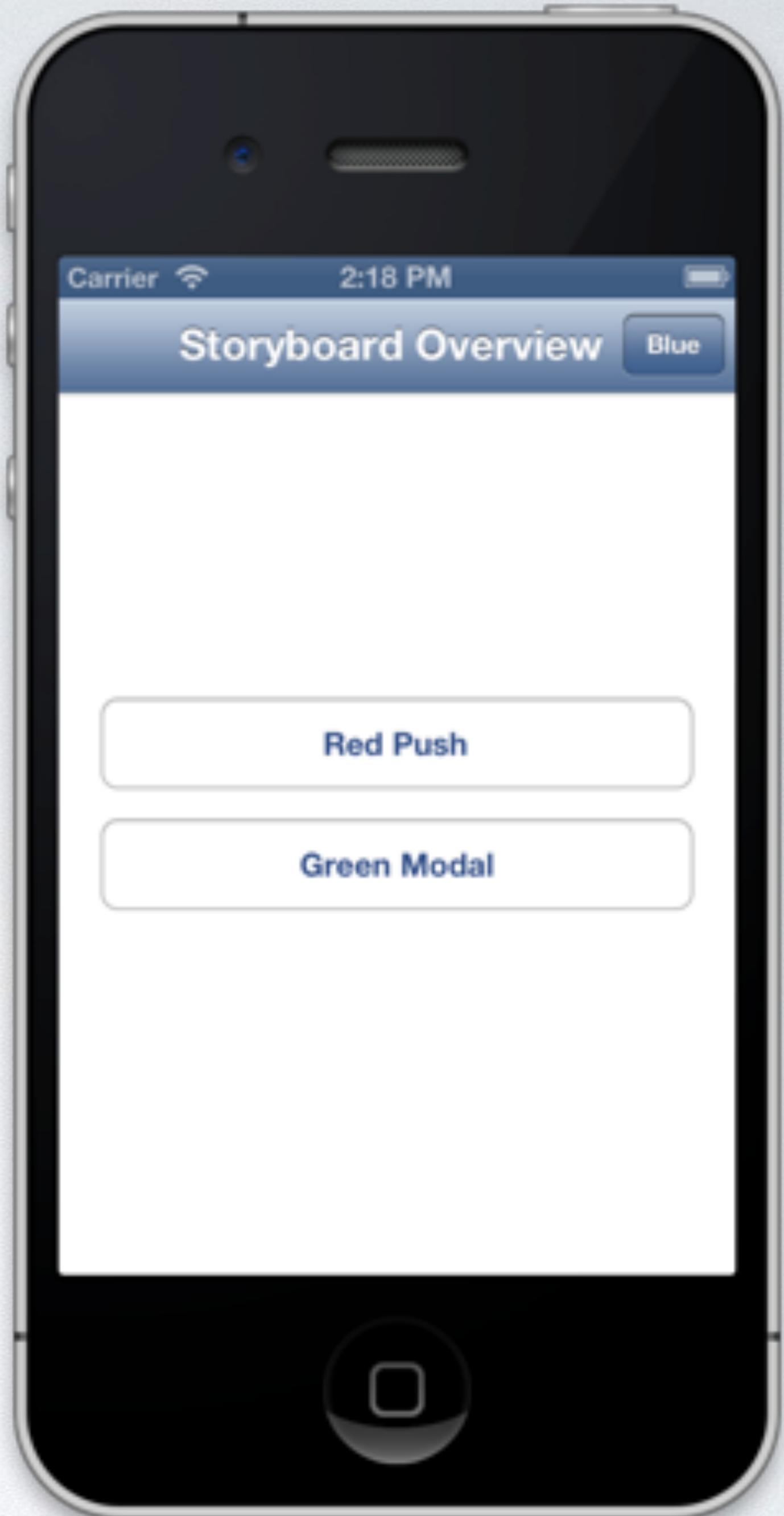
The screenshot shows the Xcode interface with the file `SBGreenViewController.m` open. The code editor displays the following Objective-C code:

```
24 }
25
26 - (void)viewDidLoad
27 {
28     [super viewDidLoad];
29     // Do any additional setup after loading the view.
30 }
31
32 - (void)didReceiveMemoryWarning
33 {
34     [super didReceiveMemoryWarning];
35     // Dispose of any resources that can be recreated.
36 }
37
38 - (IBAction)btnExit:(id)sender {
39     [self dismissViewControllerAnimated:YES completion:nil];
40 }
41 @end
```

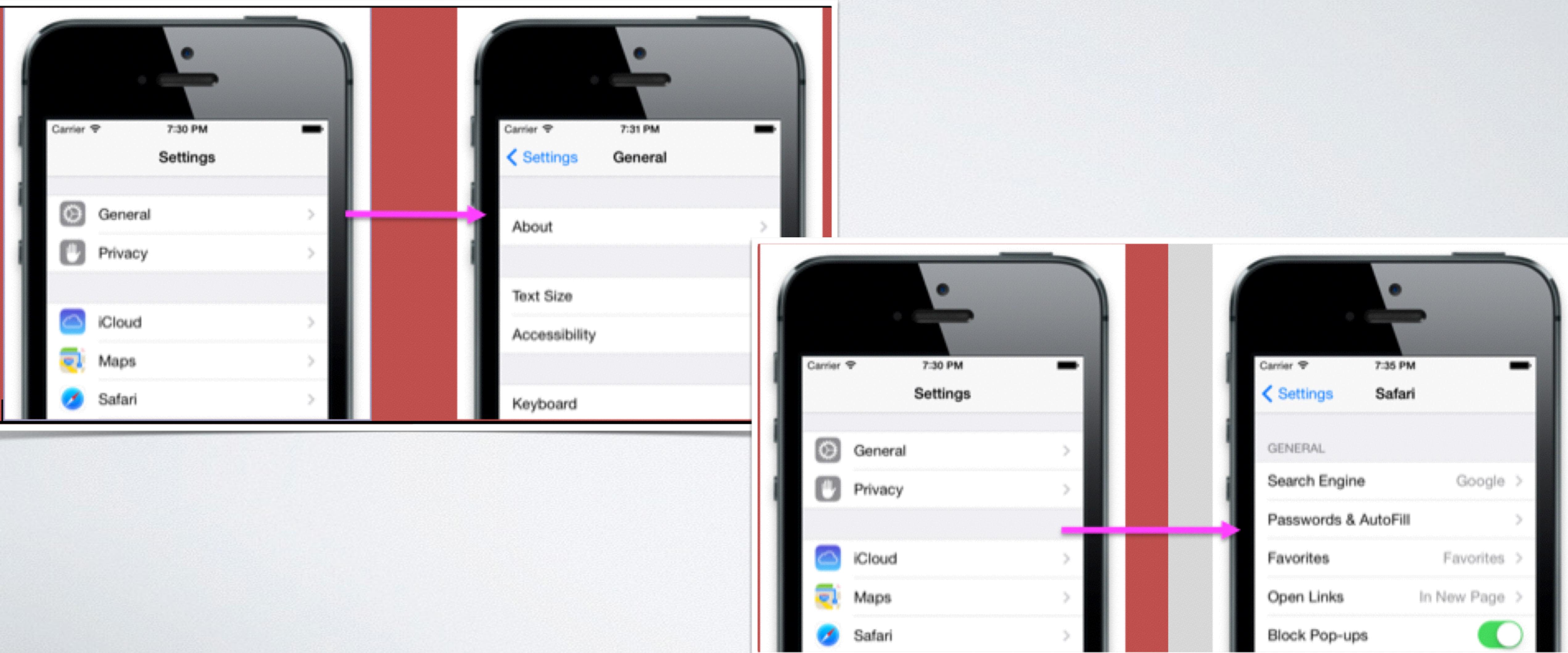
The line `[self dismissViewControllerAnimated:YES completion:nil];` is highlighted in purple, indicating it is the new code being added.



STEP 20: RUN IT!

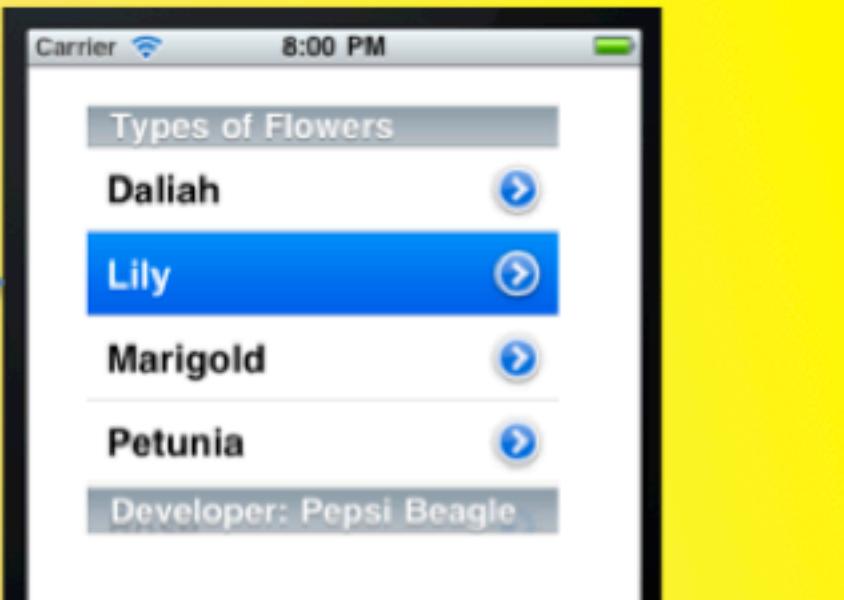


UITABLEVIEW OVERVIEW

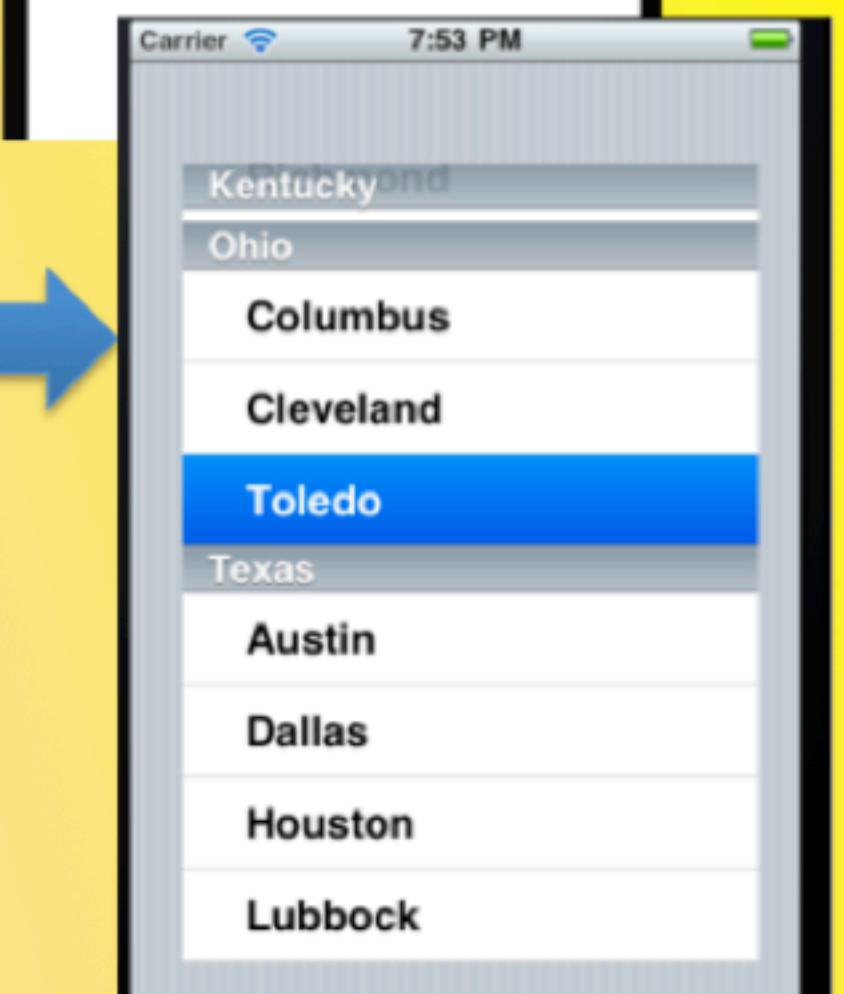


UITABLEVIEW OVERVIEW

A simple UI
Table View with
4 rows



A sectioned UI
Table View

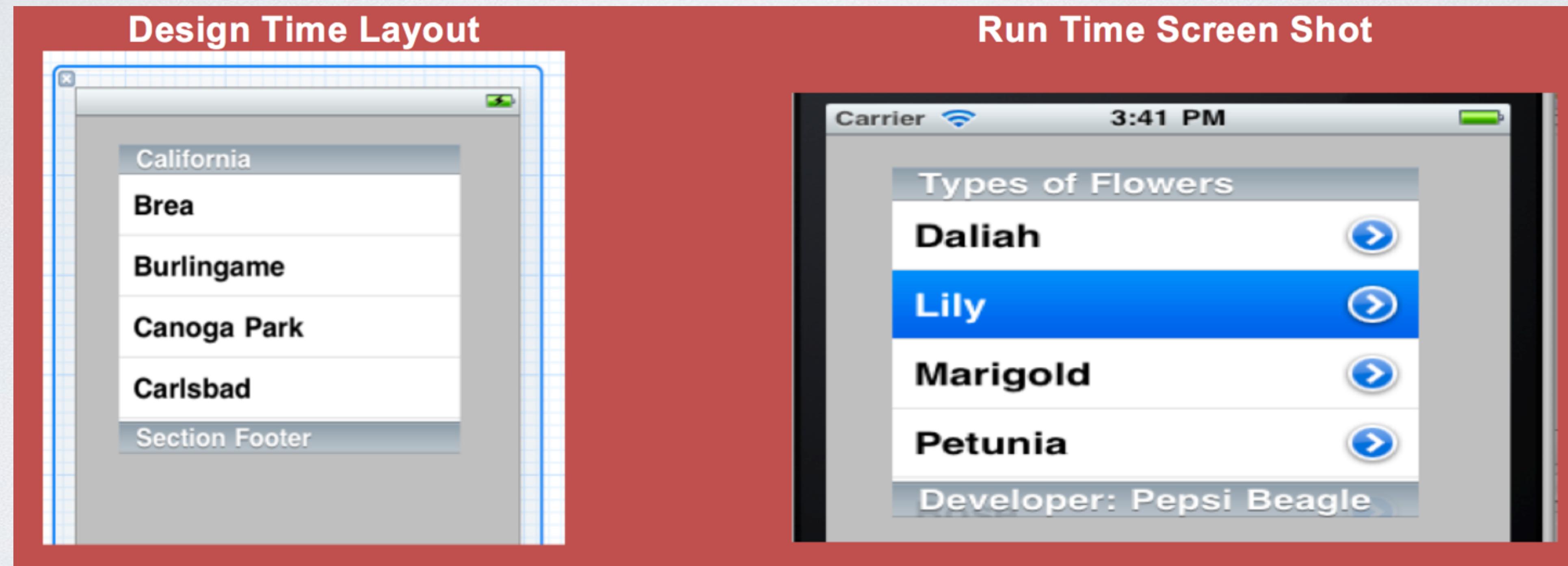


- ❖ Programmatically, the UIKit identifies rows and sections through their **index number**.
- ❖ Sections are numbered **0 through n-1** from the top of a table view to the bottom.
- ❖ Rows are numbered **0 through n-1** within a section.
- ❖ As mentioned earlier, the table view comes in one of two basic styles: **plain or grouped**.

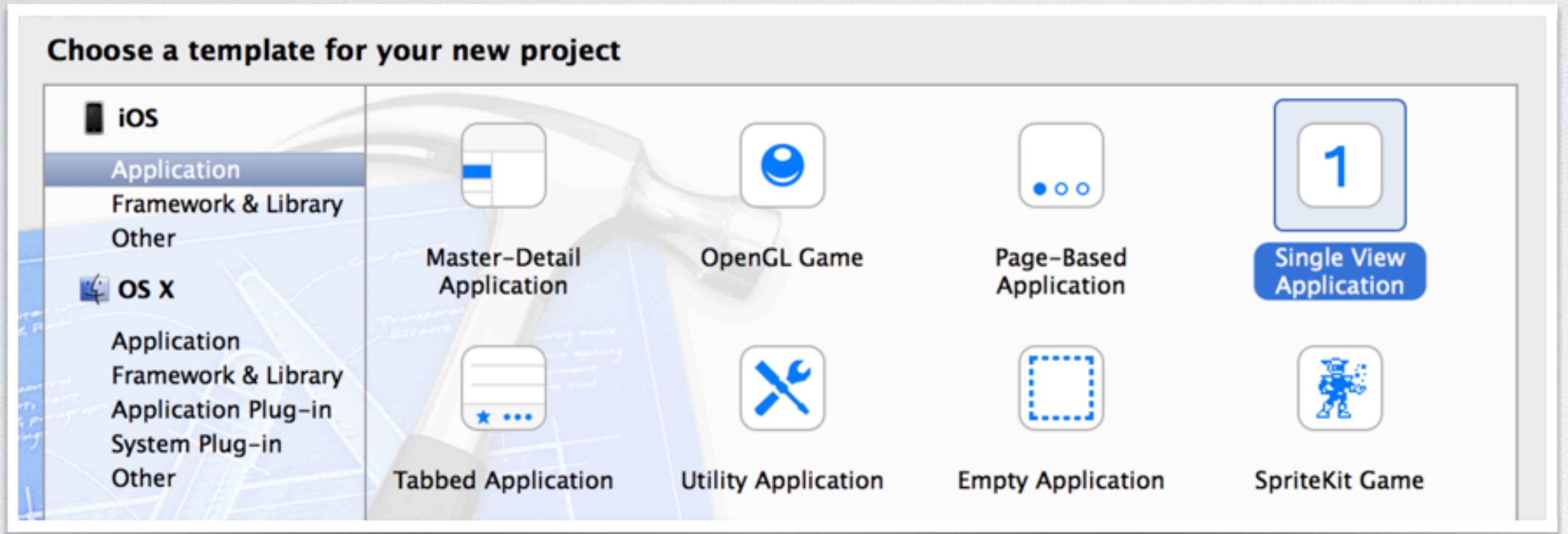
If you employ a table view object, it is always a good practice to **declare the protocol conformance in the header file**, such as:

```
@interface xyzViewController : UIViewController  
<UITableViewDataSource, UITableViewDelegate> {  
    -----  
    -----  
    -----  
    -----  
}
```

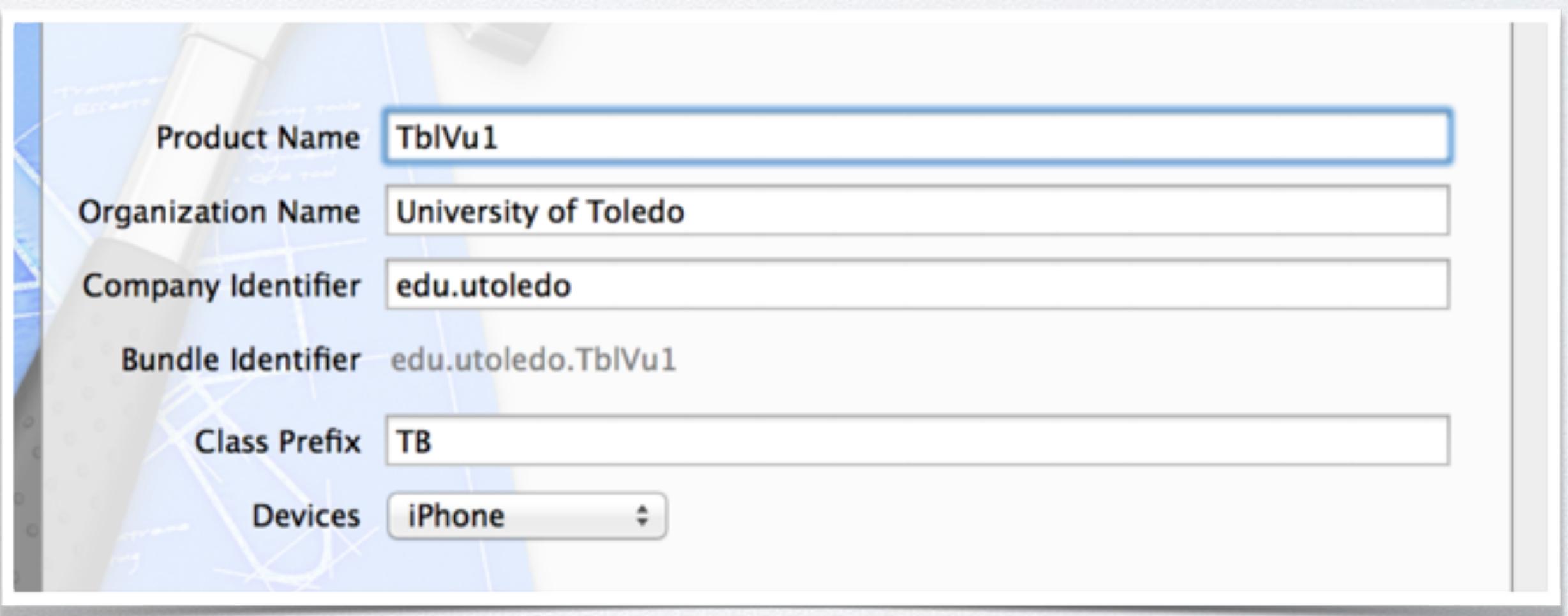
TABLE VIEW EXAMPLE



STEP 1: CREATE A SINGLE VIEW BASED APP.

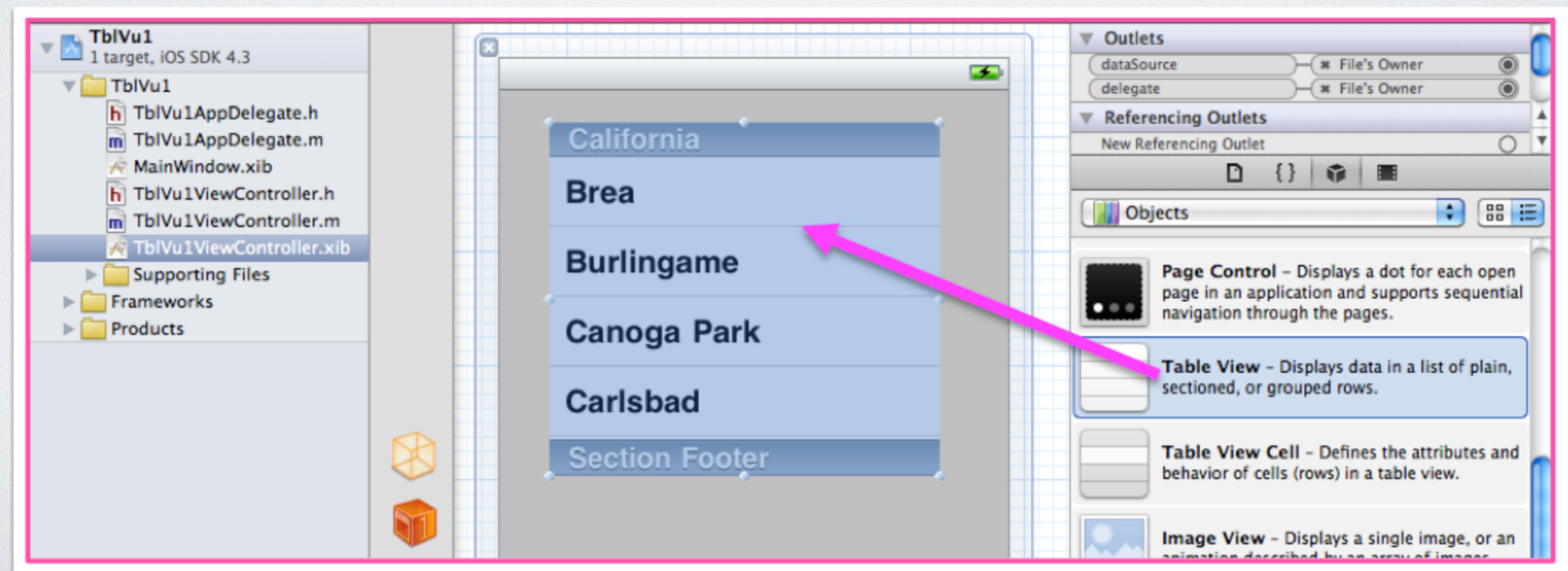


- Let's name it: **TblVu1**



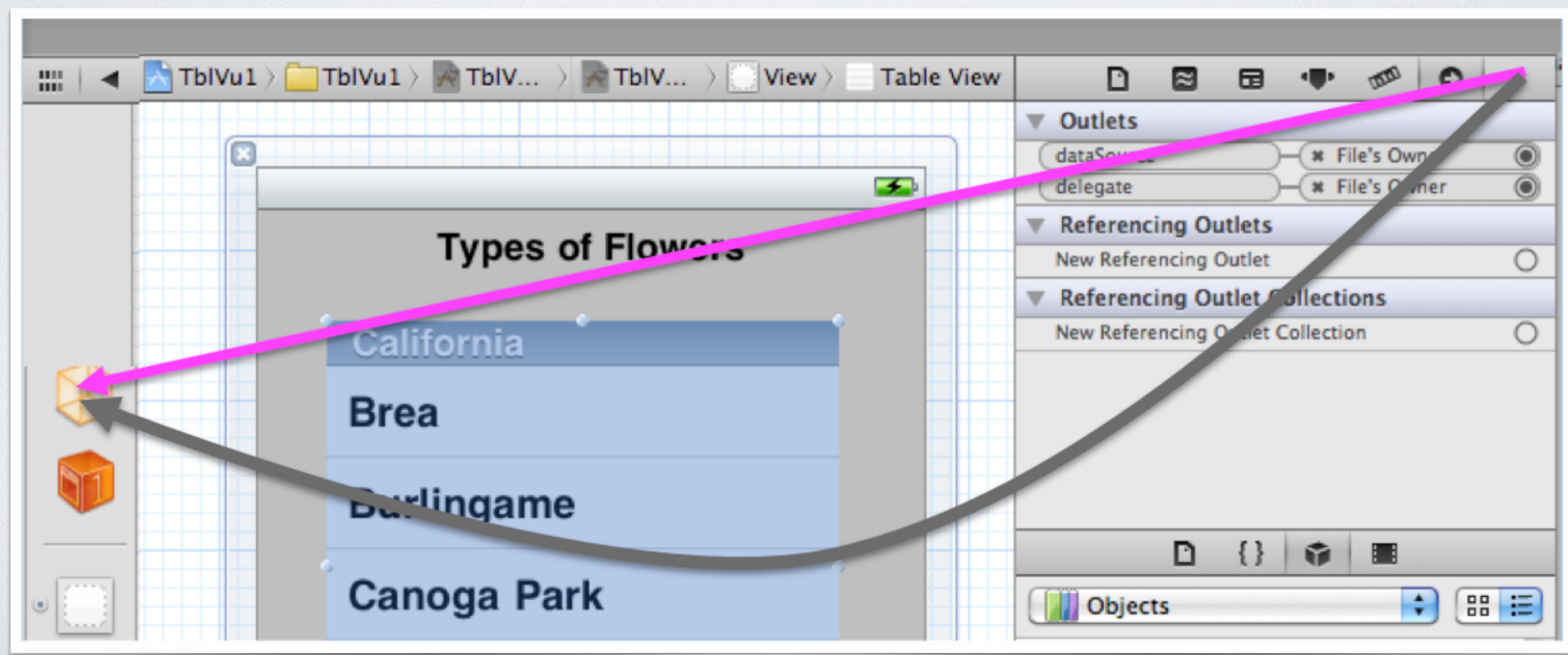
STEP 2: DRAW TABLE VIEW

- Open the Main.storyboard file.
- Draw a label and change its text to "Types of Flowers".
- Now draw a TableView. The system will display some dummy entries (in the design view) as shown below:



STEP 3: DATASOURCE & DELEGATE

- Open the **Property Inspector** of the **Table View**, then drag its **dataSource** and **delegate** outlets and drop those on the **File's Owner**.



STEP 3A: OPEN THE TBLVU1VIEWCONTROLLER.H FILE AND INSERT THE FOLLOWING CODE

```
// TblVu1ViewController.h
// TblVu1

#import <UIKit/UIKit.h>

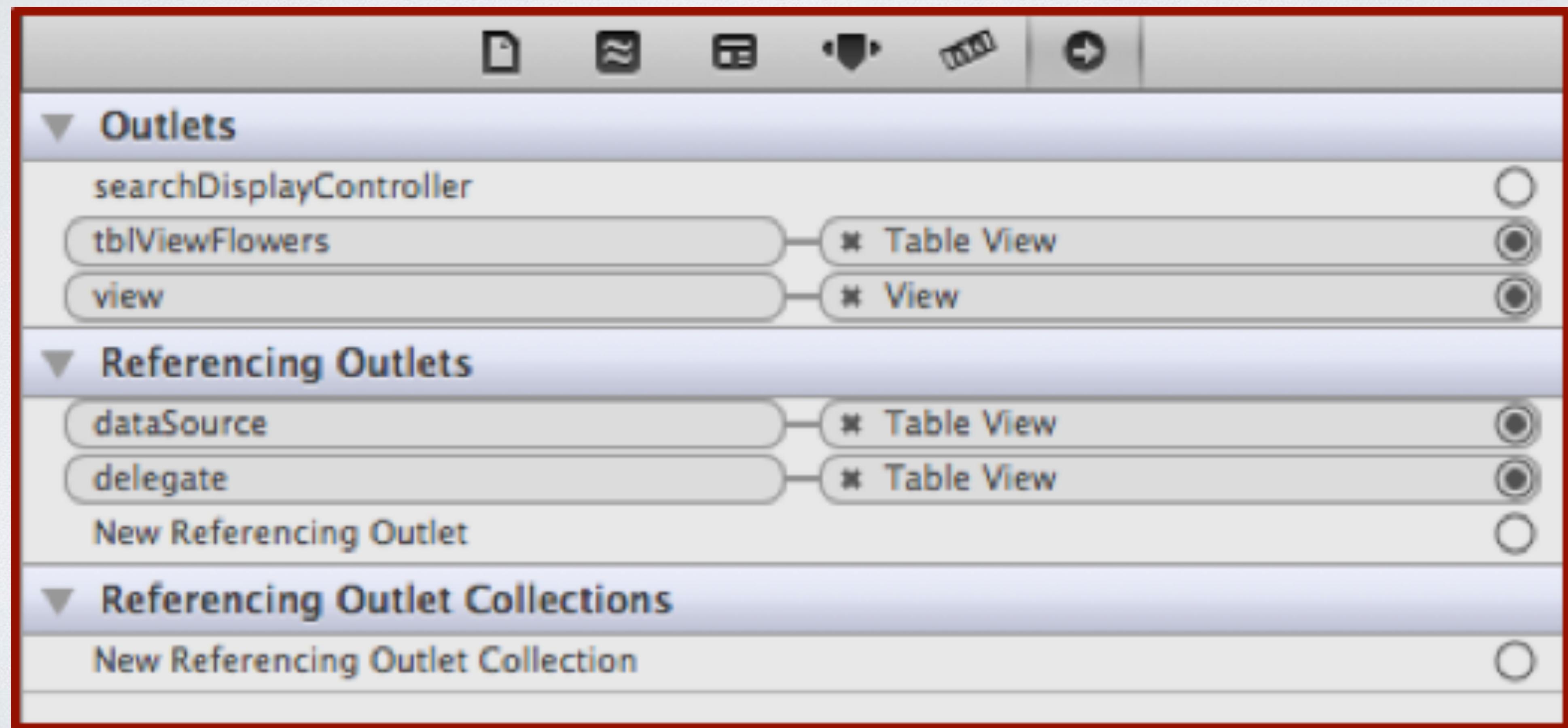
@interface TblVu1ViewController : UIViewController
<UITableViewDataSource, UITableViewDelegate>
{
}

@property (nonatomic, strong) IBOutlet UITableView *tblViewFlowers;
@property (nonatomic, strong) NSMutableArray *arrayFlowerNames;

@end
```



STEP 3B: GO BACK TO THE STORYBOARD FILE, AND CONNECT THE TBLVIEWFLOWERSOUTLET TO THE UI TABLE VIEW.



STEP 4: DEVELOPING THE CODE IN THE IMPLEMENTATION FILE

Just like the UIPickerView view, the UITableView has numerous datasource and delegate methods. In this section, we will develop and discuss the relevant code.

4A: Synthesize the properties.

```
// TblVulViewController.m
#import "TblVulViewController.h"
@implementation TblVulViewController
@synthesize tblViewFlowers, arrayFlowerNames;
```

4B: Populate the arrayFlowerNames array in the viewDidLoad method:

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    // Populate the arrayFlowerNames array
    arrayFlowerNames = [[NSMutableArray alloc]
        initWithObjects: @"Daliah", @"Lily", @"Marigold", @"Petunia", @"Rose", nil];
}
```



4C: EMPLOY SOME OF THE DATA SOURCE METHODS OF THE TABLEVIEW OBJECT TO SUPPLY THE NUMBER OF SECTIONS AND NUMBER OF ROWS IN EACH SECTION.

The task is easy for us because we have only one section.

```
// How Many sections will we have in the table view?
```

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1; // we will have only one section in our table entries.
}
```

```
// How many rows in each section?
```

```
- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    return [arrayFlowerNames count];
}
```



4D: POPULATING THE TABLE VIEW. WE CAN USE ONE OF THE DELEGATED METHODS OF THE TABLE VIEW CLASS TO POPULATE THE CELLS OF EACH ROW.

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @“theCell”;

    // Let us get a pointer to a new UITableViewCell for reusable purpose.

    UITableViewCell *theCell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

    // if it is nil, it must be a new cell. So, let us initialize it with default
    // style and autorelease feature. In case of a large table view, when the user scrolls
    // up or down, we would like to reuse the cell.

    if (theCell == nil) {
        theCell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier];
    }

    // Configure the cell.

    theCell.textLabel.text =[arrayFlowerNames objectAtIndex:indexPath.row];

    // You can assign the cell.accessory type to a Disclosure symbol, as shown below:
    theCell.accessoryType = UITableViewAccessoryNone;
    return theCell;
}
```

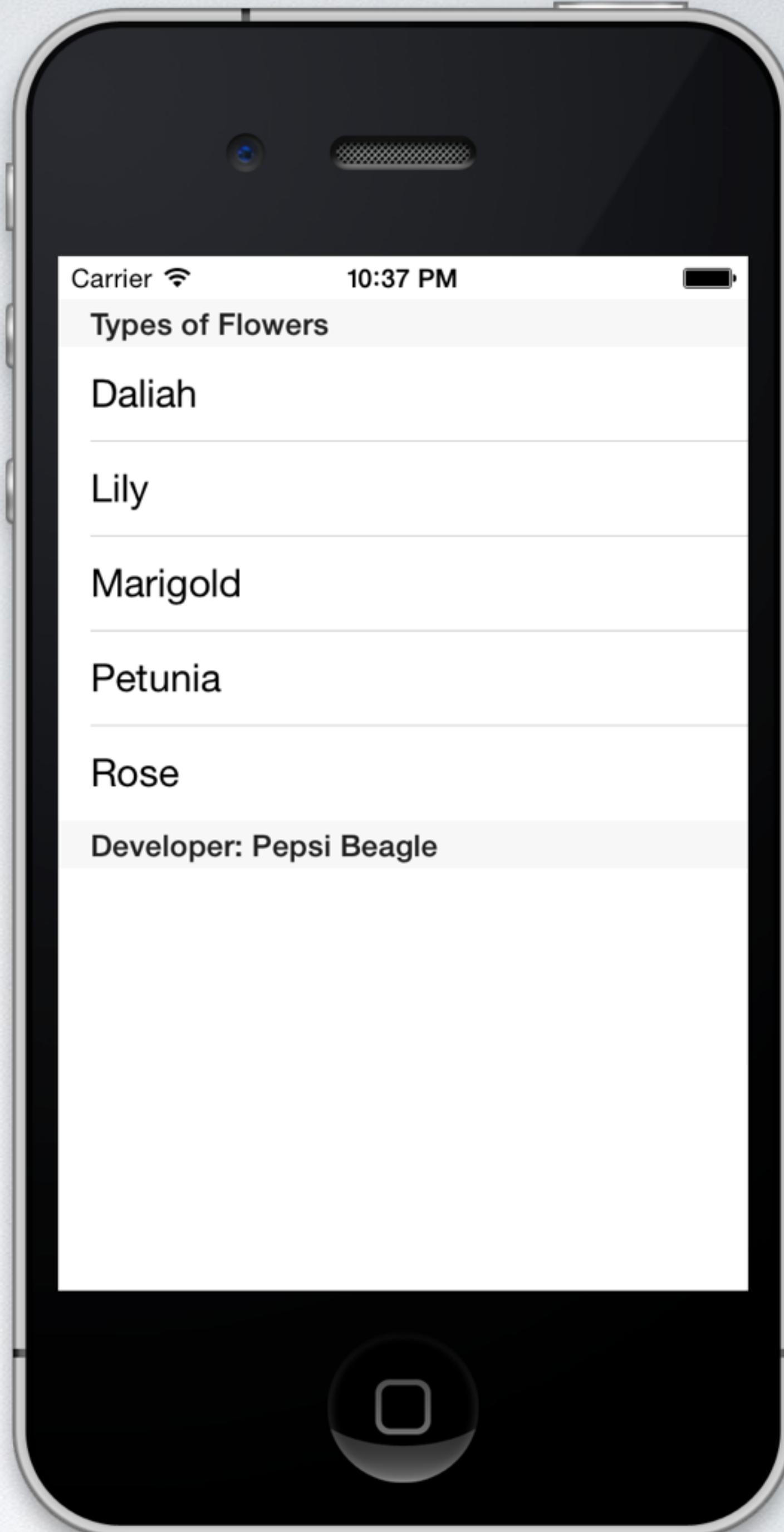


4E: OPTIONAL HEADER AND FOOTER TITLE OF SECTIONS: YOU MAY IMPLEMENT THE FOLLOWING METHODS FOR THE HEADER AND FOOTER TITLES

```
// Implement the following delegated method to display a header for a section.  
- (NSString *)tableView:(UITableView *)tableView  
titleForHeaderInSection:(NSInteger)section  
{  
    return @"Types of Flowers";  
}  
  
// Implement the following delegated method to display a footer for a section.  
- (NSString *) tableView:(UITableView *)tableView  
titleForFooterInSection:(NSInteger)section  
{  
    return @"Developer: Pepsi Beagle";  
}
```



YOU ARE DONE!



Don Miller

don@GroundSpeedHQ.com

@donmiller



GroundSpeed™
rapid web + mobile software