Crunch Time

Featuring NSTimers and Model-View-Controller!

Made with Xcode 7.1 and iOS 9

By Bliss Chapman

Welcome

- CocoaNuts is a group of “love of the game” iOS developers who meet weekly to discuss iOS technologies and the app development process.

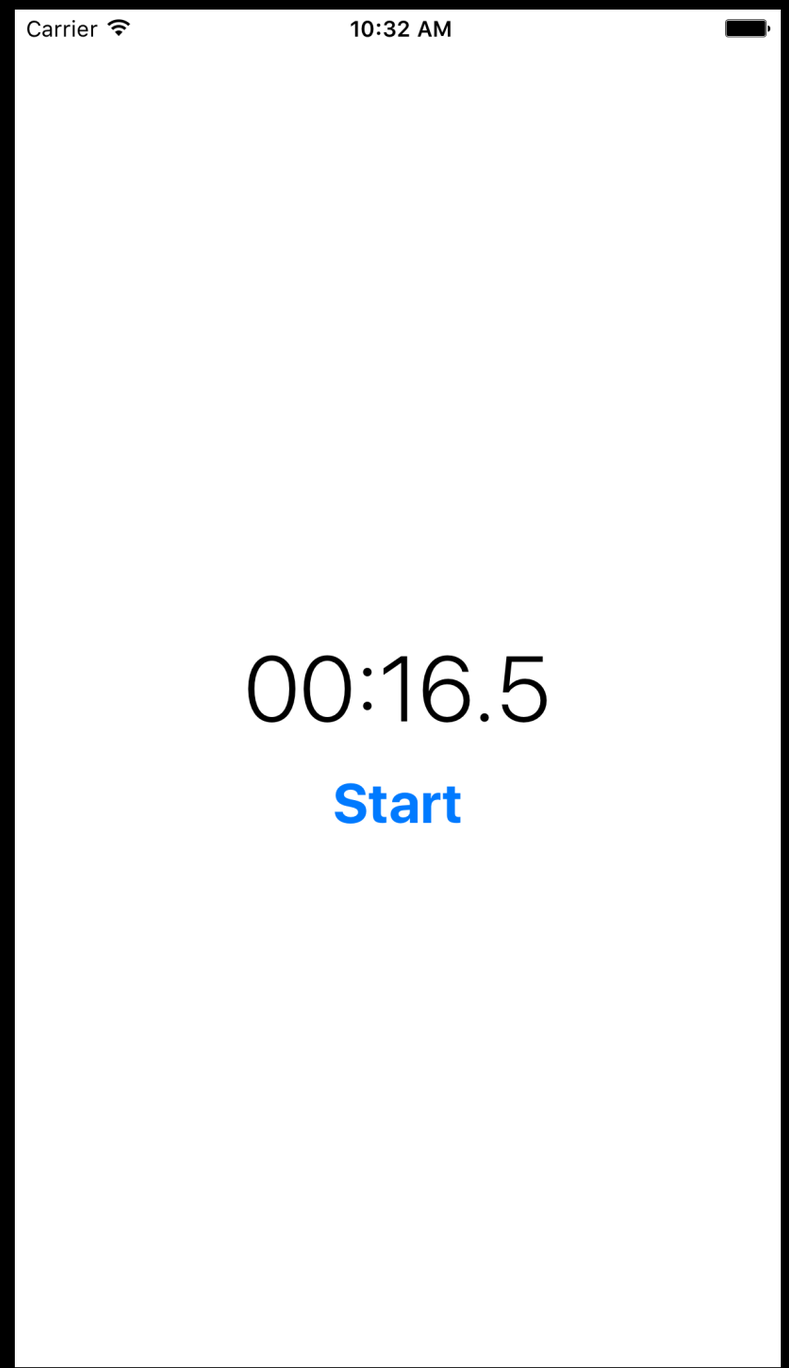
- If you are a beginner, please do not feel overwhelmed! You are in the right place and we are here to help you.

- As you make your way through the demo, focus on familiarizing yourself with Xcode and patterns in the app development process, ask lots of questions, and most importantly have fun!

Crunch Time

- As we approach the end of the semester, procrastination and Netflix binge-watching is inevitable.

- Luckily, today you will build a simple stop watch app to make your procrastination process more efficient.

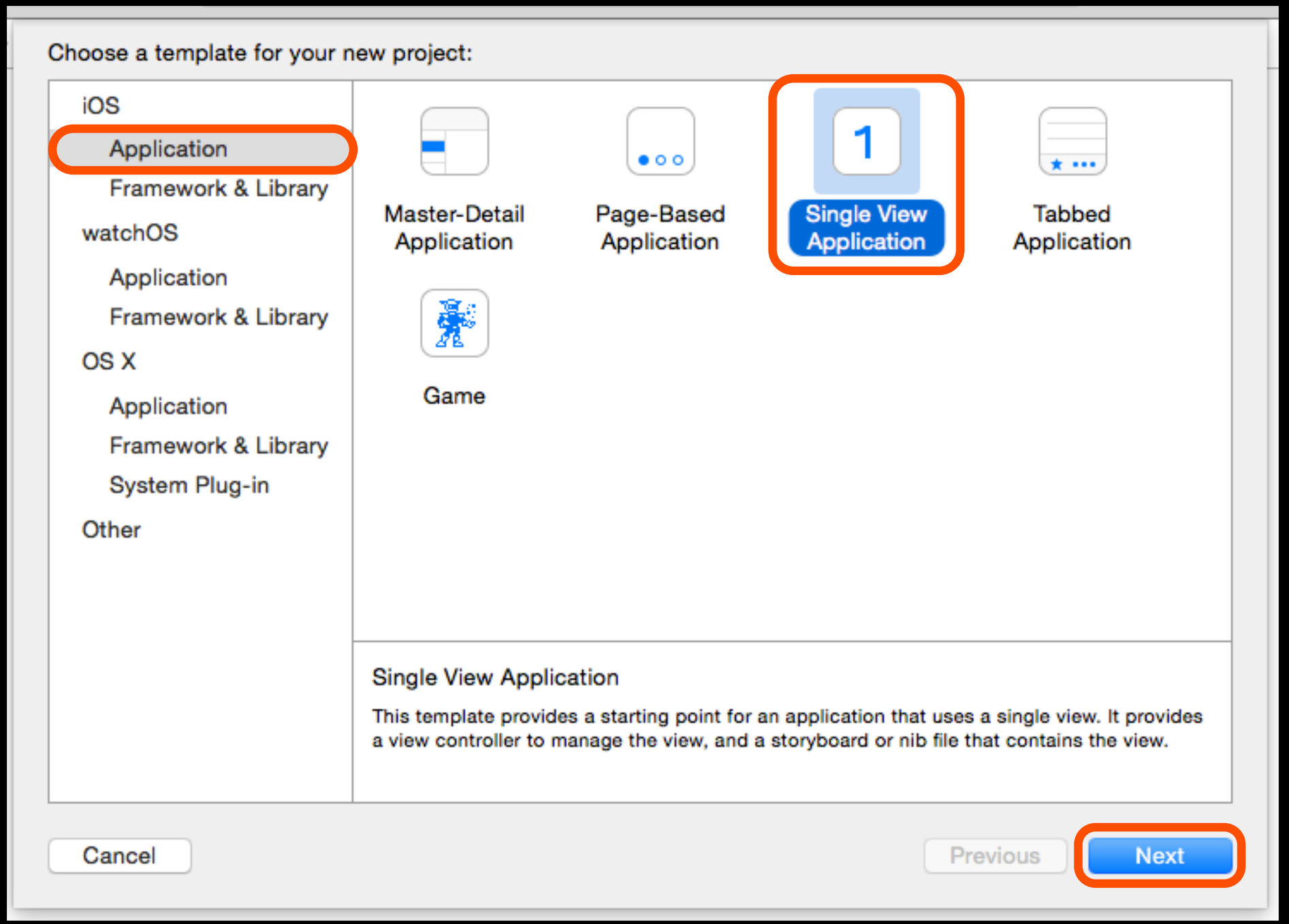


Demo

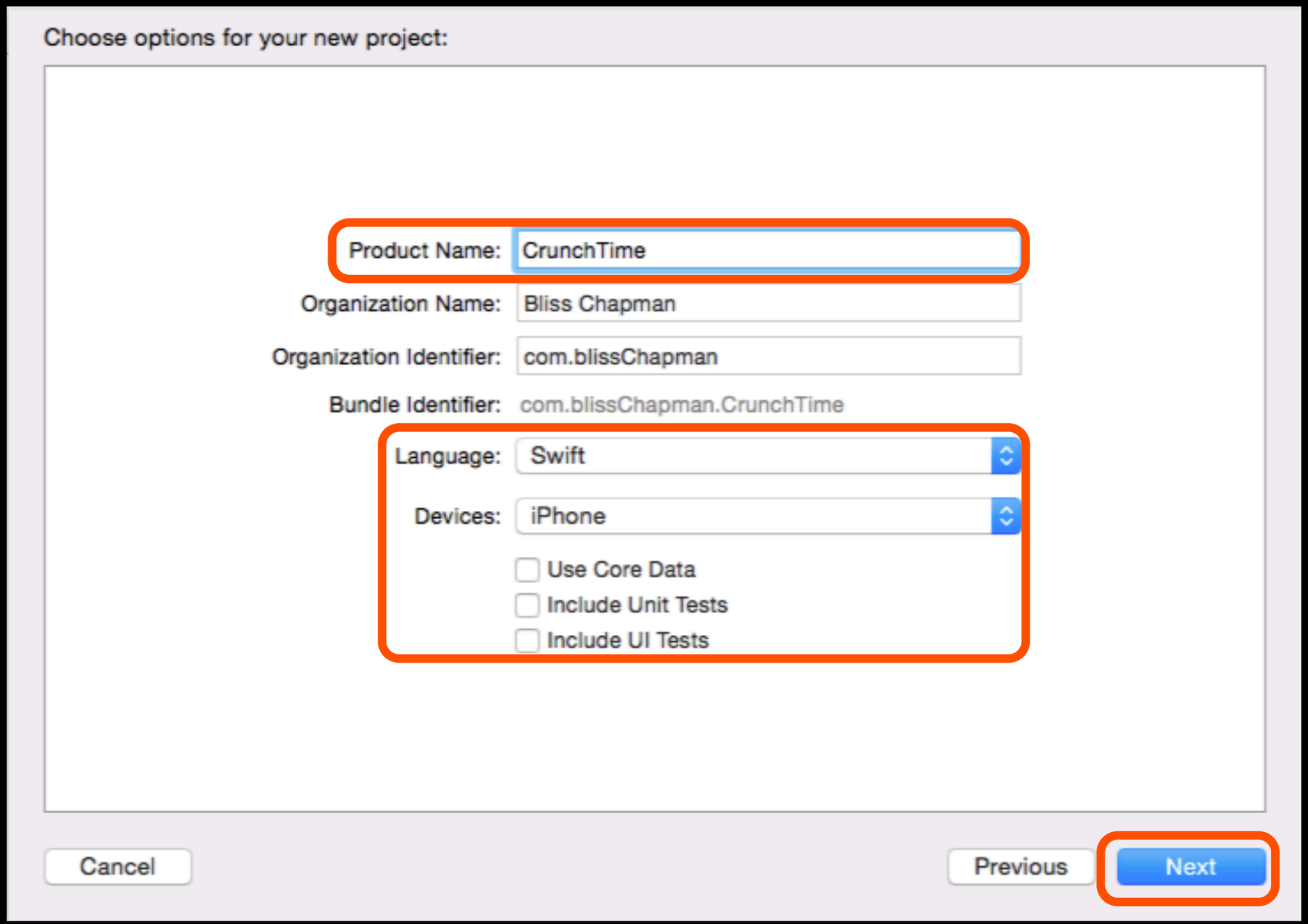
Launch Xcode.

Xcode is an integrated development environment (IDE) which contains all the tools you need to build an iOS, Mac, or Watch app.

Create a new project.



Select the template for this project.



Select the options shown above and then create your project.

CrunchTime 〉 iPhone 6s Plus | Indexing | Processing files

CrunchTime

- CrunchTime
  - CrunchTime
    - AppDelegate.swift
    - ViewController.swift
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
  - Products

CrunchTime | General | Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

Identity

Bundle Identifier: com.blissChapman.CrunchTime
Version: 1.0
Build: 1
Team: None

Deployment Info

Deployment Target: 9.1
Devices: iPhone
Main Interface: Main
Device Orientation: Portrait, Upside Down, Landscape Left, Landscape Right
Status Bar Style: Default
Hide status bar
Requires full screen

App Icons and Launch Images

App Icons Source: AppIcon
Launch Images Source: Use Asset Catalog
Launch Screen File: LaunchScreen

Embedded Binaries

Add embedded binaries here

Linked Frameworks and Libraries

Name | Status

Add frameworks & libraries here

Quick Help

No Quick Help

No Matches

CrunchTime 〉 iPhone 6s Plus

CrunchTime: **Ready** | Today at 10:40 PM

CrunchTime 〉 CrunchTime 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 No Selection

- CrunchTime M
  - CrunchTime
    - AppDelegate.swift ?
    - ViewController.swift ?
    - Main.storyboard A
    - Assets.xcassets ?
    - LaunchScreen.storyboard ?
    - Info.plist ?
  - Products

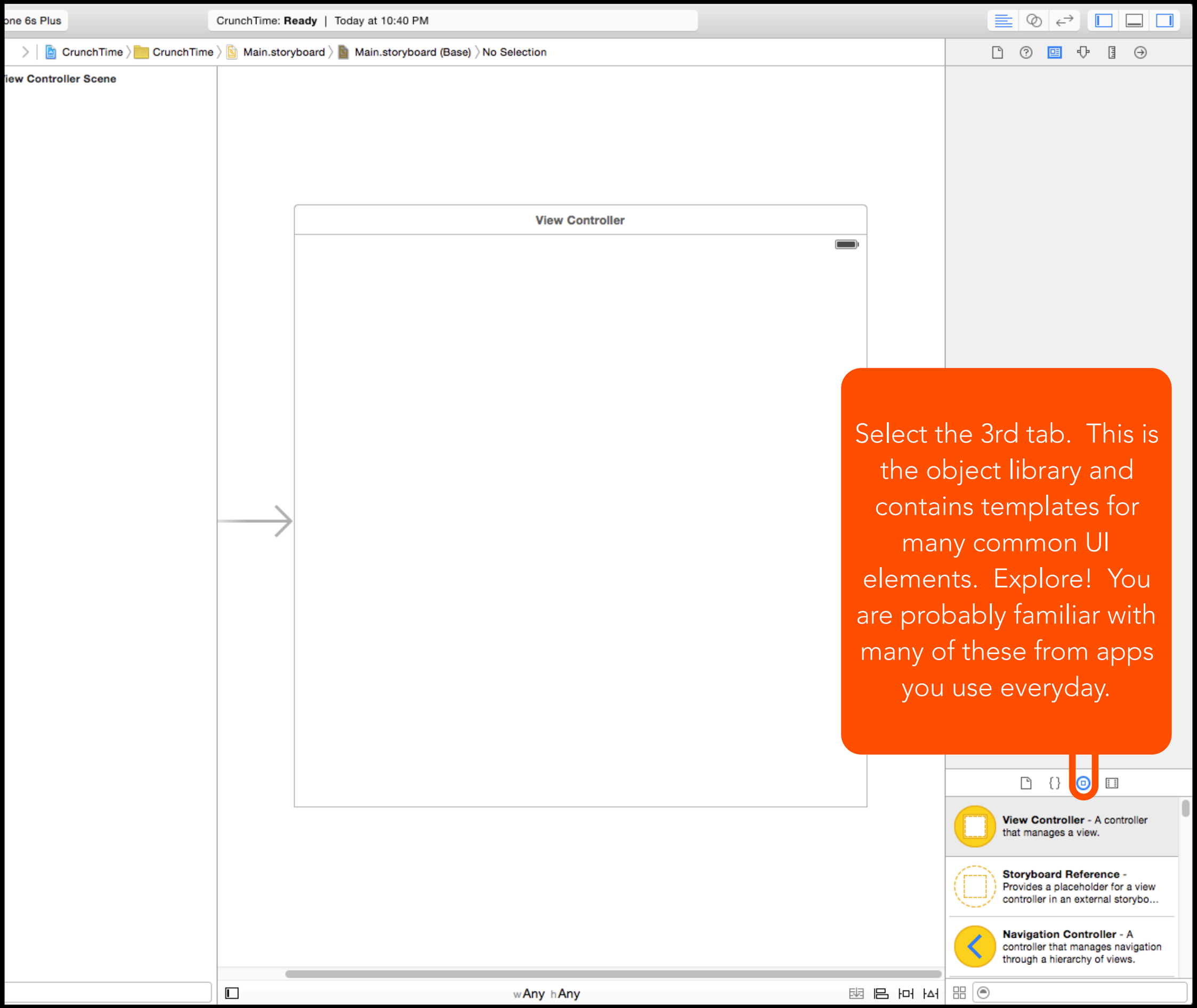
1

View Controller Scene

View Controller

This is the Storyboard, where the user interface is created.

wAny hAny


one 6s Plus
CrunchTime: Ready | Today at 10:40 PM
CrunchTime
CrunchTime
Main.storyboard
Main.storyboard (Base)
No Selection
iew Controller Scene
View Controller
Select the 3rd tab. This is the object library and contains templates for many common UI elements. Explore! You are probably familiar with many of these from apps you use everyday.
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.
wAny hAny

CrunchTime 〉 CrunchTime 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View

- View Controller Scene
  - View Controller
    - Top Layout Guide
    - Bottom Layout Guide
    - View
      - Label
    - First Responder
    - Exit
    - Storyboard Entry Point

Drag a label onto the canvas.

wAny hAny

Custom Class
Class: UIView
Module: None

Identity
Restoration ID:

User Defined Runtime Attributes
Key Path | Type | Value

Document
Label: Xcode Specific Label
Object ID: 8bC-Xf-vdC
Lock: Inherited - (Nothing)
Notes:
No Font

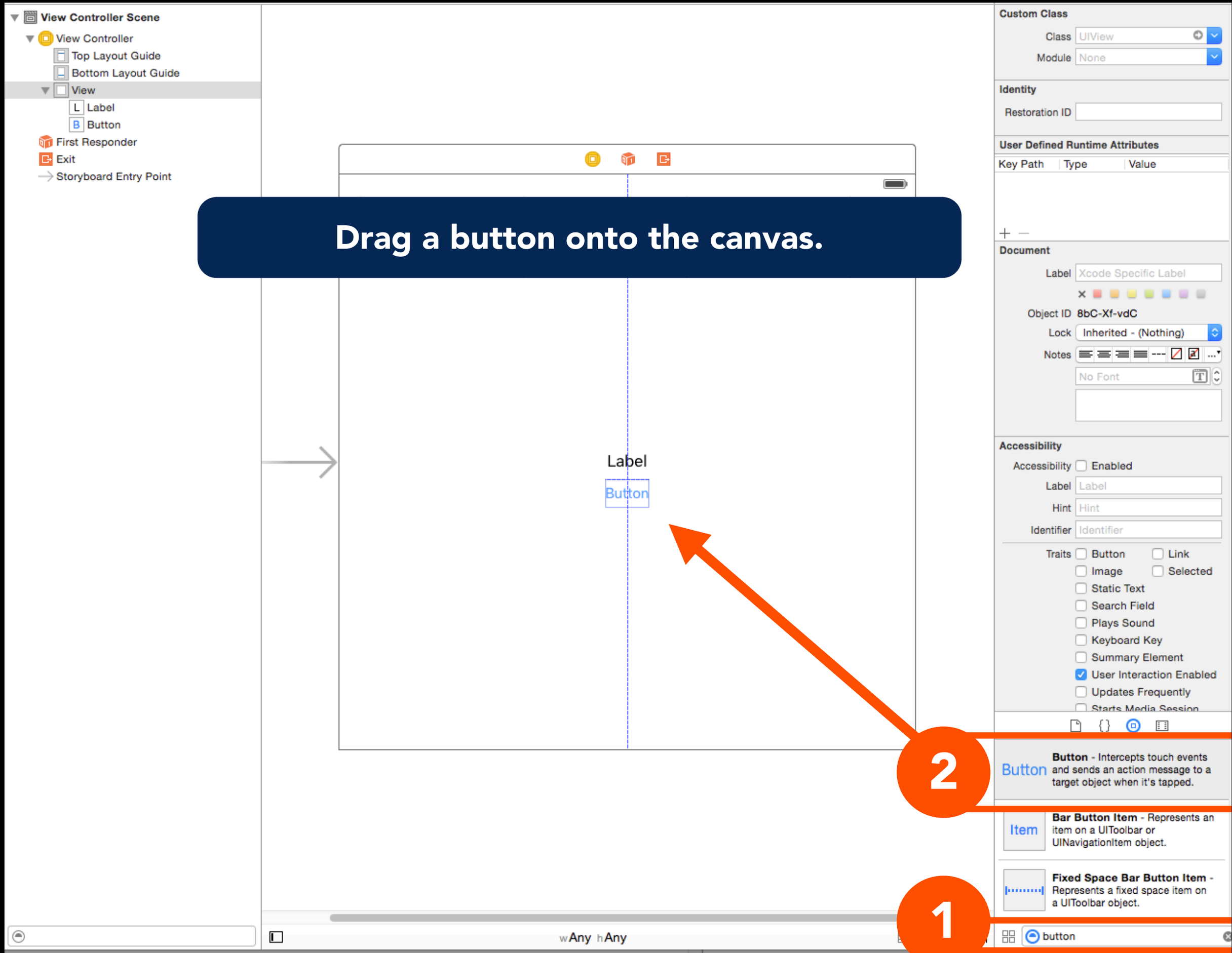
Accessibility
Accessibility: Enabled
Label: Label
Hint: Hint
Identifier: Identifier
Traits: Button, Link, Image, Selected, Static Text, Search Field, Plays Sound, Keyboard Key, Summary Element, User Interaction Enabled, Updates Frequently, Starts Media Session

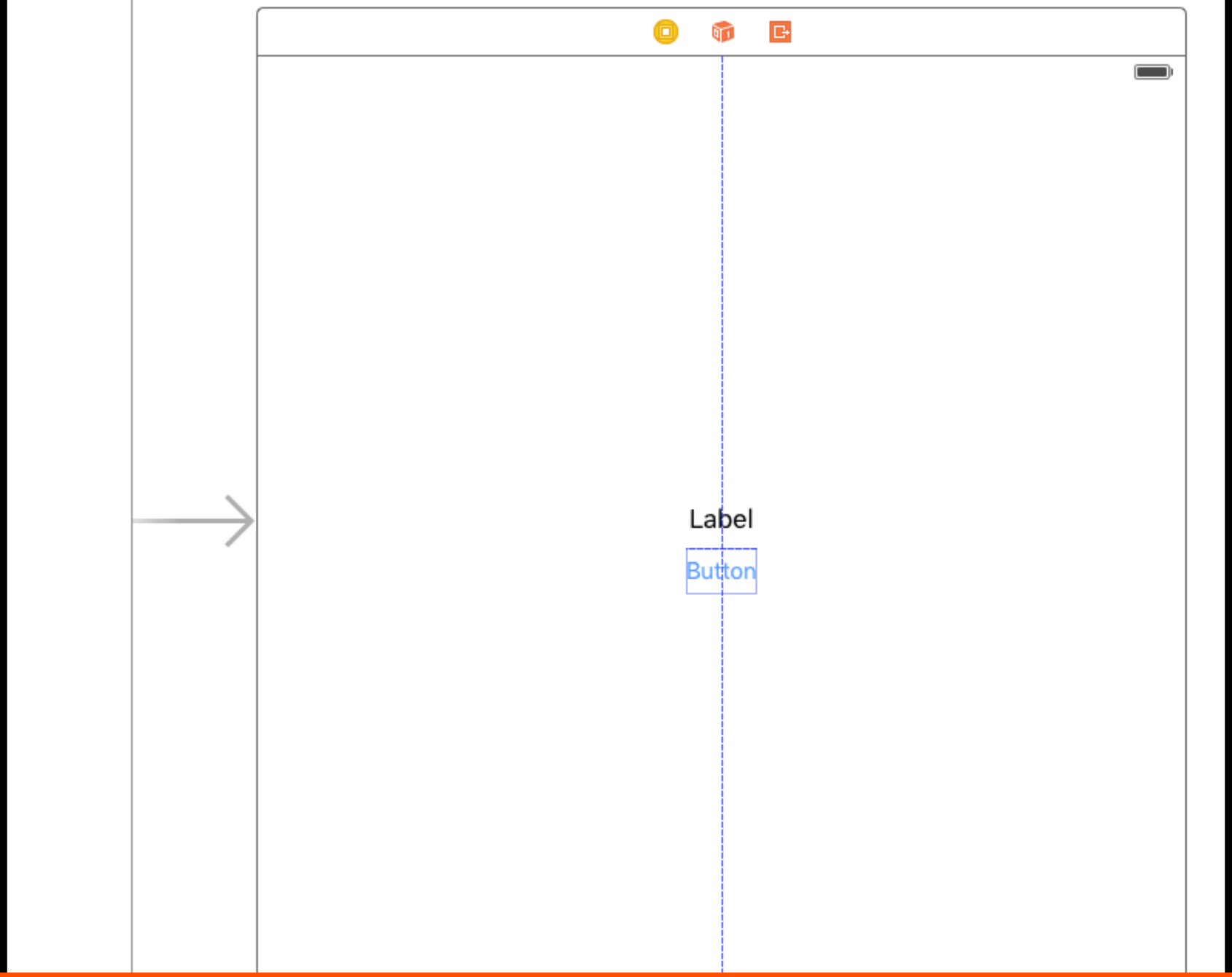
Label **Label** - A variably sized amount of static text.

label

2

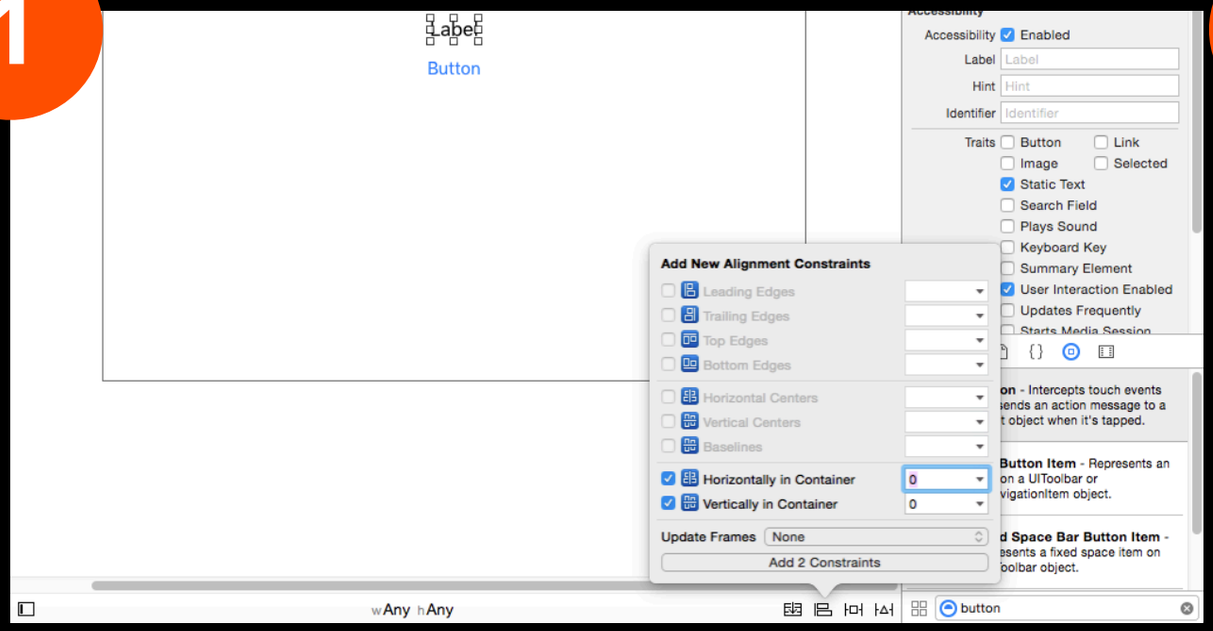
1


View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
Label
Button
First Responder
Exit
Storyboard Entry Point
Drag a button onto the canvas.
Label
Button
wAny hAny
Custom Class
Class UIView
Module None
Identity
Restoration ID
User Defined Runtime Attributes
Key Path Type Value
Document
Label Xcode Specific Label
Object ID 8bC-Xf-vdC
Lock Inherited - (Nothing)
Notes
No Font
Accessibility
Accessibility Enabled
Label Label
Hint Hint
Identifier Identifier
Traits Button Link Image Selected Static Text Search Field Plays Sound Keyboard Key Summary Element User Interaction Enabled Updates Frequently Starts Media Session
Button
Button - Intercepts touch events and sends an action message to a target object when it's tapped.
Item
Bar Button Item - Represents an item on a UIToolbar or UINavigationItem object.
Fixed Space Bar Button Item - Represents a fixed space item on a UIToolbar object.
button
2
1



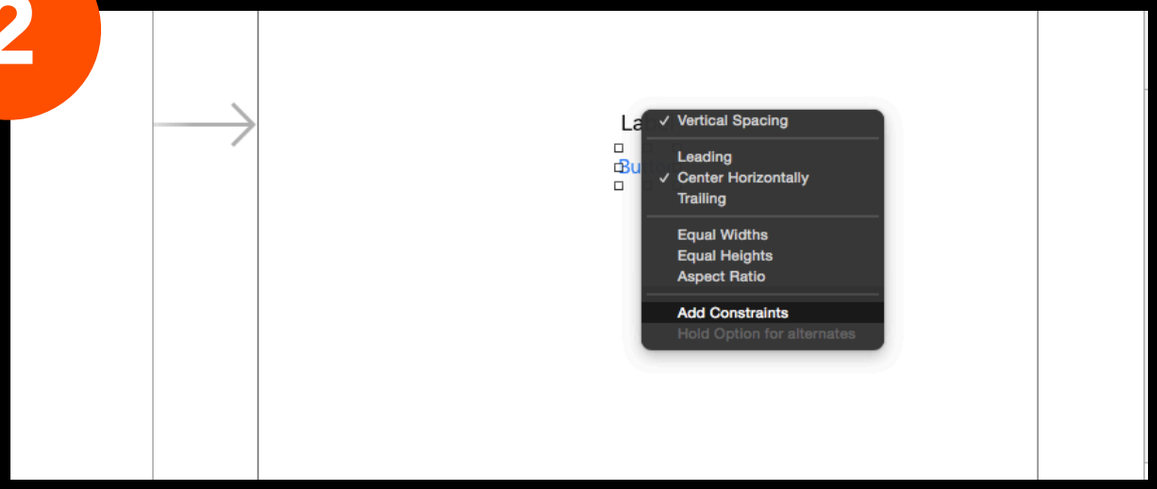
Notice how this canvas does not represent any particular form factor? This is because as developers we should design our user interface to be flexible to all screen sizes. We do this with a powerful technology called “autolayout.” Autolayout is used by creating rules or "constraints" that establish an item's position and size relative to another item or the view iteself.

1



Select the label then click the align button to add two new constraints that inform the label to always be in the horizontal and vertical center of the screen when the app is launched. Now, the label will be in the center of the view regardless of the device's size or orientation.

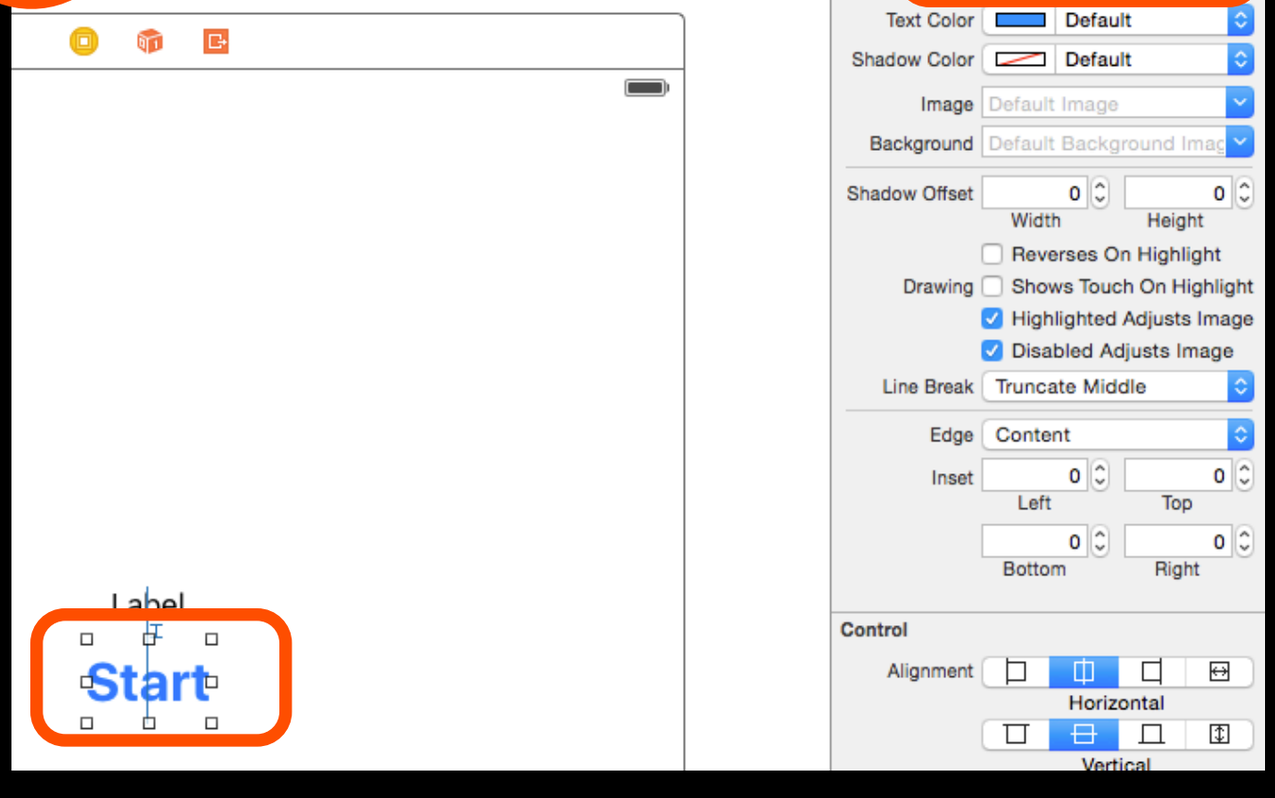
2



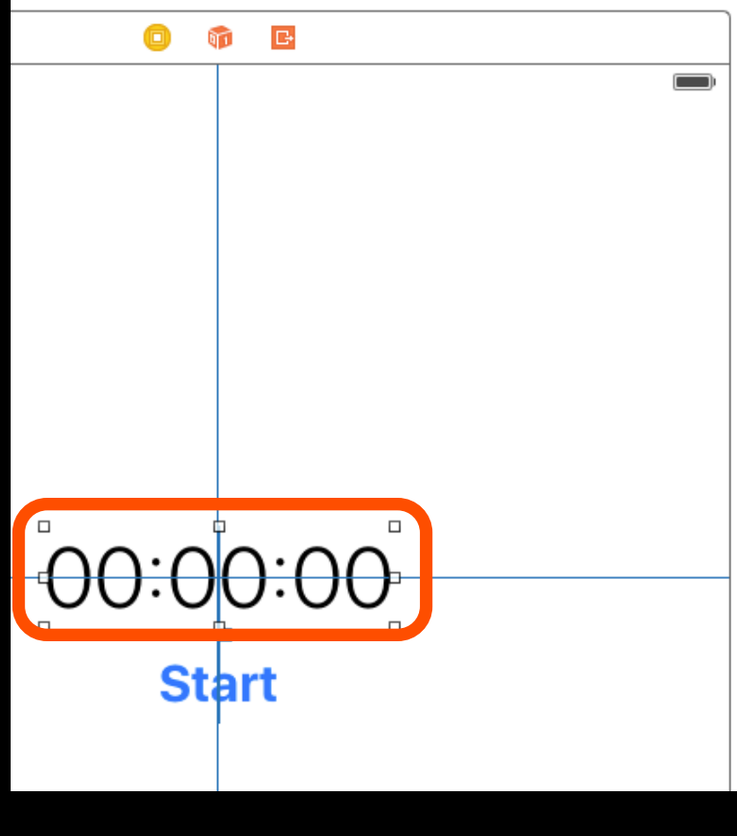
Next, hold down control and click and drag from the button to the label. This should trigger the popup shown above. Then while holding shift, select the Vertical Spacing and Center Horizontally options. Finally, select "Add Contraints."

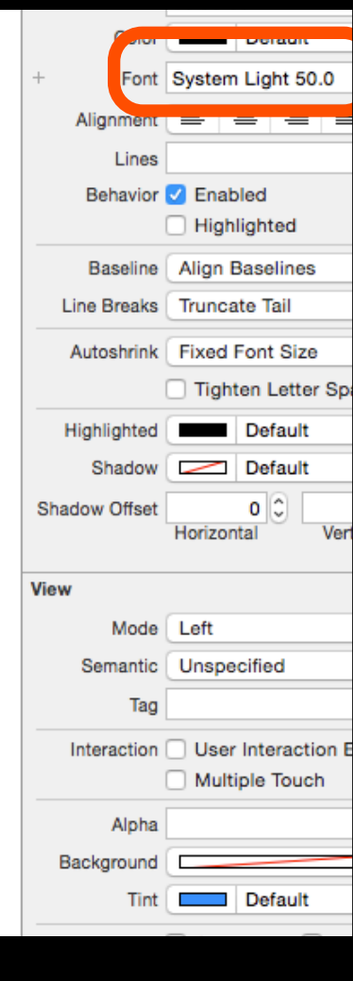
Add Constraints.

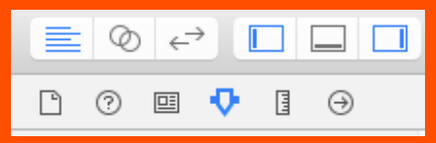
1



2





Next, lets customize our UI (user interface). First, double-click on the button to change it's title to "Start". Then, open the attributes inspector 

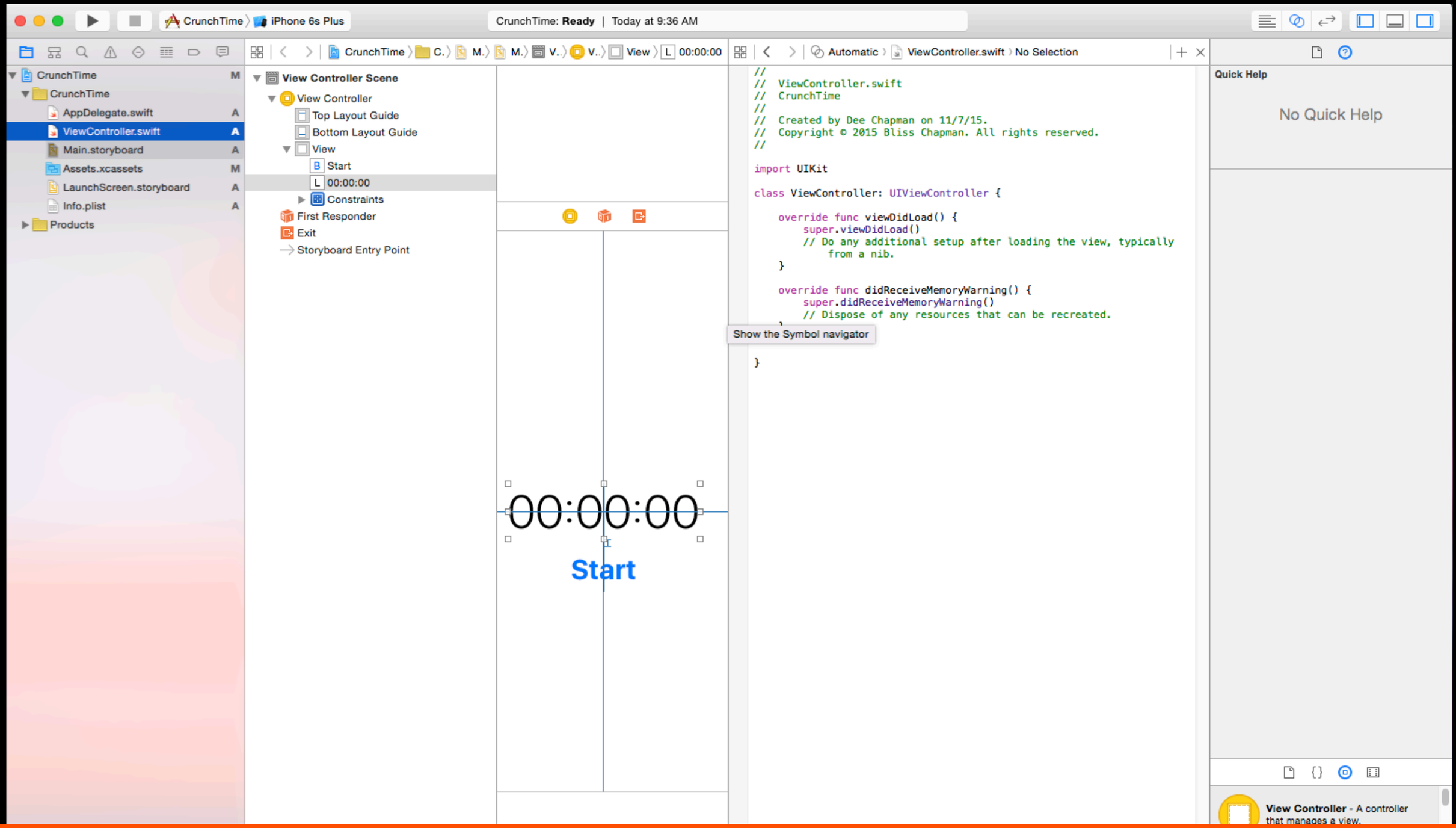
and adjust the font to be 30.0 and System Bold. Repeat this process for the label, changing the text to "00:00:00" and System Light 50.0.

Customize!

Code!

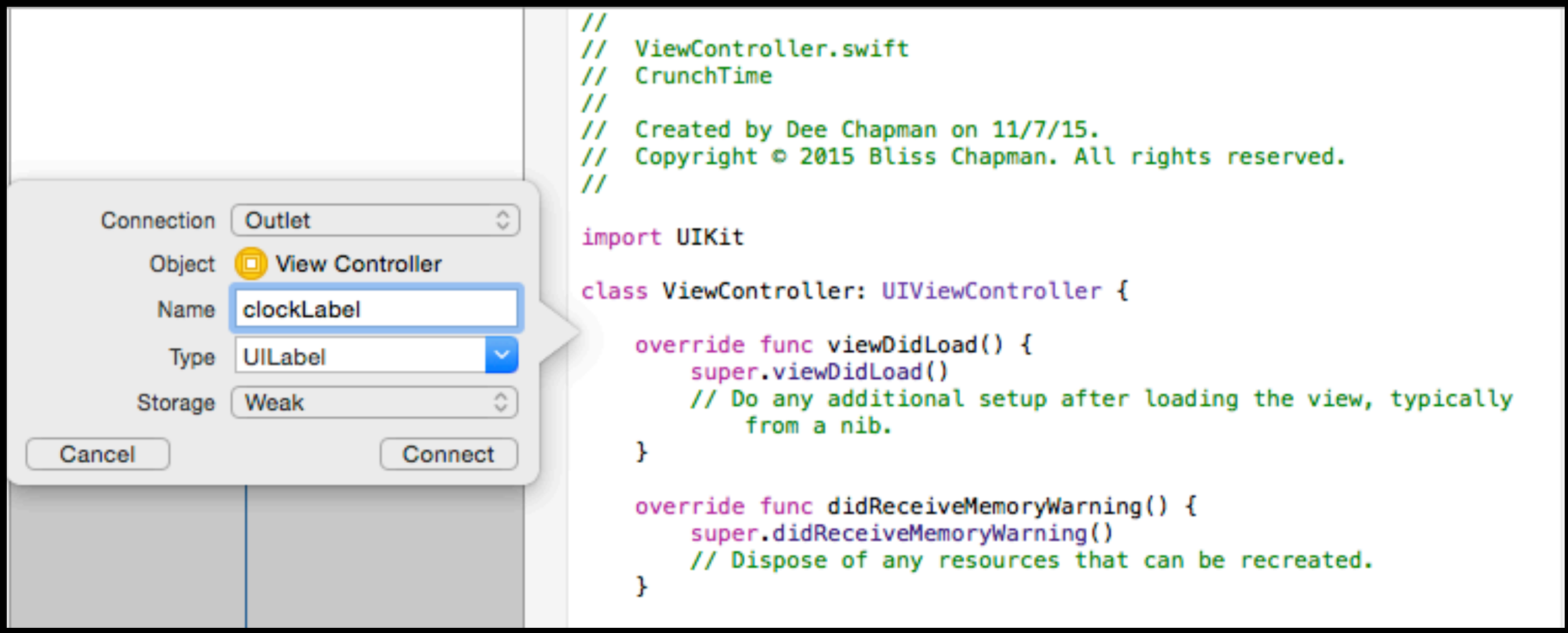
- Now we are finally ready to write some code! Our entire user interface has been set up however it doesn't DO anything yet. Try it out by clicking the run button in the upper right. 

- Before we close our storyboard, we need some way to access the properties of the button and label we dragged out and to detect when the button was tapped. We do this by creating outlets and actions.



While holding down option, select the ViewController.swift file in the project navigator to open Assistant Editor mode.

Open Assistant Editor



While holding down control, drag from the label in to the space below the class declaration and above the viewDidLoad method. Name this outlet "clockLabel" and then hit connect.

Creat a label outlet

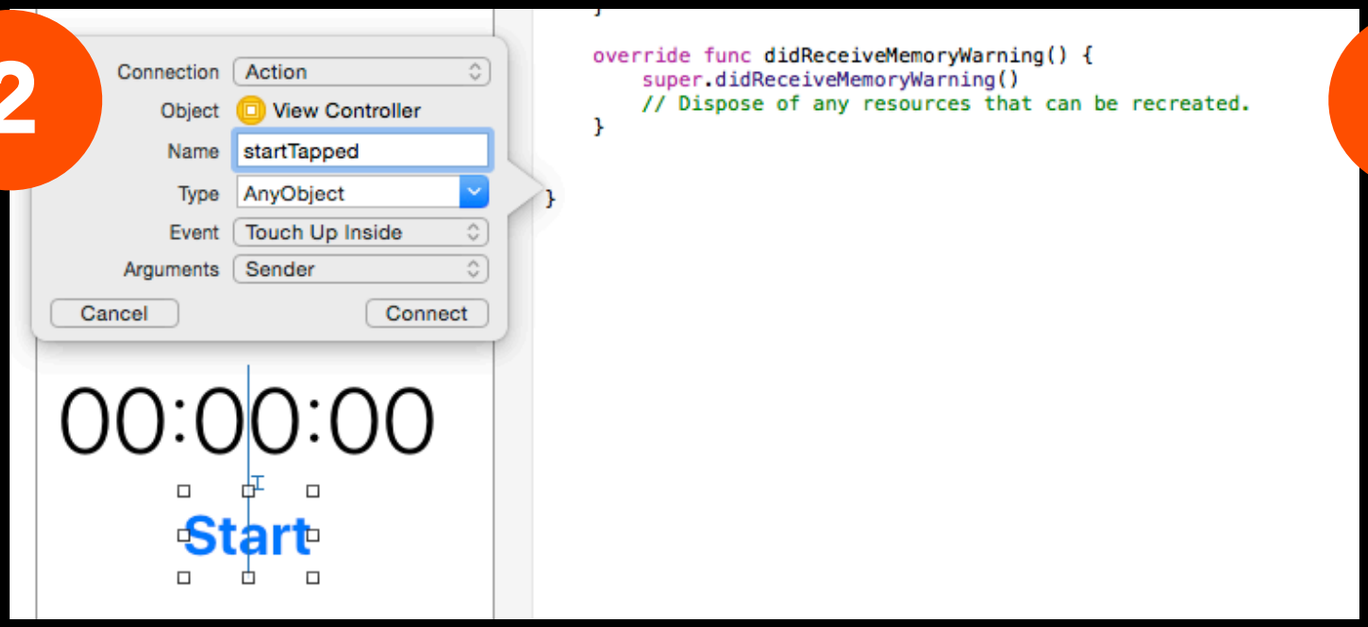
Repeat this process to make a button outlet named "startButton".

1

```
lass ViewController: UIViewController {

    @IBOutlet weak var clockLabel: UILabel!
    @IBOutlet weak var startButton: UIButton!
```

Finally, our code will need some way to detect when the button was tapped. Control drag again from the button in to the bottom of the class declaration (under the view controller lifecycle methods). Create a new Action and name it "startTapped".

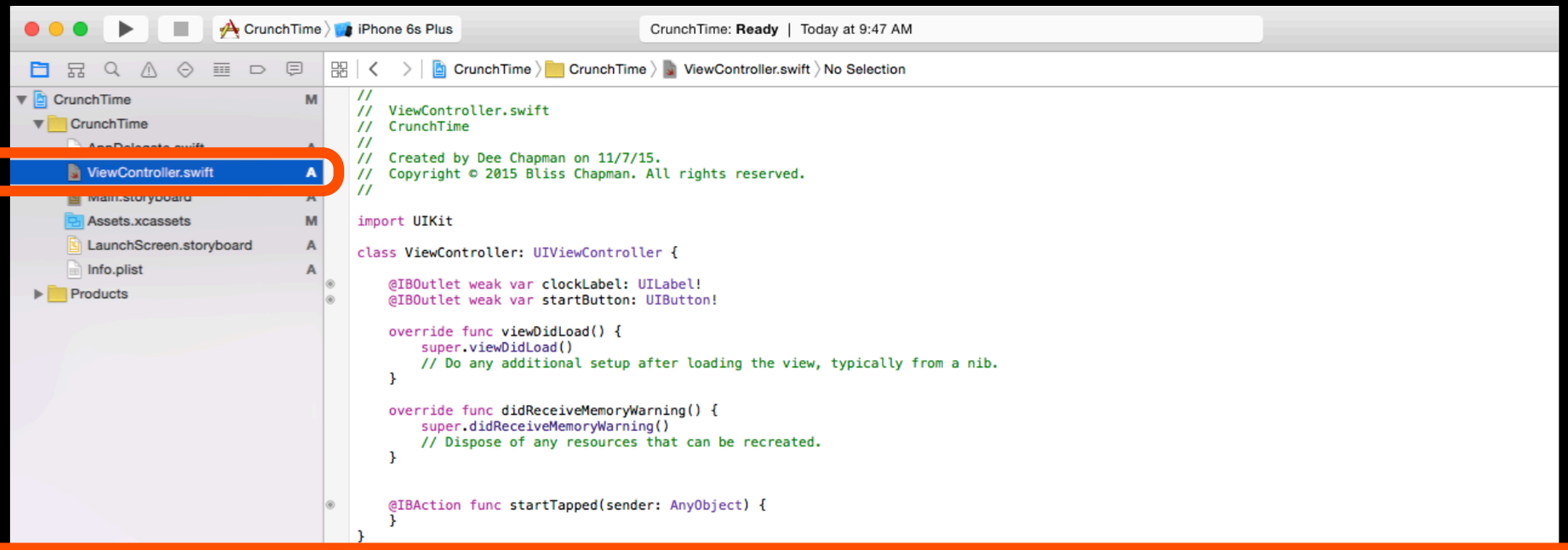
2



3

```
@IBAction func startTapped(sender: AnyObject) {
}
```

Create a button outlet and action



In the "model view controller" (MVC) design paradigm, this is the controller. It's role is to interpret the model and display it in the view. Essentially, this is just the code that controls our screen.

If we implement the stopwatch functionality with good coding style, we should really create a model that acts as a stopwatch. This stopwatch object should have all the functionality are users expect (starting, stopping, and current time) and our ViewController can remain lean and mean, leveraging this stopwatch object for all the difficult work.

This is good "object-oriented programming" because if we ever need to use this stopwatch somewhere else, we have a neatly defined object that can be dragged and dropped into different projects or even used in mulitple places within the same app. Even if it's never reused, all the data and behavior associated with the stopwatch will be in ONE place making for cleaner, more readable code.

If this is confusing to you, don't worry! This is tricky to understand and will come with time and practice. For now, ask lots of questions and don't forget to have fun!

1 First, create a new instance of our (to be defined) SimpleStopwatch.

2 Next, we determine what SHOULD happen when the start button is tapped. If the clock is NOT running:

- schedule a timer to continuously update our label with the value of the clock
- change the title of the button to "Stop"
- start the clock.

If it's already running:

- change the button title back to "Start"
- stop the stopwatch.

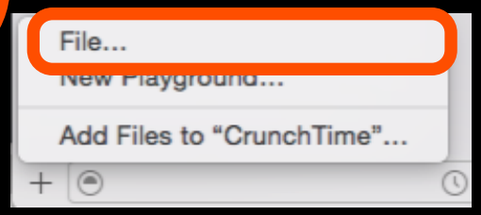
```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var clockLabel: UILabel!
    @IBOutlet weak var startButton: UIButton!

    let stopwatch = SimpleStopwatch()

    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.
    }


    @IBAction func startTapped(sender: AnyObject) {
        if !stopwatch.isRunning {
            NSTimer.scheduledTimerWithTimeInterval(0.1, target: self,
                selector: "updateClockLabel:", userInfo: nil, repeats: true)
            startButton.setTitle("Stop", forState: .Normal)
            stopwatch.start()
        } else {
            startButton.setTitle("Start", forState: .Normal)
            stopwatch.stop()
        }
    }

    func updateClockLabel(timer: NSTimer) {
        if stopwatch.isRunning {
            clockLabel.text = stopwatch.elapsedTimeAsString
        } else {
            timer.invalidate()
        }
    }
}
```

3 Finally, write an updateClockLabel that will be called every tenth of a second to update the clockLabel with the "elapsedTimeAsString" property of the stopwatch. If the stopwatch is not running, this method will stop the timer.

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var clockLabel: UILabel!
    @IBOutlet weak var startButton: UIButton!

    let stopwatch = SimpleStopwatch()

    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.
    }


    @IBAction func startTapped(sender: AnyObject) {
        if !stopwatch.isRunning {
            NSTimer.scheduledTimerWithTimeInterval(0.1, target: self,
                selector: "updateClockLabel:", userInfo: nil, repeats: true)
            startButton.setTitle("Stop", forState: .Normal)
            stopwatch.start()
        } else {
            startButton.setTitle("Start", forState: .Normal)
            stopwatch.stop()
        }
    }

    func updateClockLabel(timer: NSTimer) {
        if stopwatch.isRunning {
            clockLabel.text = stopwatch.elapsedTimeAsString
        } else {
            timer.invalidate()
        }
    }
}
```

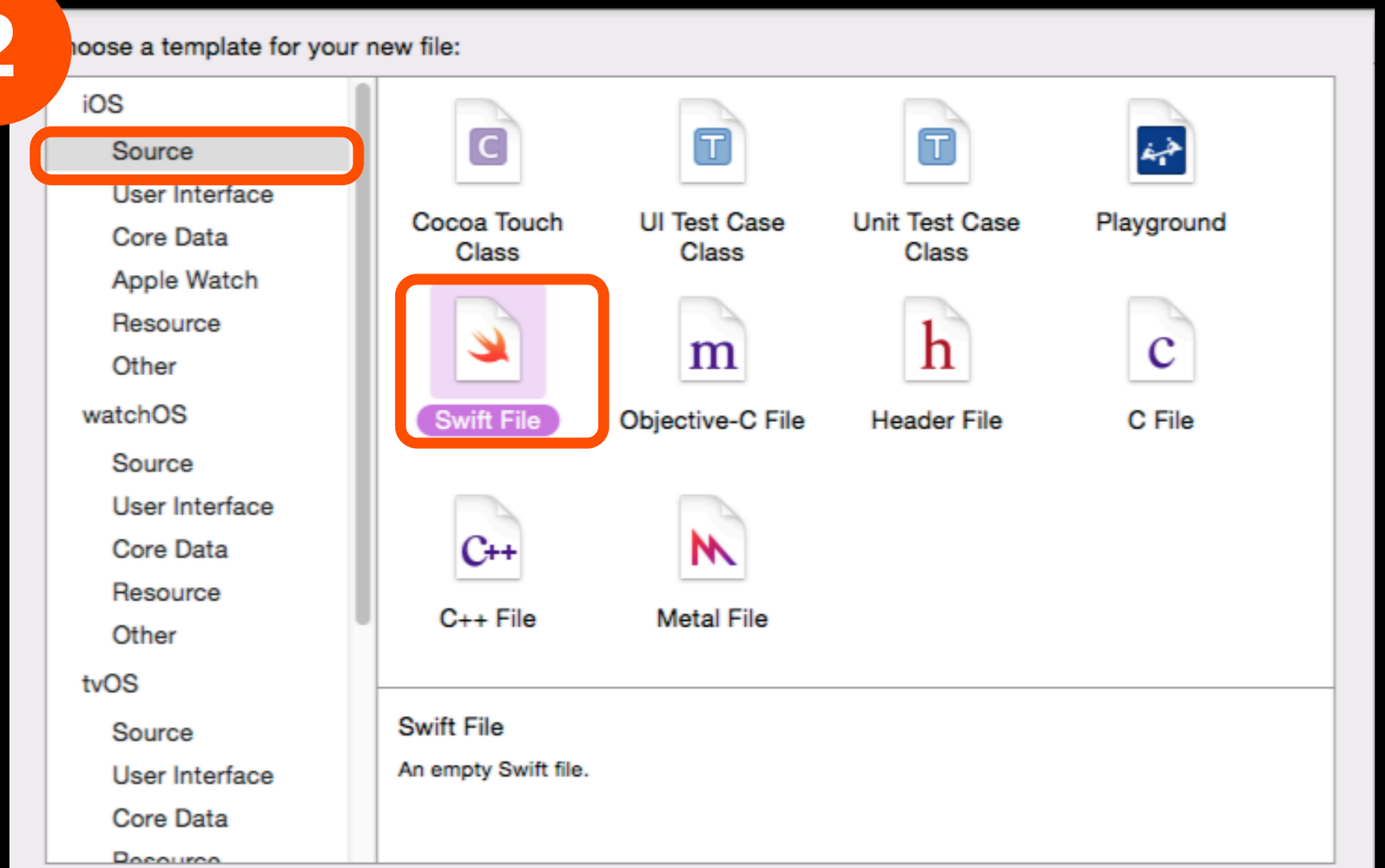
Notice how we have now established all the functionality our SimpleStopwatch object will have. We wrote our ViewController methods as if it already exists and now it's clear what behavior and data this object will need to contain.

1

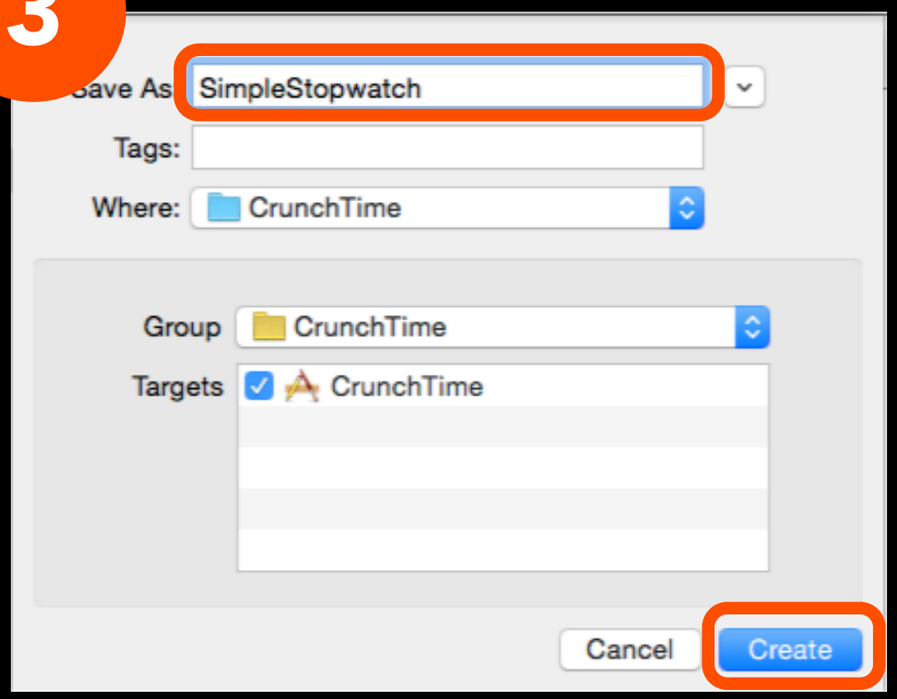


Add a new file by clicking the + button in the bottom left.

2



3



Create SimpleStopwatch file.

```swift
import Foundation

class SimpleStopwatch {

    private var startTime: NSDate?

    var elapsedTime: NSTimeInterval {
        guard let startTime = self.startTime else {
            return 0
        }

        return -startTime.timeIntervalSinceNow
    }

    var elapsedTimeAsString: String {
        return String(format: "%02d:%02d.%d",
            Int(elapsedTime / 60), Int(elapsedTime % 60), Int(elapsedTime * 10 % 10))
    }

    var isRunning: Bool {
        return startTime != nil
    }
```

Now we start filling out some of the properties of our stopwatch class. If you are unfamiliar with the idea of classes, you can think of them as a template or a blueprint for an "object." Any "instance" of a class (an object) will have all the data and behavior associated with that class. So any new SimpleStopwatch will have a startTime, an elapsedTime, an elapsedTimeAsString, and an isRunning property.

Notice how three of these properties are "computed properties" meaning their value is computed based on something else.

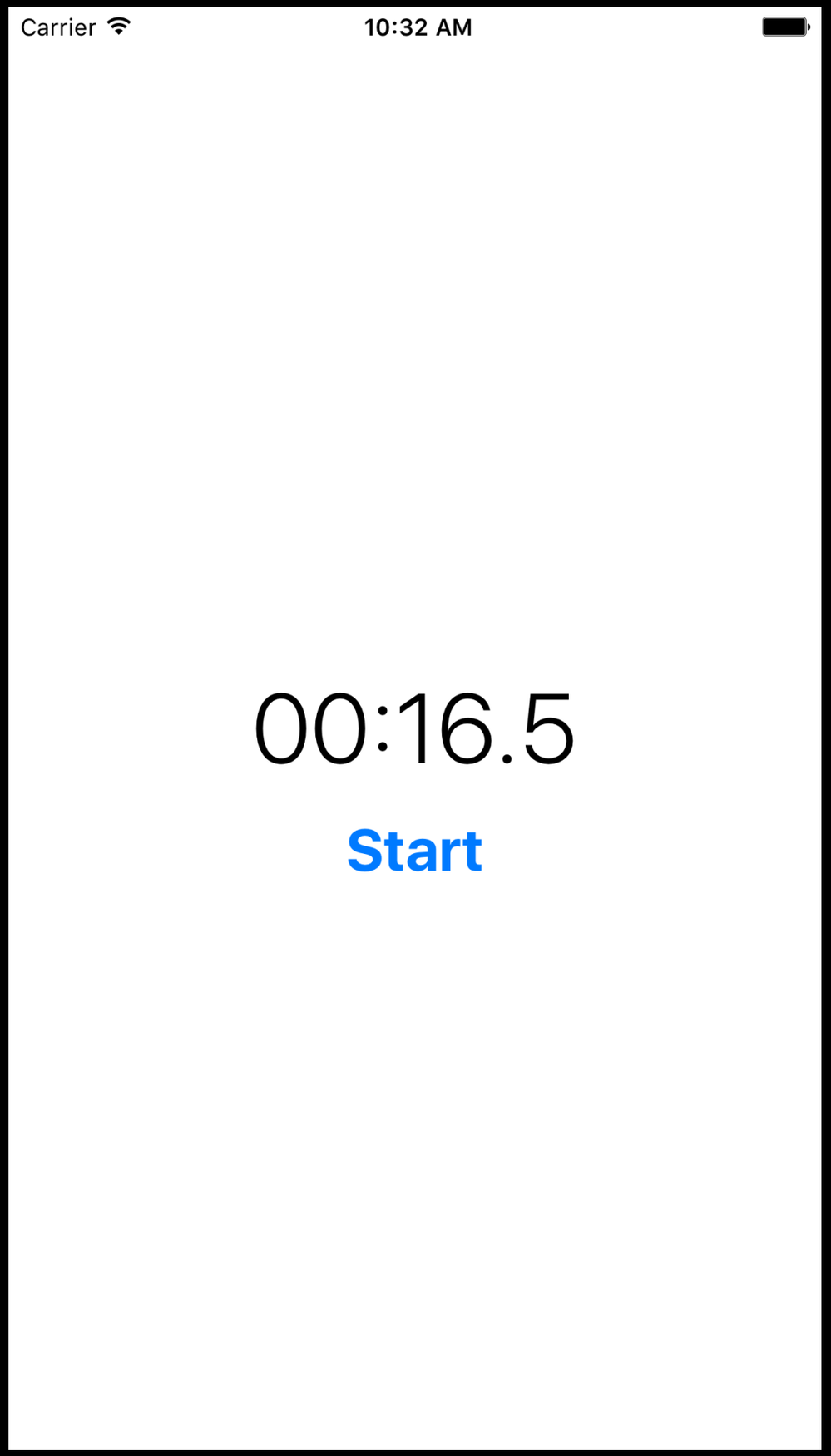
Create properties of SimpleStopwatch.

```
var isRunning: Bool {
    return startTime != nil
}
```

```
//MARK: Behavior
func start() {
    startTime = NSDate()
}

func stop() {
    startTime = nil
}
```

As a final step, our stopwatch needs to know how to start and stop. Implementing these methods couldn't be simpler! Now, any SimpleStopwatch object we create can start and stop and this means that we've finished defining all the behaviors our ViewController code required.

Define stopwatch behaviors.



Yay, now build and run to see your new procrastination helper in action!

Congratulations!

- Great job making it through those walls of text ☺️!
- Thank you so much for coming. Remember, if you have any questions please, please ask!
- There are lots of past demos on the CocoaNuts website for you to work through and many other fantastic resources we recommend to further you on your iOS development journey.
- Best of luck and I hope to see you next week!