

iOS
DeCal

lecture 10

UI/UX and Programmatic Design

cs198-001 : fall 2017

Announcements

Custom App progress form due before lab
(~1 minute)

Only 2 labs left (no lab next week)

Final Presentations: Friday of RRR at
3:30pm in HP Auditorium

If you have a conflict, let us know on
Piazza now

Wednesday's Lab

Markdown mini-lecture (how to style GitHub README files)

Meet with your assigned TA and go over your progress so far / discuss things you may need to change

iOS Human Interface Guidelines

Your “go-to” resource for best practices concerning correct usage / placement / properties of UI elements ([link](#))

The image shows three iPhone screens side-by-side, each displaying a different aspect of the iOS Human Interface Guidelines:

- Left Screen (Overview):** Shows the main navigation menu with sections like Overview, Design Principles, What's New in iOS 10, Interface Essentials, Interaction, Features, Visual Design, Graphics, UI Bars, UI Views, UI Controls, Extensions, Technologies, and Resources.
- Middle Screen (Home Screen):** Shows the iOS 10 home screen with various app icons (Mail, Calendar, Photos, Camera, Maps, Clock, Weather, News, Wallet, Notes, Reminders, Stocks, Videos, iBooks, iTunes Store, App Store, Home, Health) and a weather widget.
- Right Screen (Lyft App):** Shows a Lyft app interface with a map of San Francisco, a pickup location, and a destination to "San Francisco International Airport". It displays a message from SIRI suggesting a ride and provides a "Request" button.

iOS Human Interface Guidelines

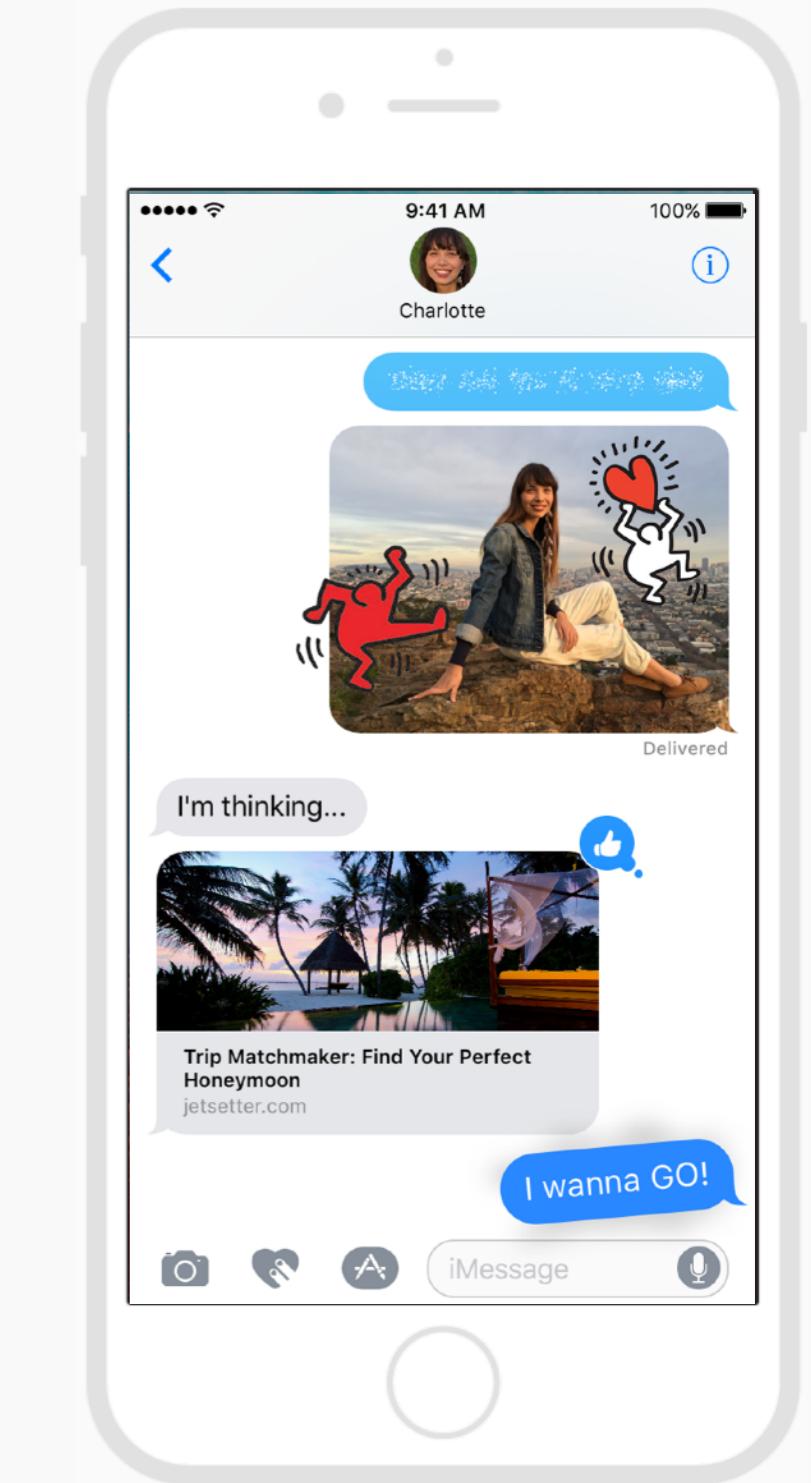
iMessage Integration

Enables you to implement a Messaging Extension for your app

Can share text, photo, stickers, interactive games (in-message!)

For iMessage Apps, be sure to have a distinct focus (should be relatively simple)

From the guidelines “Don’t try to design one app that combines both stickers and ridesharing, for example.”



iOS Human Interface Guidelines

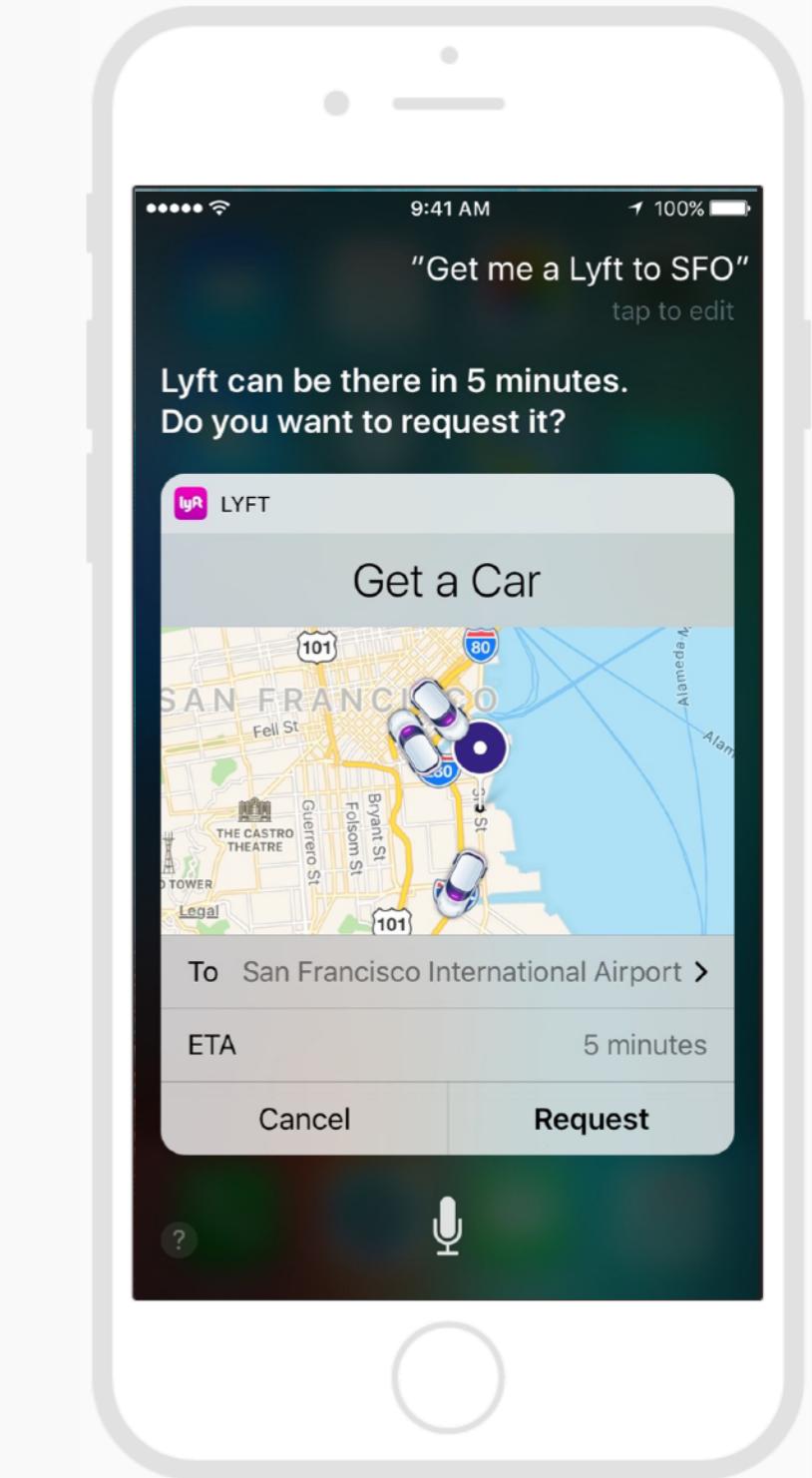
Integration with Siri

Allow users to access your app through voice controls

Can be useful for apps involving audio and video calling, messaging, payments, fitness, directions, etc.

For Siri-Enabled Apps, recommended to minimize interaction

Users expect a fast response (stay focused, don't provide more information than needed)

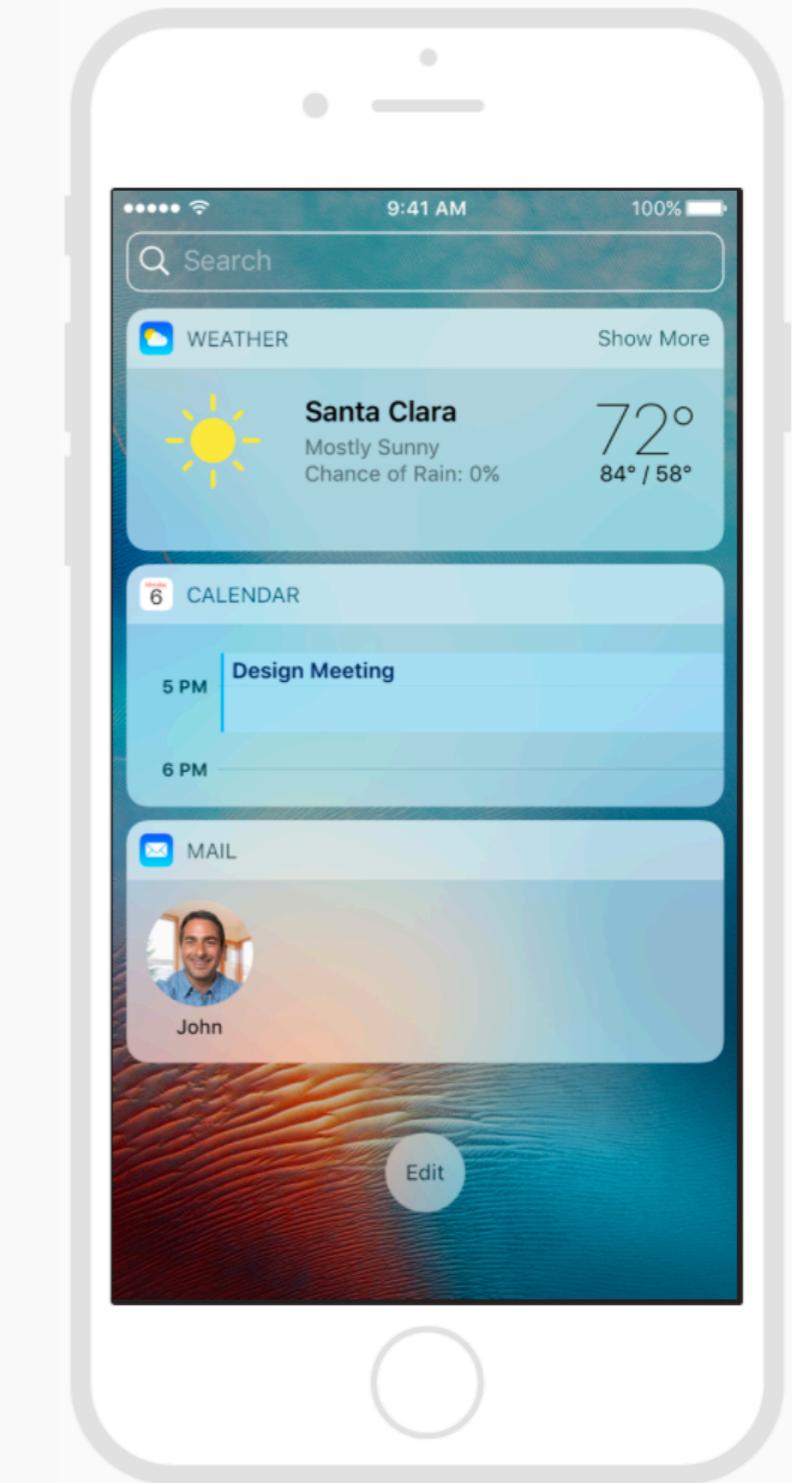


iOS Human Interface Guidelines

Search Widgets

Display notifications from your application on the user's Search and Home Screen

Very customizable (can add buttons, images, layout customization, etc.)



iOS Human Interface Guidelines

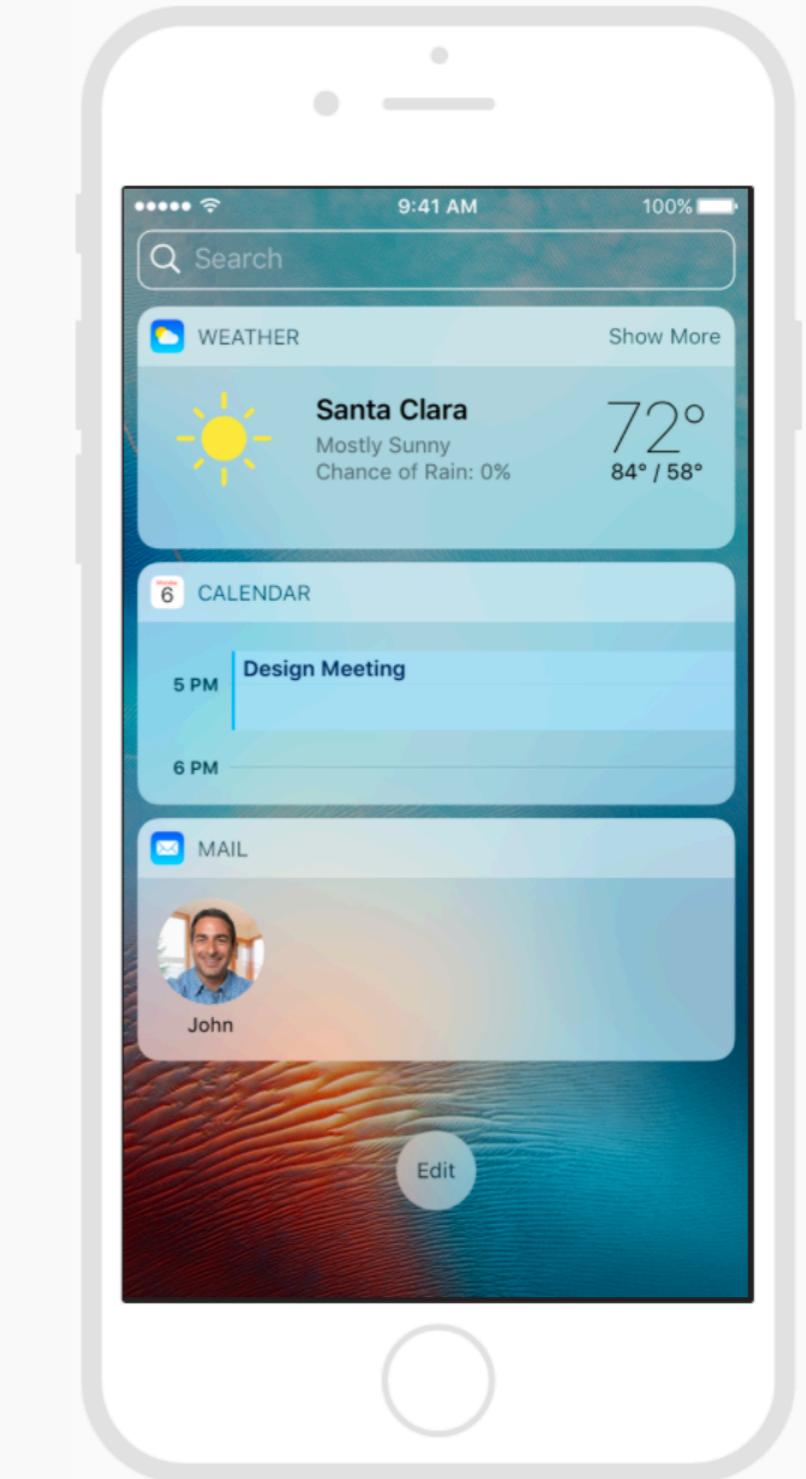
Search Widgets

Display notifications from your application on the user's Search and Home Screen

Very customizable (can add buttons, images, layout customization, etc.)

To view widgets

Search Screen > accessed by swiping to the right on Home or Lock Screen



iOS Human Interface Guidelines

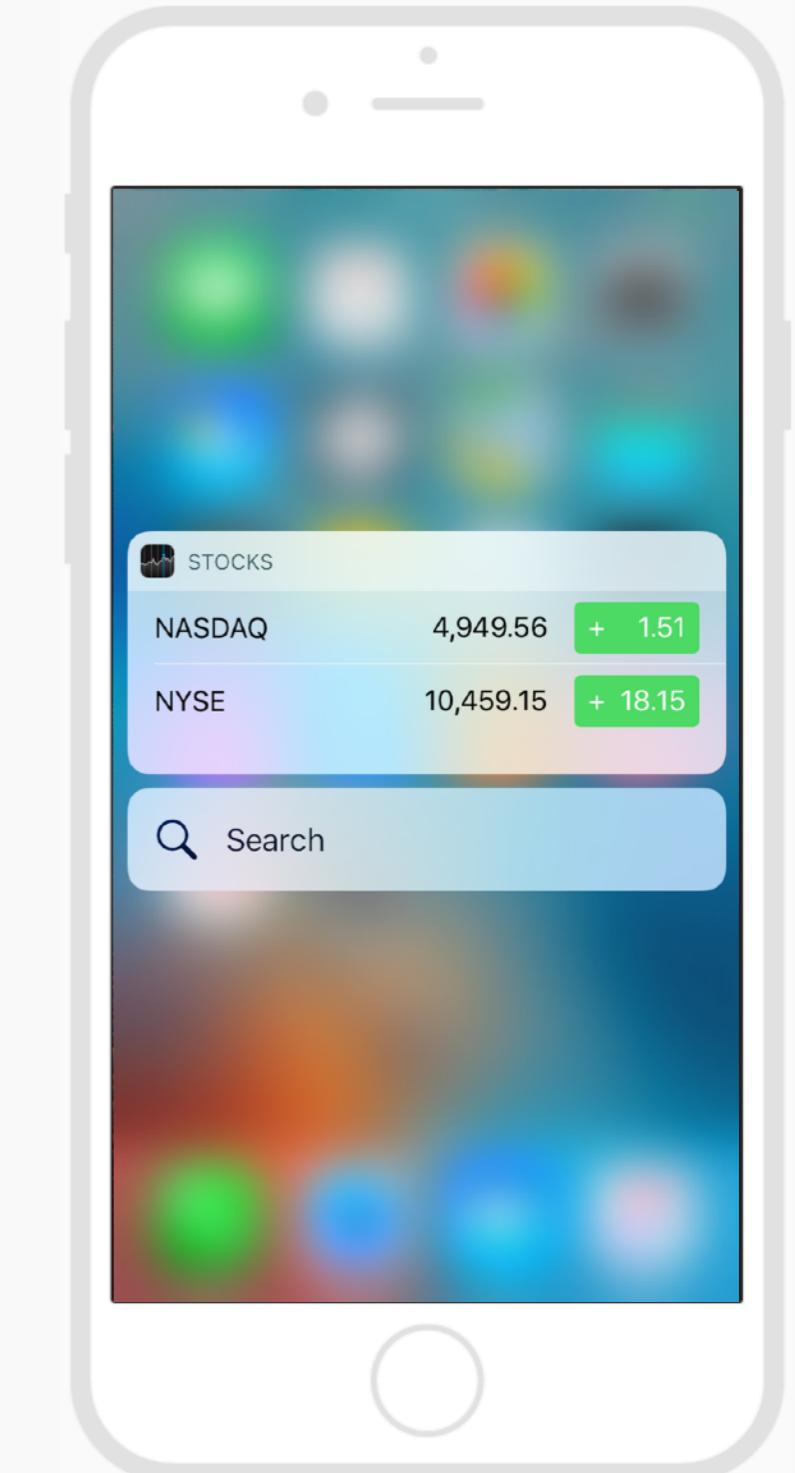
Search Widgets

Display notifications from your application on the user's Search and Home Screen

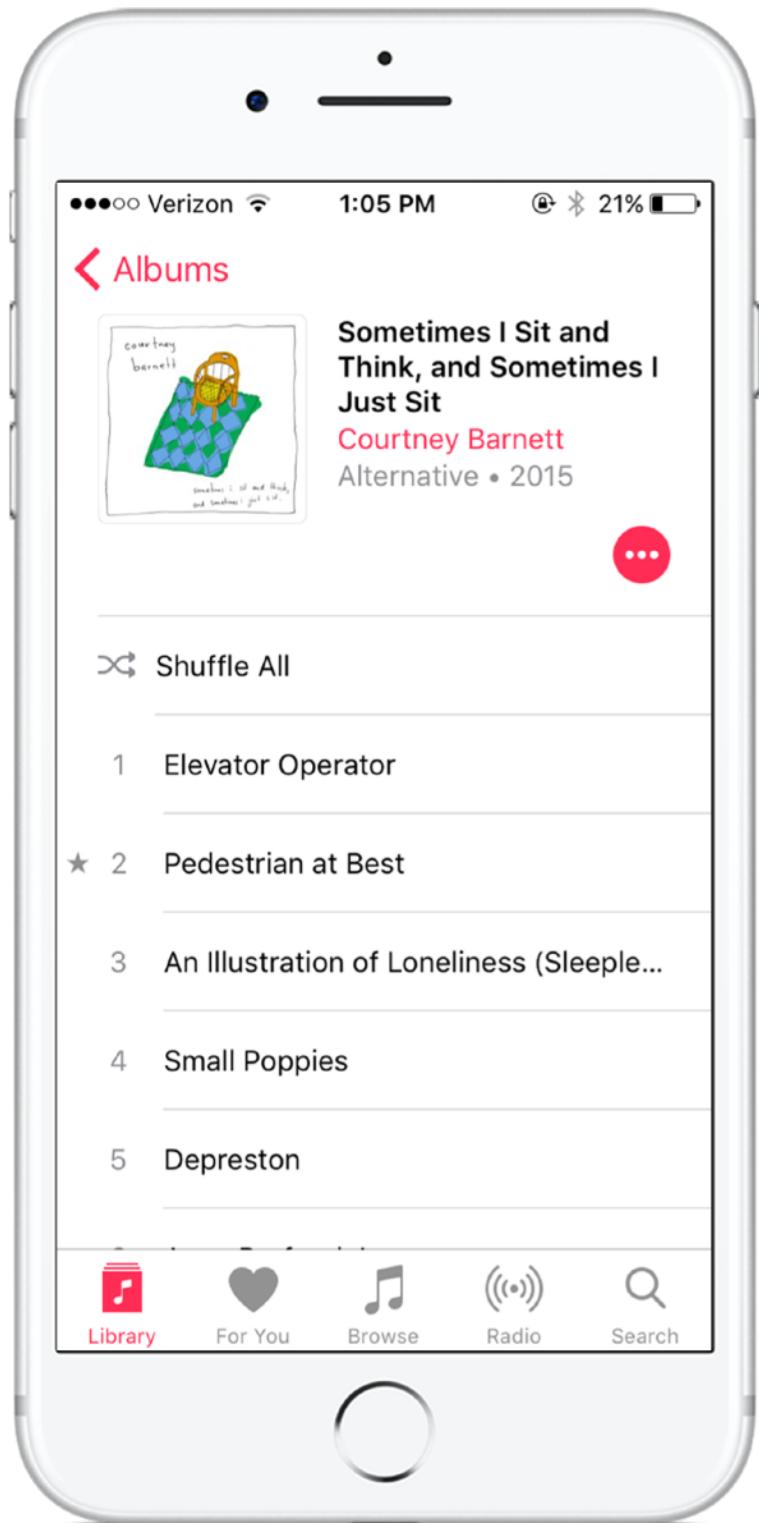
Very customizable (can add buttons, images, layout customization, etc.)

To view widgets

Home Screen > apply pressure on an app icon using 3D Touch



iOS HIG : Interface Terminology



Bars

Lets your users know “where” they are in their application. May contain buttons to trigger navigation (segues) and titles to clarify location in app

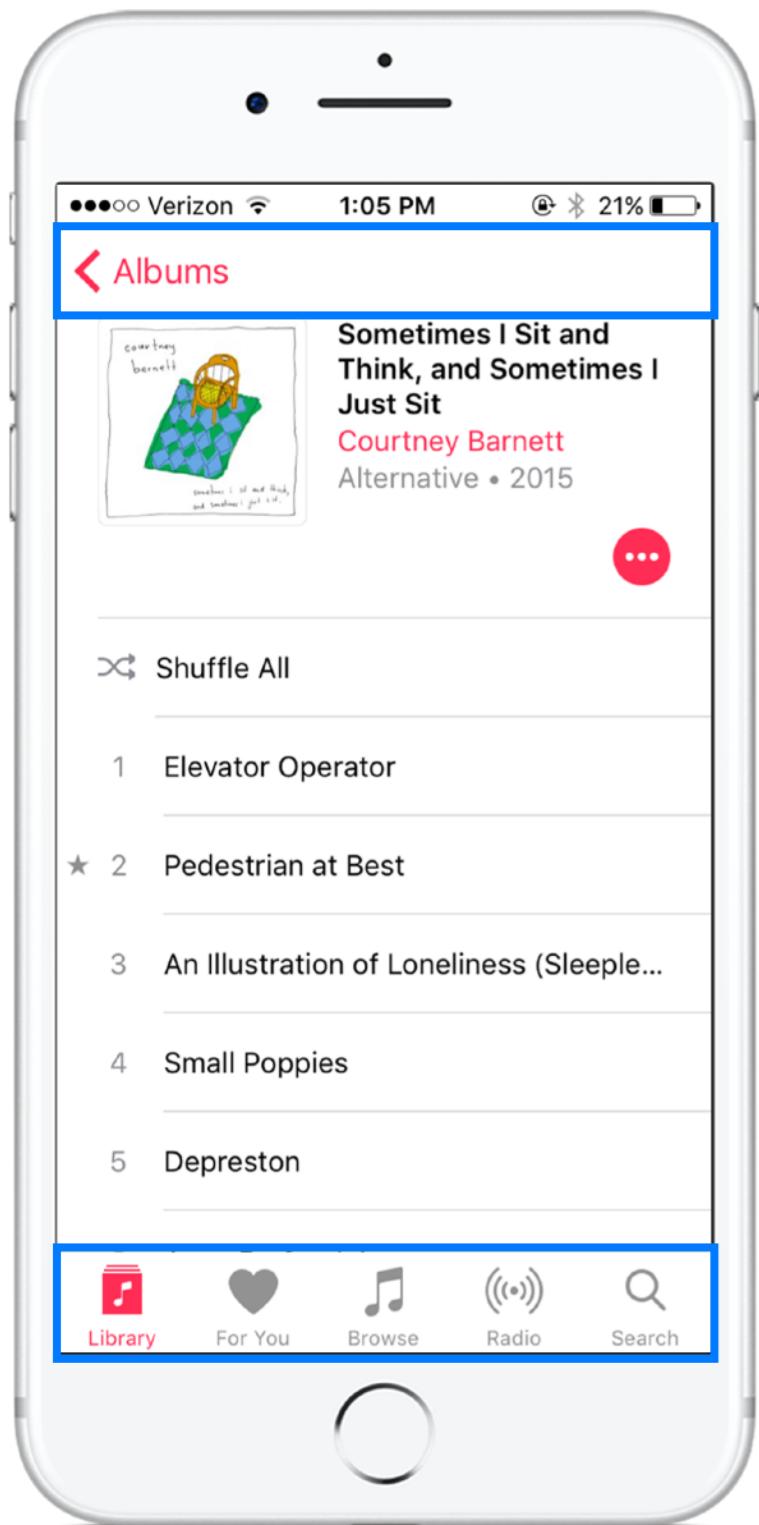
Views

Contain the content of what the users sees. This includes both the entire “screen” visible, as well as the other subviews (text, graphics, etc.)

Controls

Buttons, text fields, segmented controls, pickers,

iOS HIG : Interface Terminology



Bars

Lets your users know “where” they are in their application. May contain buttons to trigger navigation (segues) and titles to clarify location in app

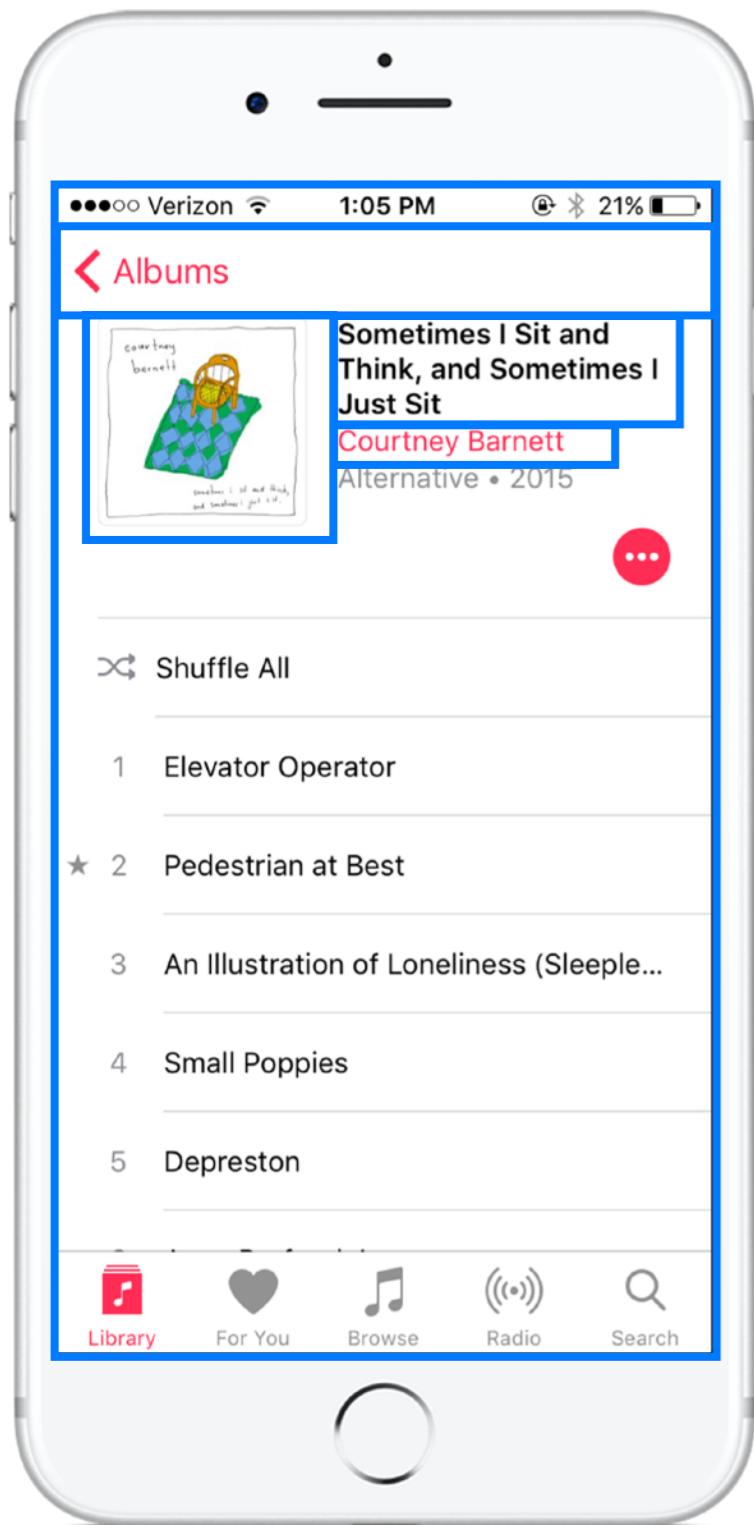
Views

Contain the content of what the users sees. This includes both the entire “screen” visible, as well as the other subviews (text, graphics, etc.)

Controls

Buttons, text fields, segmented controls, pickers,

iOS HIG : Interface Terminology



Bars

Lets your users know “where” they are in their application. May contain buttons to trigger navigation (segues) and titles to clarify location in app

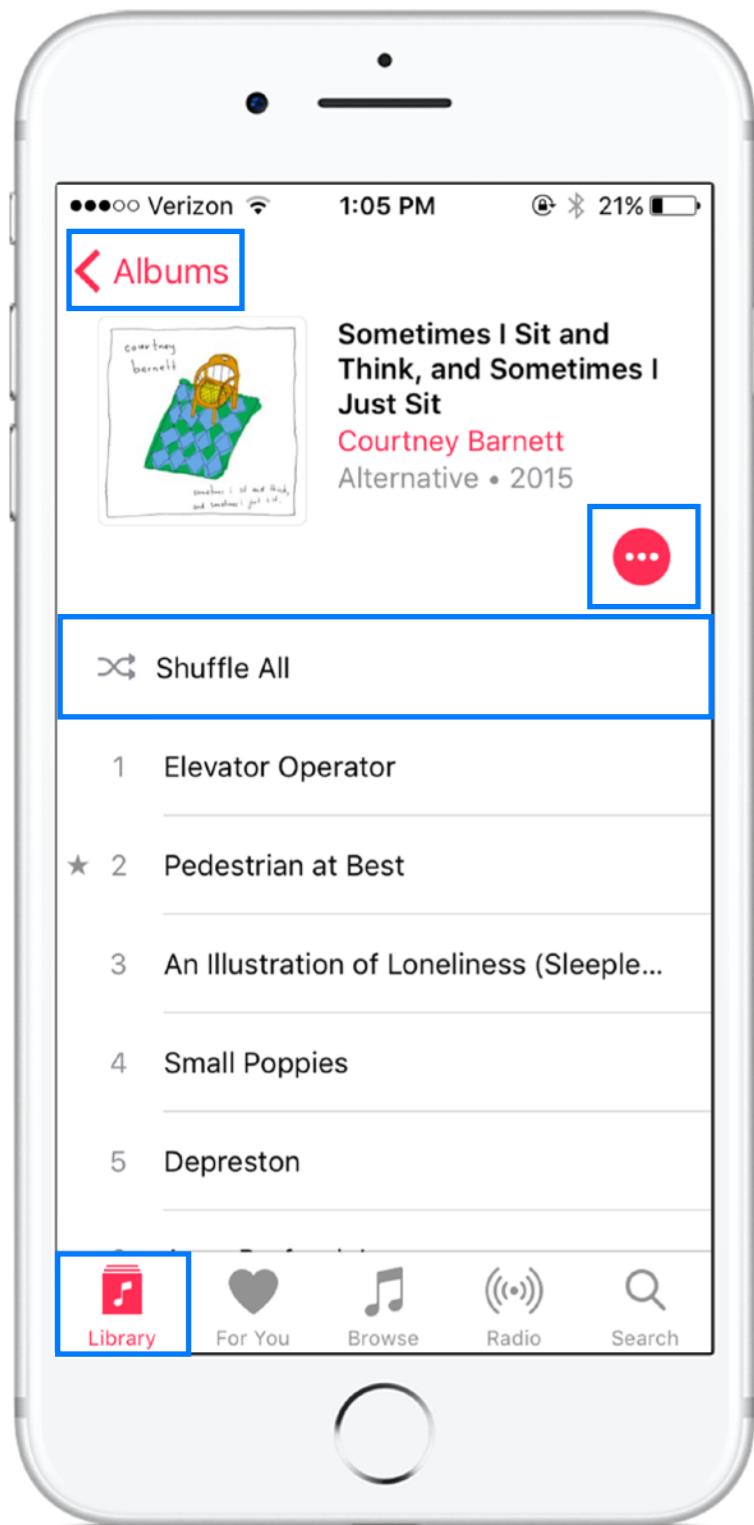
Views

Contain the content of what the users sees. This includes both the entire “screen” visible, as well as the other subviews (text, graphics, etc.)

Controls

Buttons, text fields, segmented controls, pickers,

iOS HIG : Interface Terminology



Bars

Lets your users know “where” they are in their application. May contain buttons to trigger navigation (segues) and titles to clarify location in app

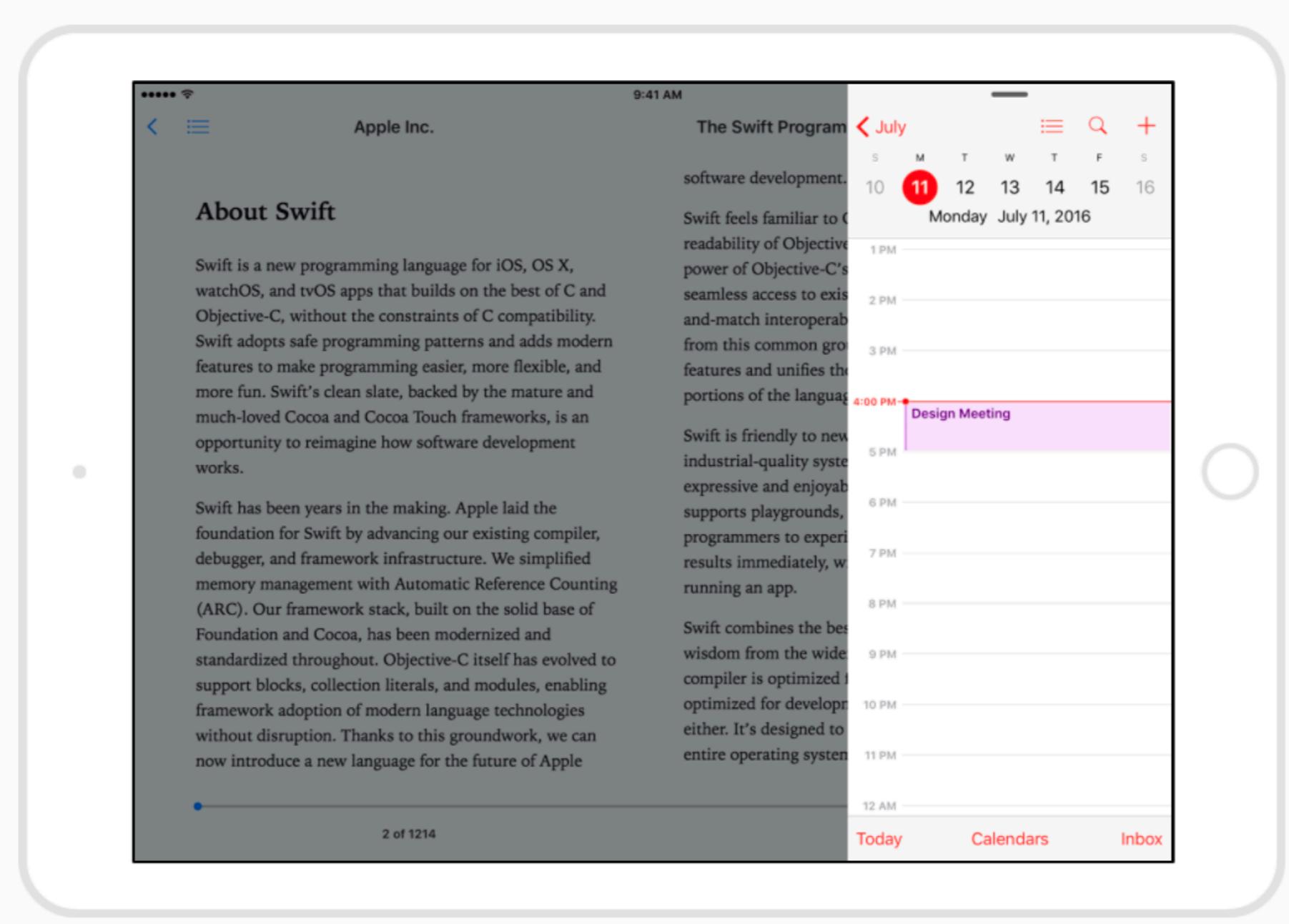
Views

Contain the content of what the users sees. This includes both the entire “screen” visible, as well as the other subviews (text, graphics, etc.)

Controls

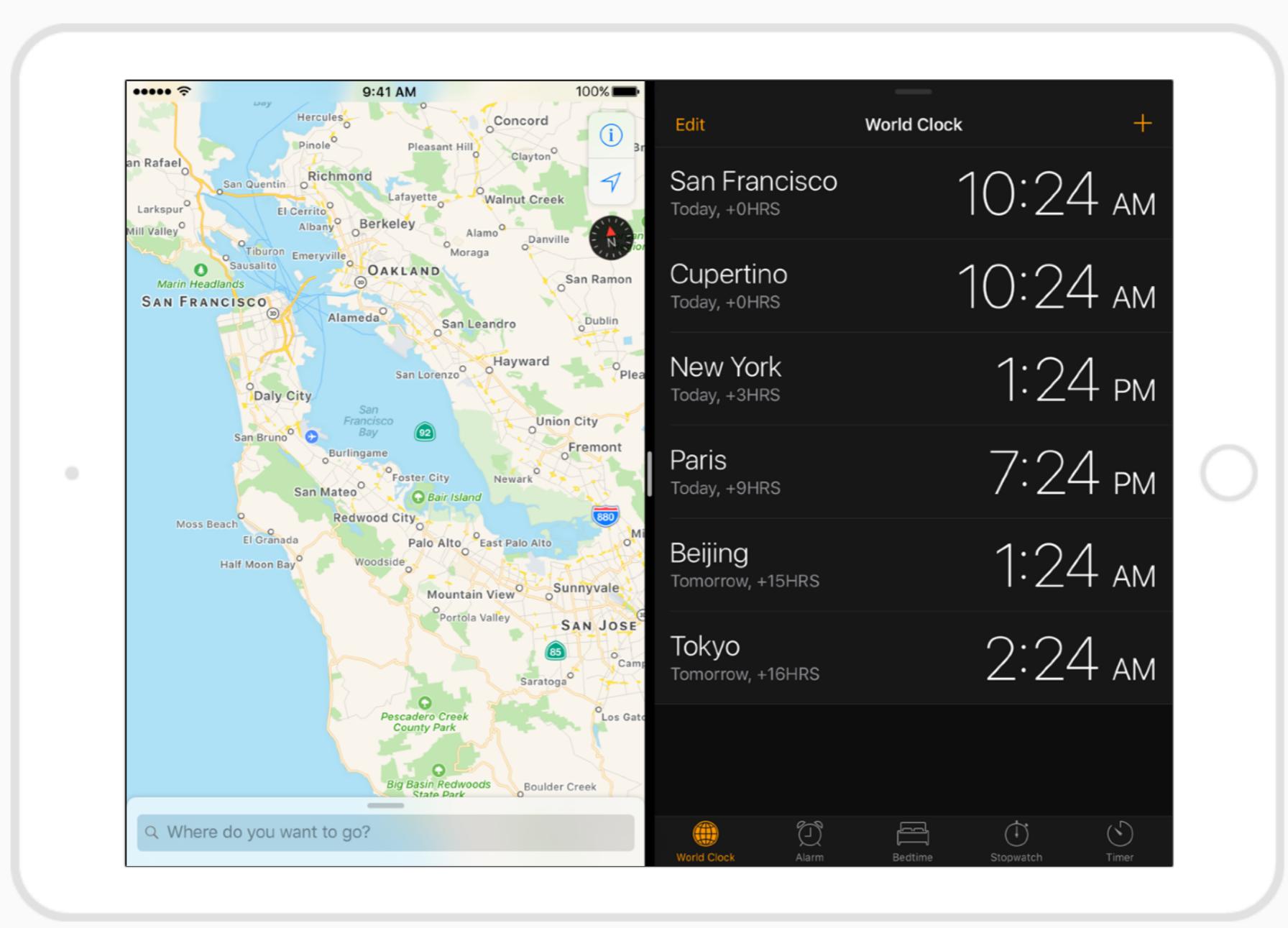
Buttons, text fields, segmented controls, etc.

iOS HIG : Multitasking (iPad)



Designing with Multitasking in mind (example Slide Over)

iOS HIG : Multitasking (iPad)



Designing with Multitasking in mind (example Slide View)

iOS HIG : Branding

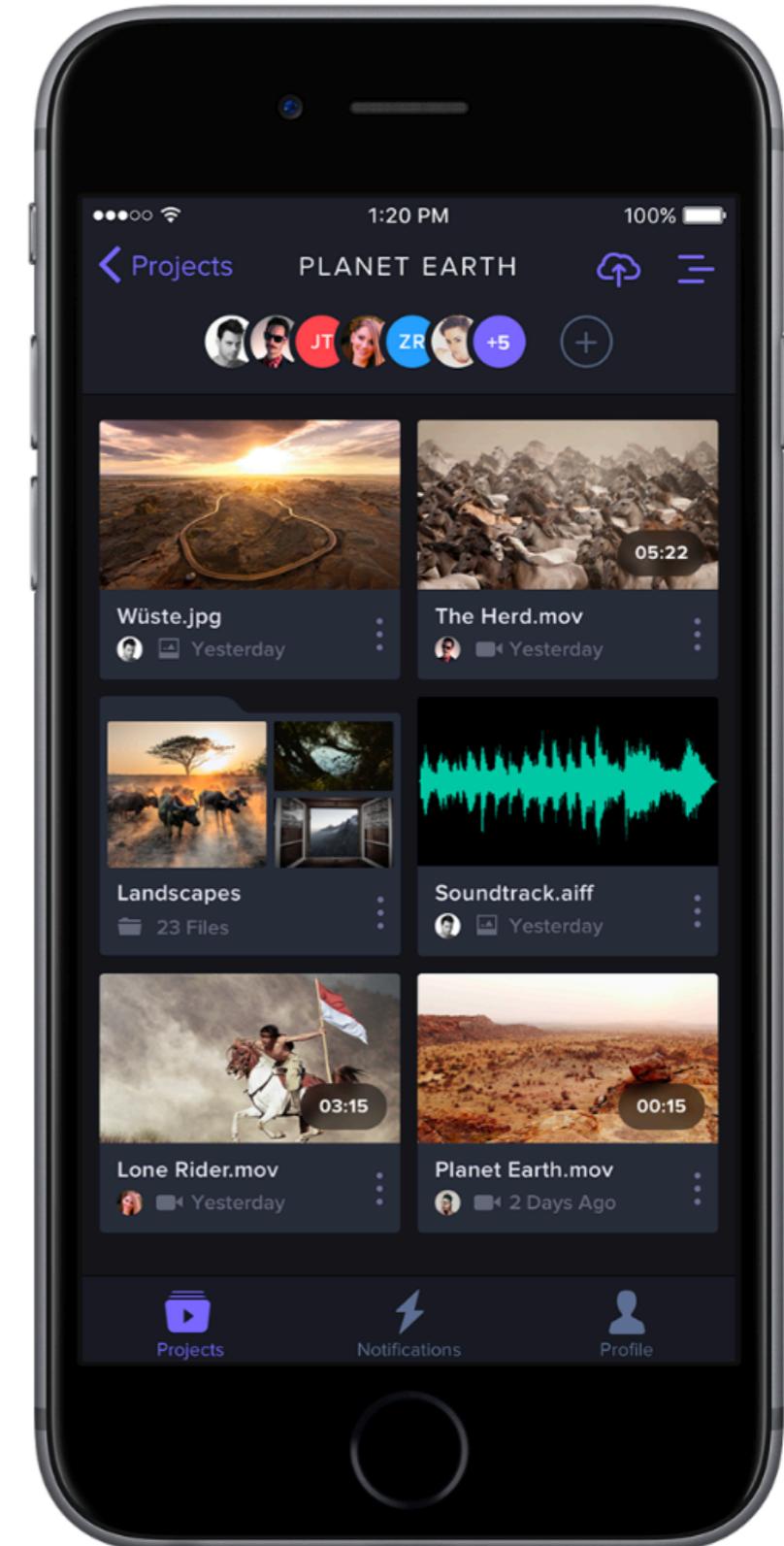
While it is important to have an app “image” or “brand”, avoid over-using logos, icon images, etc.

Examples:

No need to include logo in every view of your application

Focus on design schemes (fonts, colors, layouts) rather than overt branding

Avoid sacrificing screen space for your brand unless necessary



Frame.io
Video Collaboration

iOS HIG : Branding

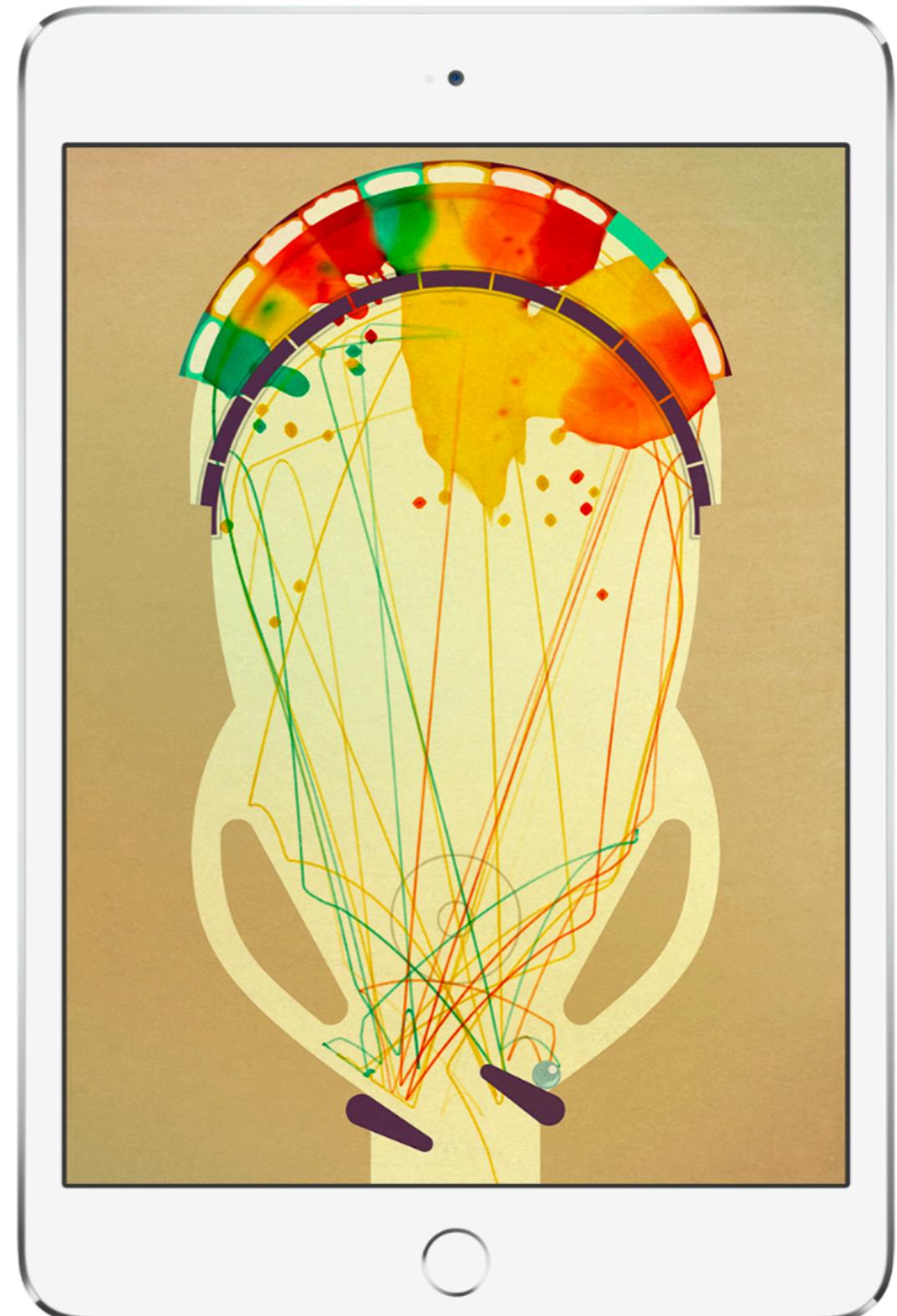
While it is important to have an app “image” or “brand”, avoid over-using logos, icon images, etc.

Examples:

No need to include logo in every view of your application

Focus on design schemes (fonts, colors, layouts) rather than overt branding

Avoid sacrificing screen space for your brand unless necessary



INKS

State of Play Games

iOS HIG : Color

Add cohesion to your app by defining a consistent color scheme

Distinguish between interactive and un-interactive UI elements using color

Create color constants to be used throughout your application by creating `UIColor` objects

R 255
G 59
B 48

Red

R 255
G 149
B 0

Orange

R 255
G 204
B 0

Yellow

R 76
G 217
B 100

Green

R 90
G 200
B 250

Teal Blue

R 0
G 122
B 255

Blue

R 88
G 86
B 214

Purple

R 255
G 45
B 85

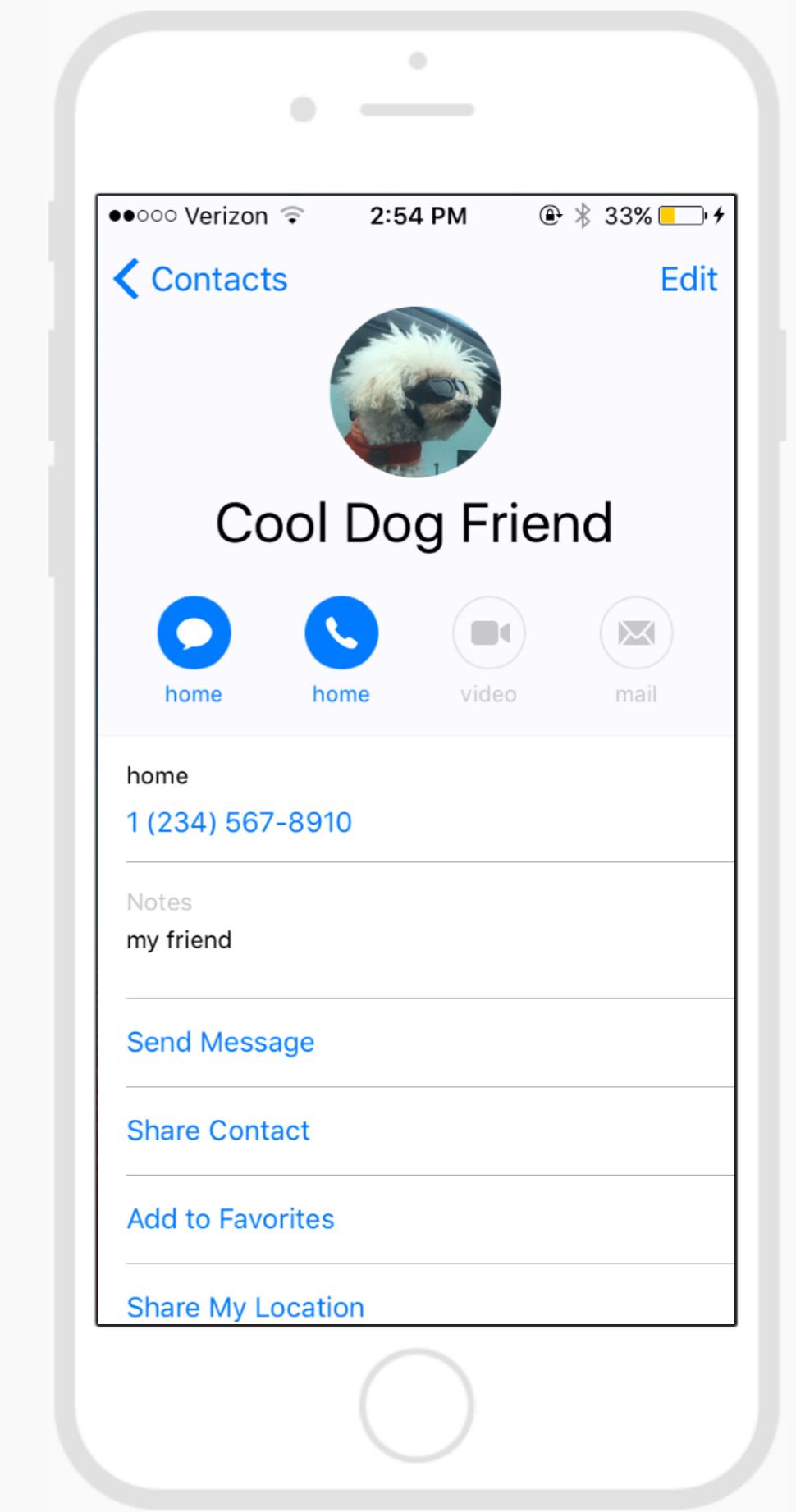
Pink

iOS HIG : Color

Add cohesion to your app by defining a consistent color scheme

Distinguish between interactive and un-interactive UI elements using color

Create color constants to be used throughout your application by creating UIColor objects

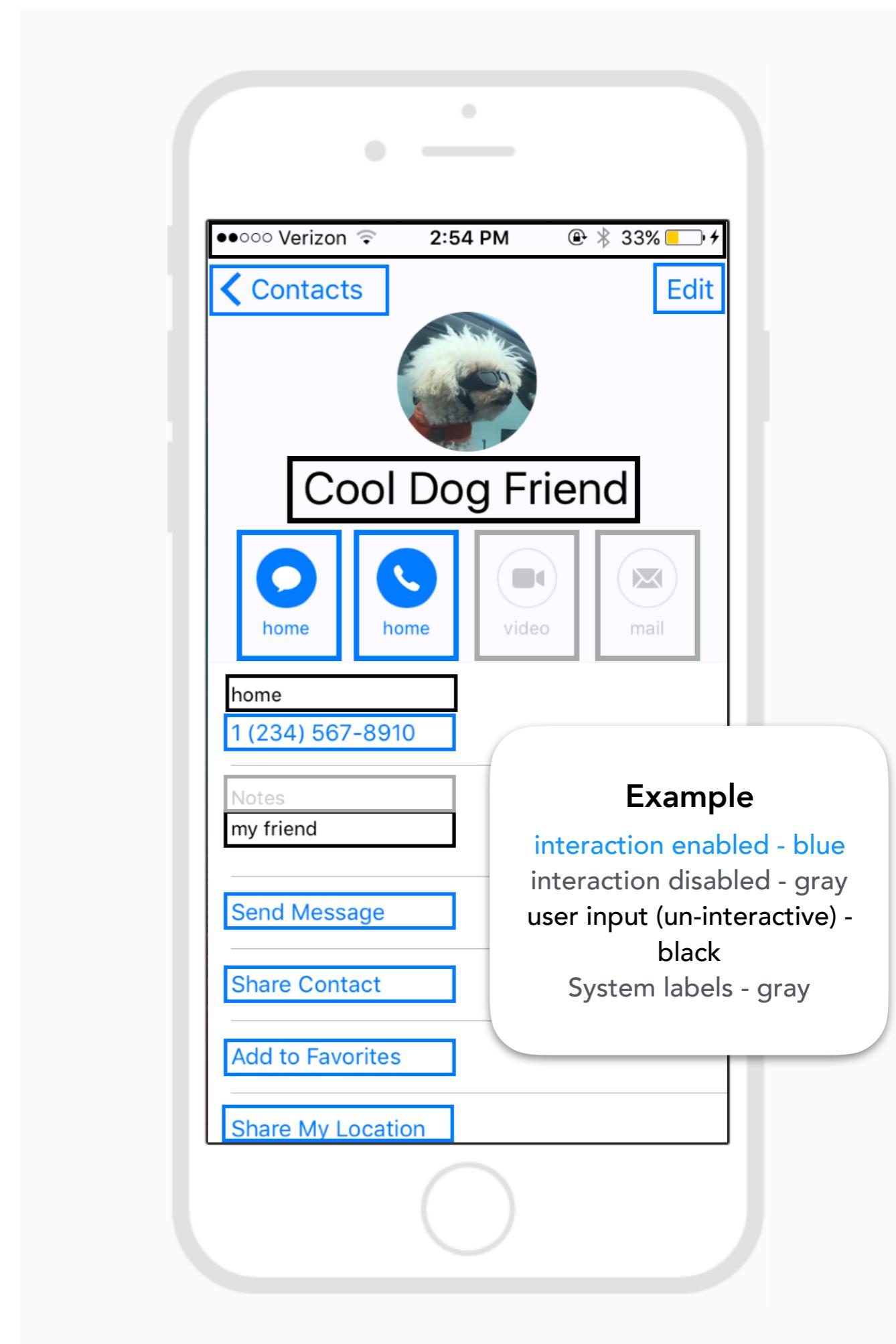


iOS HIG : Color

Add cohesion to your app by defining a consistent color scheme

Distinguish between interactive and un-interactive UI elements using color

Create color constants to be used throughout your application by creating UIColor objects



iOS HIG : Color

Add cohesion to your app by defining a consistent color scheme

Distinguish between interactive and un-interactive UI elements using color

Create color constants to be used throughout your application by creating `UIColor` objects

Common iOS Design practice to set “Enabled Color” as your app’s brand color



Pages
By Apple

The screenshot shows a mobile application interface. At the top is a navigation bar with a back arrow, a refresh icon, a plus sign, a user profile icon with a checkmark, and a three-dot menu icon. Below the navigation bar is a status bar displaying signal strength, network (Verizon LTE), time (3:55 PM), battery level (88%), and a battery icon. The main content area contains the following text:

iOS DeCal (Spring 2017) - Final Project

App Proposal Due - Tuesday, March 21 at 11:59pm
App Proposal Review - Thursday, March 23 (during lab)
Progress Check-In - Thursday, April 6 (during lab)
Code Due - Tuesday, May 2 at 11:59pm
Final Presentations - Friday, May 5 at 10:00am

For your final project, you will be creating an iOS application completely of your own design. You may either work in a group (up to 4 people total) or individually, but keep in mind that we will be expecting more from larger groups.

App Proposal (due 3/21)
Before starting on your project, you'll need to write up an app proposal providing an overview of your application. Your proposal must include the following information:

1. **Group Members** - List each member of your group along with their berkeley.edu email (or just your name/e-mail if working alone).

iOS HIG : Color

Add cohesion to your app by defining a consistent color scheme

Distinguish between interactive and un-interactive UI elements using color

Create color constants to be used throughout your application by creating `UIColor` objects

Creating a `UIColor` object with Predefined Colors

`class var black: UIColor`

A color object in the sRGB color space whose grayscale value is 0.0 and whose alpha value is 1.0.

`class var blue: UIColor`

A color object whose RGB values are 0.0, 0.0, and 1.0 and whose alpha value is 1.0.

`class var brown: UIColor`

A color object whose RGB values are 0.6, 0.4, and 0.2 and whose alpha value is 1.0.

`class var clear: UIColor`

A color object whose grayscale and alpha values are both 0.0.

`class var cyan: UIColor`

A color object whose RGB values are 0.0, 1.0, and 1.0 and whose alpha value is 1.0.

`class var darkGray: UIColor`

A color object whose grayscale value is 1/3 and whose alpha value is 1.0.

... and more (see [UIColor](#))

iOS HIG : Color

Add cohesion to your app by defining a consistent color scheme

Distinguish between interactive and un-interactive UI elements using color

Create color constants to be used throughout your application by creating UIColor objects

Creating a Custom UIColor object using Color Spaces

`init(white: CGFloat, alpha: CGFloat)`

Initializes and returns a color object using the specified opacity and grayscale values.

`init(hue: CGFloat, saturation: CGFloat, brightness: CGFloat, alpha: CGFloat)`

Initializes and returns a color object using the specified opacity and HSB color space component values.

`init(red: CGFloat, green: CGFloat, blue: CGFloat, alpha: CGFloat)`

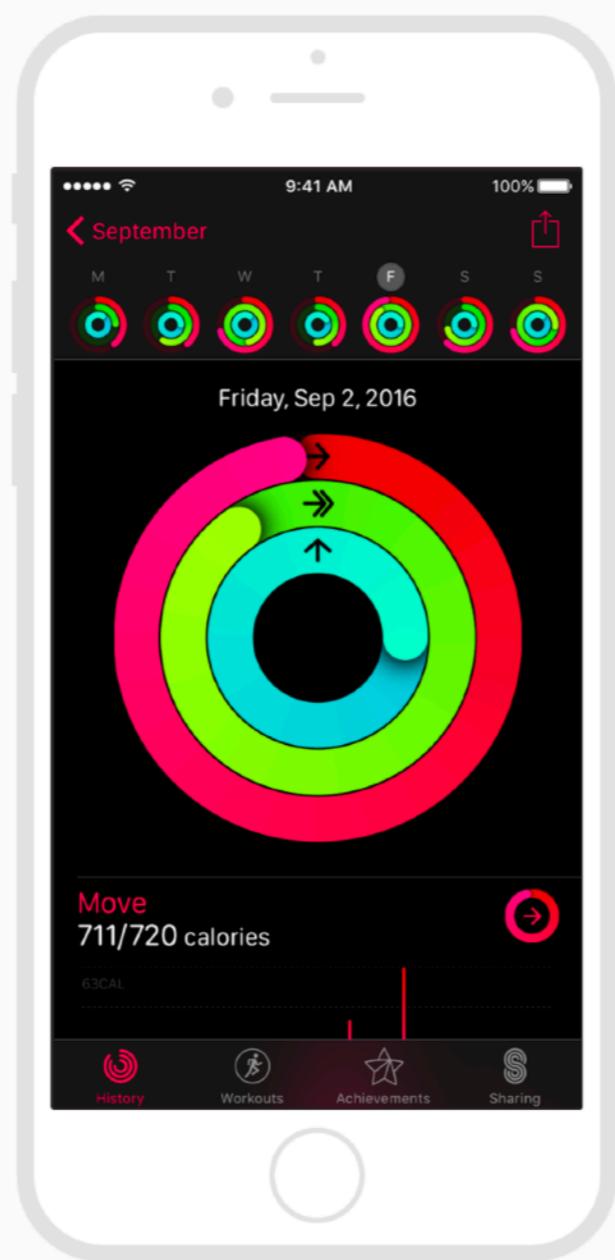
Initializes and returns a color object using the specified opacity and RGB component values.

`init(displayP3Red: CGFloat, green: CGFloat, blue: CGFloat, alpha: CGFloat)`

Initializes and returns a color object using the specified opacity and RGB component values in the Display P3 color space.

iOS HIG : Color

**App Colors
(standard)**



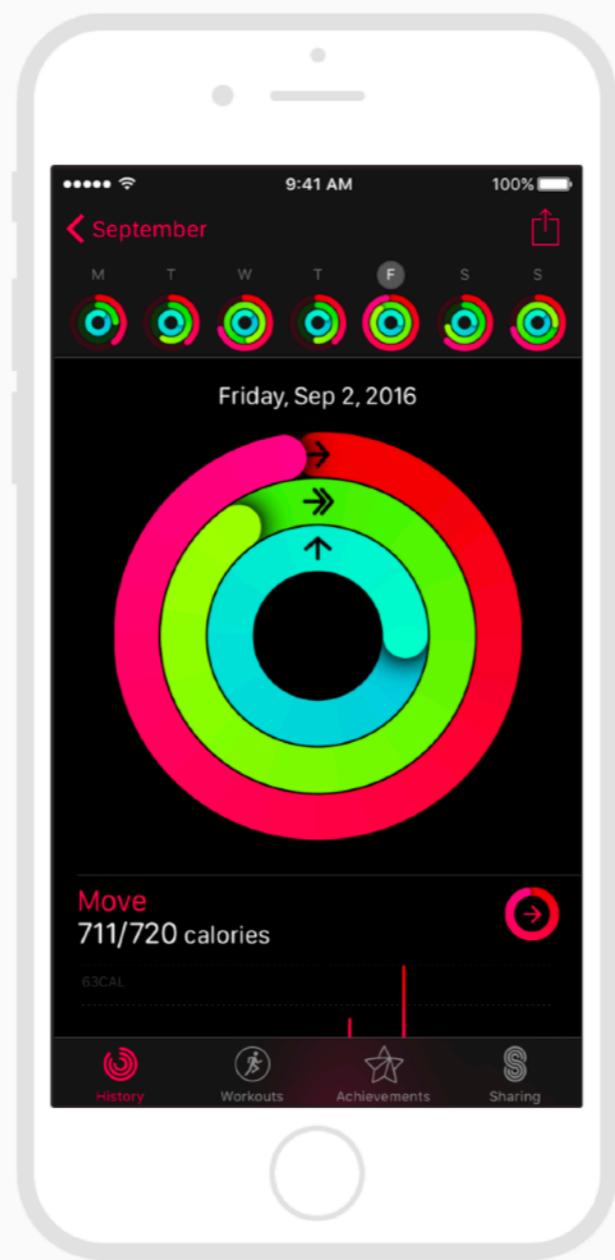
**App Colors
(without red-green)**



Keep in mind what your app will look like for users with various types of color vision impairment

iOS HIG : Color

**App Colors
(standard)**

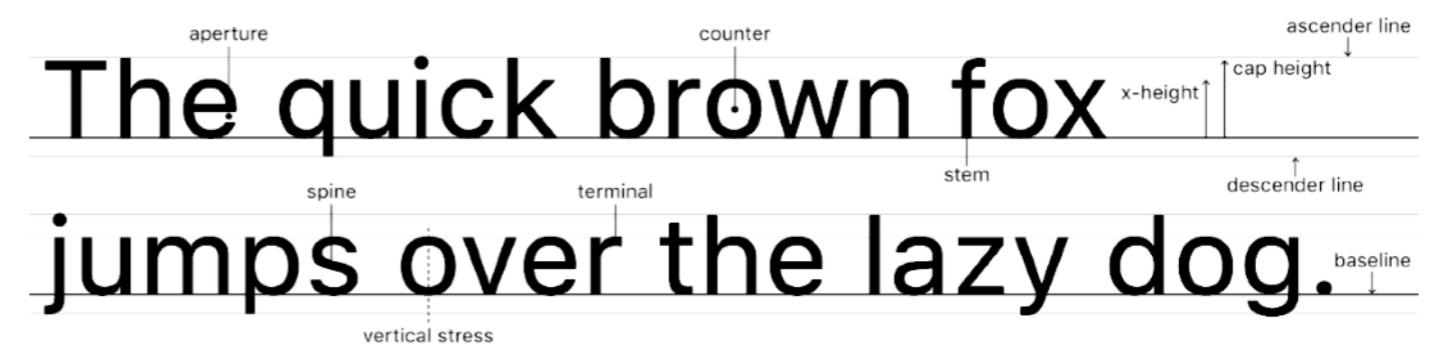


**App Colors
(without red-green)**



Photoshop has accessibility color filters to help you do this
<http://www.adobe.com/accessibility/products/photoshop.html>

iOS HIG : Fonts and Typography



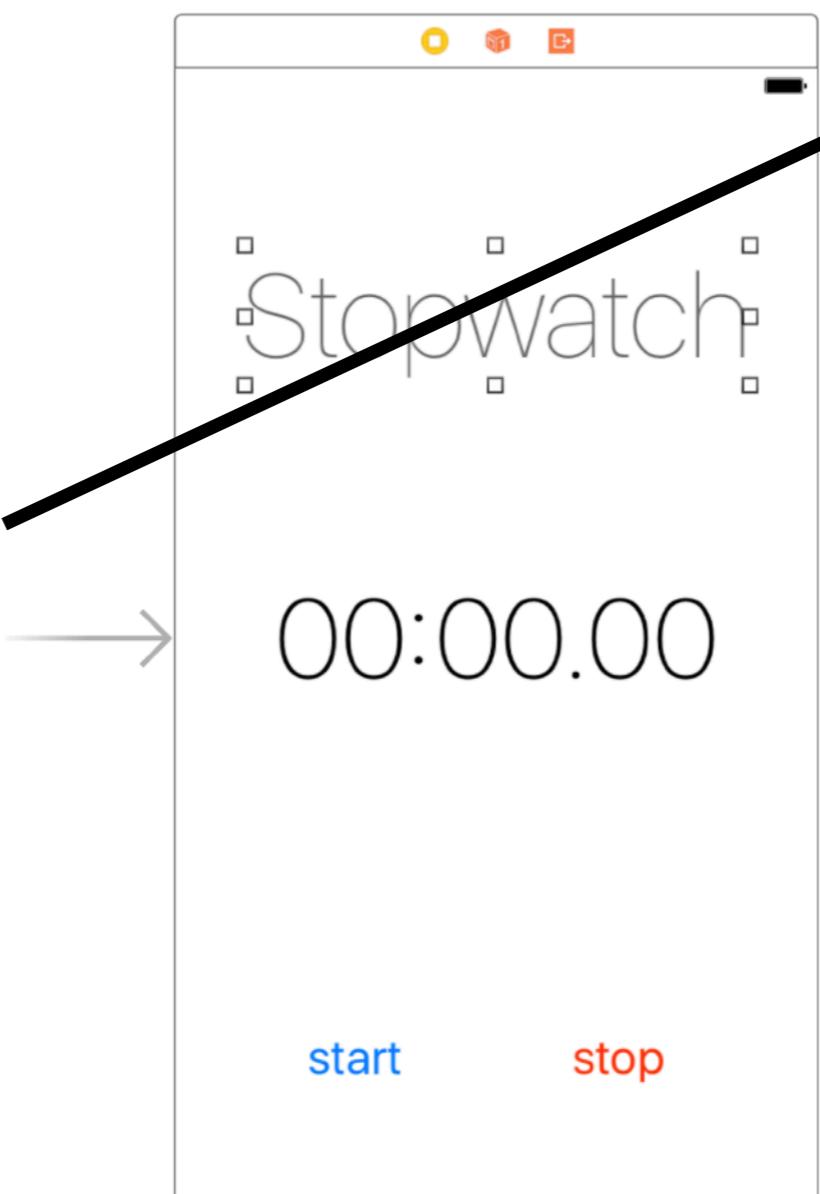
San Francisco

The System Font for iOS

Created by a team at Apple in 2014

iOS HIG : Fonts

When you add new UI elements with text to your app, the font family will default to **System** (San Francisco)



Label

Text Plain

Stopwatch

Color Dark Gray Color

Font System Ultra Light 69.0

Font System - System

Family

Style Ultra Light

Size 69

Done

Autoshrink Fixed Font Size

Tighten Letter Spacing

Highlighted Default

Shadow Default

Shadow Offset 0 -1

Width Height

View

Content Mode Left

Semantic Unspecified

Tag 0

User Interaction Enabled

Multiple Touch

Alpha 1

Background

Tint Default

Opaque Hidden

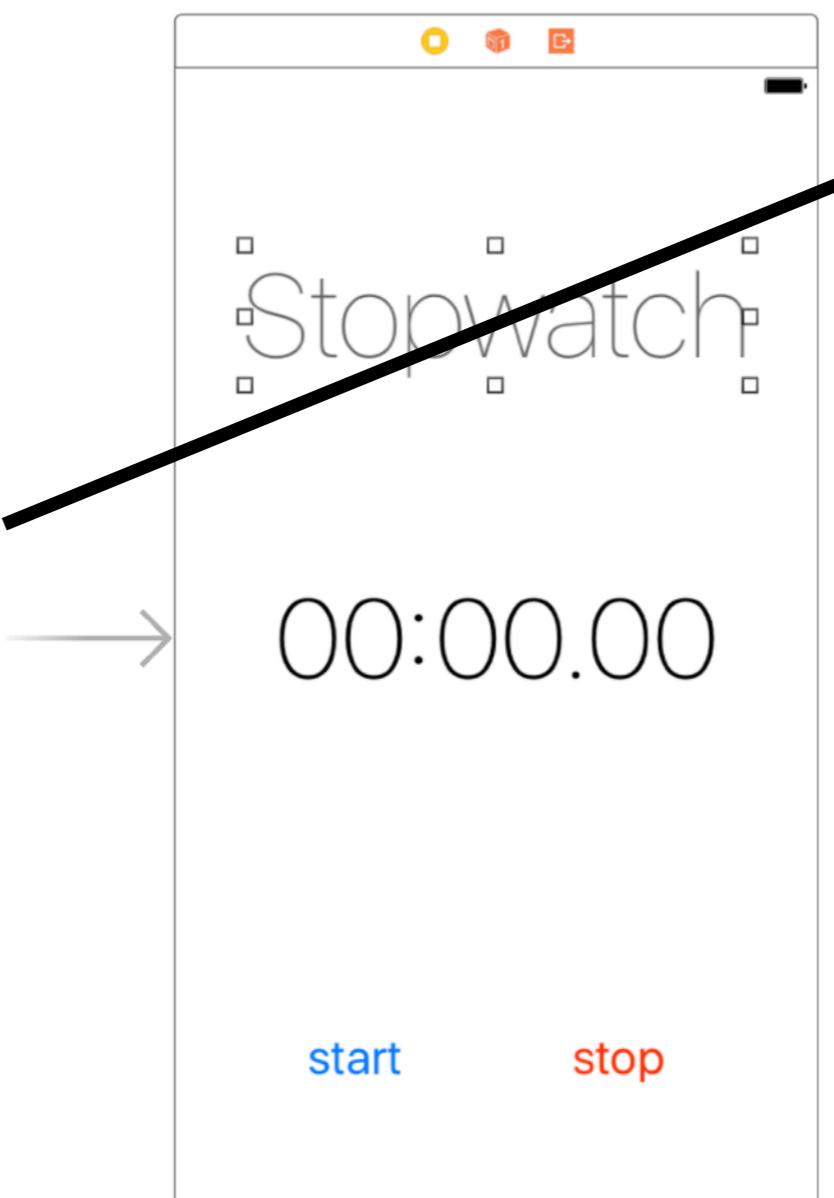
Labels

Labels

iOS HIG : Fonts

Set Font to
“Custom” to
change to a
different Font

Family



Label

Text Plain

Stopwatch

Color Dark Gray Color

Font System Ultra Light 69.0

Font Custom

Family Helvetica Neue

Style Regular

Size 17

Done

Autoshrink Fixed Font Size

Tighten Letter Spacing

Highlighted Default

Shadow Default

Shadow Offset 0 -1

Width Height

View

Content Mode Left

Semantic Unspecified

Tag 0

User Interaction Enabled

Multiple Touch

Alpha 1

Background

Tint Default

Opaque

Hidden

iOS HIG : Fonts

Generally, try to stick to one font throughout your entire app

Instead of using different fonts, try experimenting with a few different font styles, weights, and sizes (all within the same font family)

Example: Helvetica Neue typeface weights

Helvetica Neue Thin

Helvetica Neue Light

Helvetica Neue Regular

Helvetica Neue Medium

Helvetica Neue Bold

Programmatic Design

Storyboard : Review

Up to now, you have been creating applications using Storyboard / Interface Builder

Pros of Storyboard

Drag and drop interface makes it really easy to visualize your application immediately

Relatively low learning curve

Great for small projects

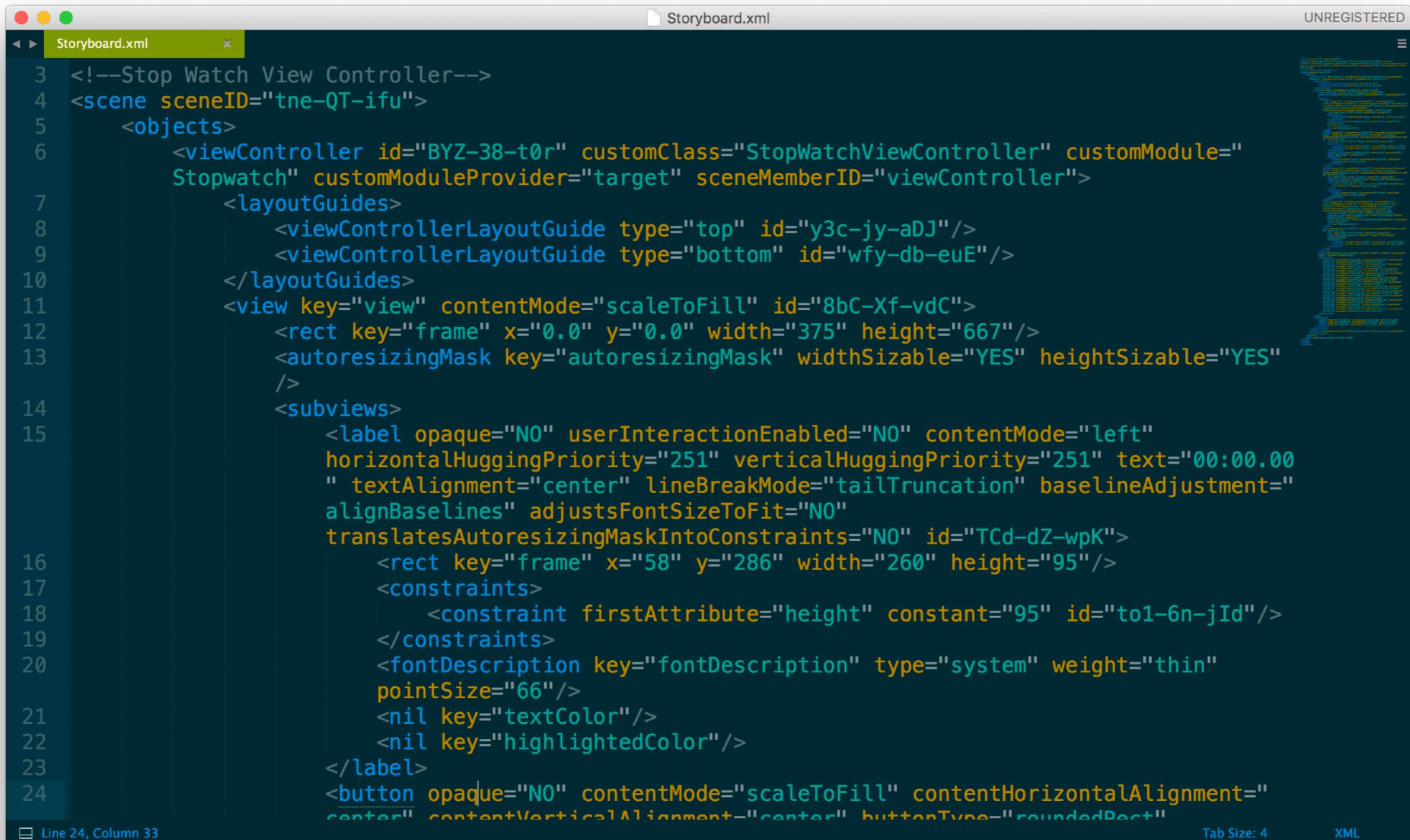
The future of User Interface programming?

Storyboard : Beneath the hood

The image shows two side-by-side screenshots of Xcode. On the left, the 'Storyboard.xml' file is open in a code editor. The XML code defines a storyboard scene for a Stopwatch view controller. It includes details like the view's frame (x=0.0, y=0.0, width=375, height=667), its autoresizing mask (widthSizable=YES, heightSizable=YES), and its subviews, which consist of a central label displaying '00:00.00' and a button below it. On the right, the storyboard is previewed in a simulator window, showing a digital stopwatch interface with the time '00:00.00' and 'start' and 'stop' buttons.

```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0"
toolsVersion="11762" systemVersion="16D32" targetRuntime="iOS.CocoaTouch"
propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" colorMatched="YES"
initialViewController="BYZ-38-t0r">
<!—Stop Watch View Controller-->
<scene sceneID="tne-QT-ifu">
<objects>
    <viewController id="BYZ-38-t0r" customClass="StopWatchViewController"
    customModule="Stopwatch" customModuleProvider="target" sceneMemberID="viewController">
        <layoutGuides>
            <viewControllerLayoutGuide type="top" id="y3c-jy-aDJ"/>
            <viewControllerLayoutGuide type="bottom" id="wfy-db-euE"/>
        </layoutGuides>
        <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
            <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
            <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
            <subviews>
                <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
                horizontalHuggingPriority="251" verticalHuggingPriority="251" text="00:00.00"
                textAlignment="center" lineBreakMode="tailTruncation"
                baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                translatesAutoresizingMaskIntoConstraints="NO" id="TCd-dZ-wpK">
                    <rect key="frame" x="58" y="286" width="260" height="95"/>
                    <constraints>
                        <constraint firstAttribute="height" constant="95" id="to1-6n-jId"/>
                    </constraints>
                    <fontDescription key="fontDescription" type="system" weight="thin"
                    pointSize="66"/>
                    <nil key="textColor"/>
                    <nil key="highlightedColor"/>
                </label>
                <button opaque="NO" contentMode="scaleToFill"
                contentHorizontalAlignment="center" contentVerticalAlignment="center"
                buttonType="roundedRect" lineBreakMode="middleTruncation"
                translatesAutoresizingMaskIntoConstraints="NO" id="udX-dS-dmN">
                    <rect key="frame" x="55" y="528.5" width="100" height="100"/>
                    <constraints>
                        <constraint firstAttribute="width" constant="100" id="Gz-1J-5fU"/>
                    </constraints>
                </button>
            </subviews>
        </view>
    </viewController>
</objects>
</scene>
</document>
```

Main.storyboard files are just XML files



The screenshot shows a text editor window with the title "Storyboard.xml". The code displayed is the XML configuration for a storyboard scene. The scene contains a view controller with a label and a button. The label has a font size of 66 and a height constraint of 95. The button has a rounded rectangular type. The code is color-coded for readability.

```
3 <!--Stop Watch View Controller-->
4 <scene sceneID="tne-QT-ifu">
5   <objects>
6     <viewController id="BYZ-38-t0r" customClass="StopWatchViewController" customModule="Stopwatch" customModuleProvider="target" sceneMemberID="viewController">
7       <layoutGuides>
8         <viewControllerLayoutGuide type="top" id="y3c-jy-aDJ"/>
9         <viewControllerLayoutGuide type="bottom" id="wfy-db-euE"/>
10    </layoutGuides>
11    <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
12      <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
13      <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES" />
14    <subviews>
15      <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="00:00.00" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="TCd-dZ-wpK">
16        <rect key="frame" x="58" y="286" width="260" height="95"/>
17        <constraints>
18          <constraint firstAttribute="height" constant="95" id="to1-6n-jId"/>
19        </constraints>
20        <fontDescription key="fontDescription" type="system" weight="thin" pointSize="66"/>
21        <nil key="textColor"/>
22        <nil key="highlightedColor"/>
23      </label>
24      <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect" id="qkD-0m-0LJ">
```

You can view the file generated by Interface Builder by opening up Main.storyboard in any text editor

The screenshot shows the Xcode interface with the storyboard file open. A blue arrow points from the word "UILabel" in the text below to the corresponding XML code in the storyboard file. The code highlights the configuration for a label, including its frame, constraints, font, and text.

```
3 <!--Stop Watch View Controller-->
4 <scene sceneID="tne-QT-ifu">
5   <objects>
6     <viewController id="BYZ-38-t0r" customClass="StopWatchViewController" customModule="Stopwatch" customModuleProvider="target" sceneMemberID="viewController">
7       <layoutGuides>
8         <viewControllerLayoutGuide type="top" id="y3c-jy-aDJ"/>
9         <viewControllerLayoutGuide type="bottom" id="wfy-db-euE"/>
10    </layoutGuides>
11    <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
12      <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
13      <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES" />
14    <subviews>
15      <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="00:00.00" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="TCd-dZ-wpK">
16        <rect key="frame" x="58" y="286" width="260" height="95"/>
17        <constraints>
18          <constraint firstAttribute="height" constant="95" id="to1-6n-jId"/>
19        </constraints>
20        <fontDescription key="fontDescription" type="system" weight="thin" pointSize="66"/>
21        <nil key="textColor"/>
22        <nil key="highlightedColor"/>
23      </label>
24      <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect" contentHorizontalAlignment="center" contentVerticalAlignment="center" id="t01-6n-jId"/>

```

UILabel

Each time you add a button / label / constraint / etc.,
you'll be able to see it added to this file

Storyboard : Problems

Cons of using Storyboard

Easy to get cluttered for larger scale applications

XML files are prone to merge conflicts when using version control (git)

No way to define layout constants or easy way to reuse UI layouts

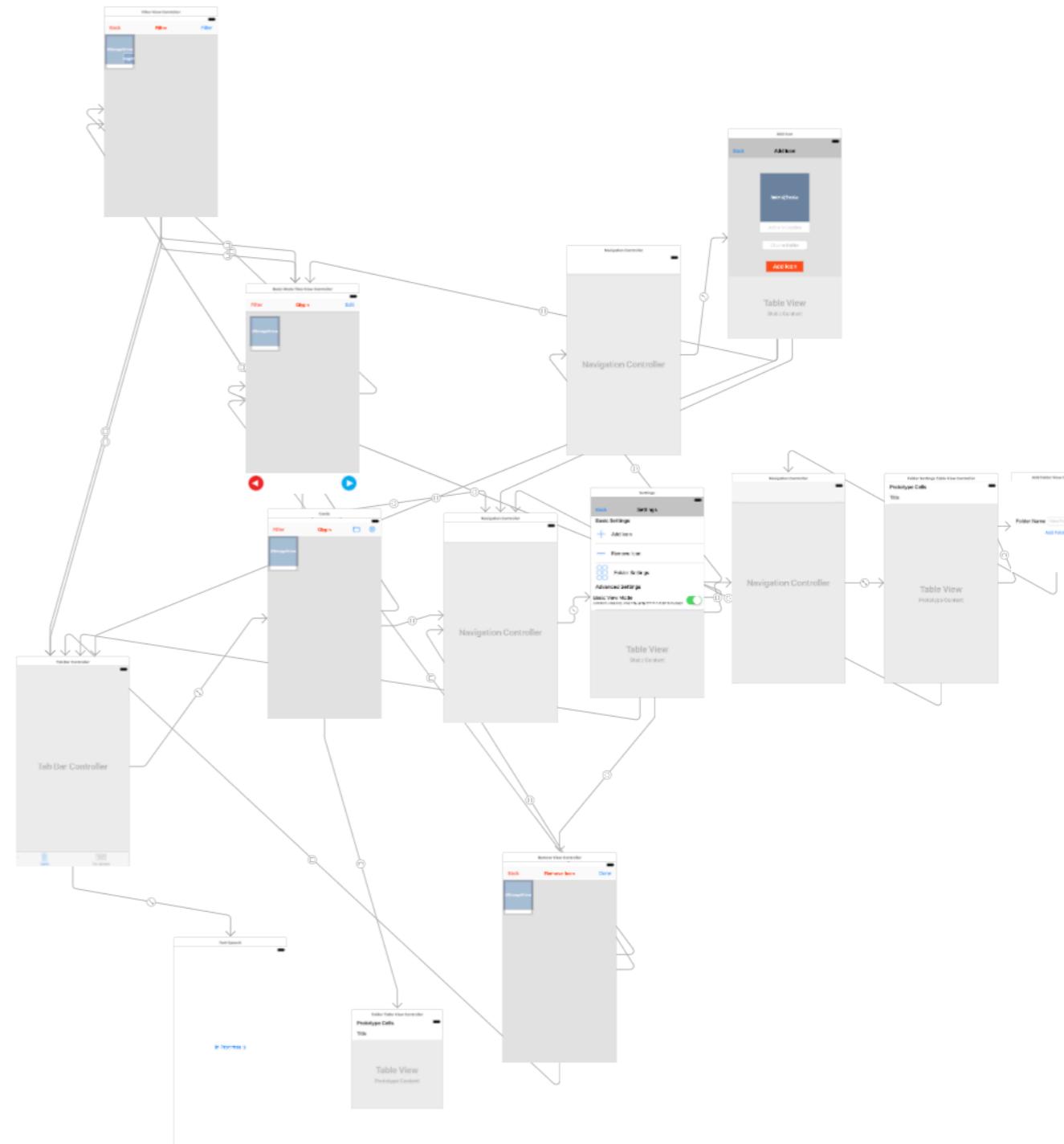
Storyboard : Problems

Cons of using Storyboard

Easy to get cluttered for larger scale applications

XML files are prone to merge conflicts when using version control (git)

No way to define layout constants or easy way to reuse UI layouts



Storyboard : Problems

Cons of using Storyboard

Easy to get cluttered for larger scale applications

XML files are prone to merge conflicts when using version control (git)

No way to define layout constants or easy way to reuse UI layouts



The screenshot shows a GitHub Gist page for a user named neilinglis. The gist was created 3 years ago and has one revision. The title of the file is "gistfile1.txt". The content of the file is an XML snippet representing a storyboard merge conflict. The XML includes markers for the HEAD of one branch and the tip of another, along with conflict markers (====) and a commit hash at the bottom.

```
1 <<<<< HEAD
2 <segue reference="kXa-Mw-CAj"/>
3 <segue reference="TDo-lS-nUS"/>
4 <segue reference="hJU-8t-Kde"/>
5 <segue reference="haI-hu-Unh"/>
6 <segue reference="2ra-9a-Rv0"/>
7 <segue reference="ixW-dA-JnA"/>
8 =====
9 <segue reference="BwM-Nh-uZ9"/>
10 <segue reference="YWK-Ch-1fU"/>
11 <segue reference="haI-hu-Unh"/>
12 <segue reference="TDo-lS-nUS"/>
13 <segue reference="hJU-8t-Kde"/>
14 <segue reference="y7Z-qu-r0P"/>
15 >>>> e9a57872e96f17a8d2d785e4de0132e75229a262
```

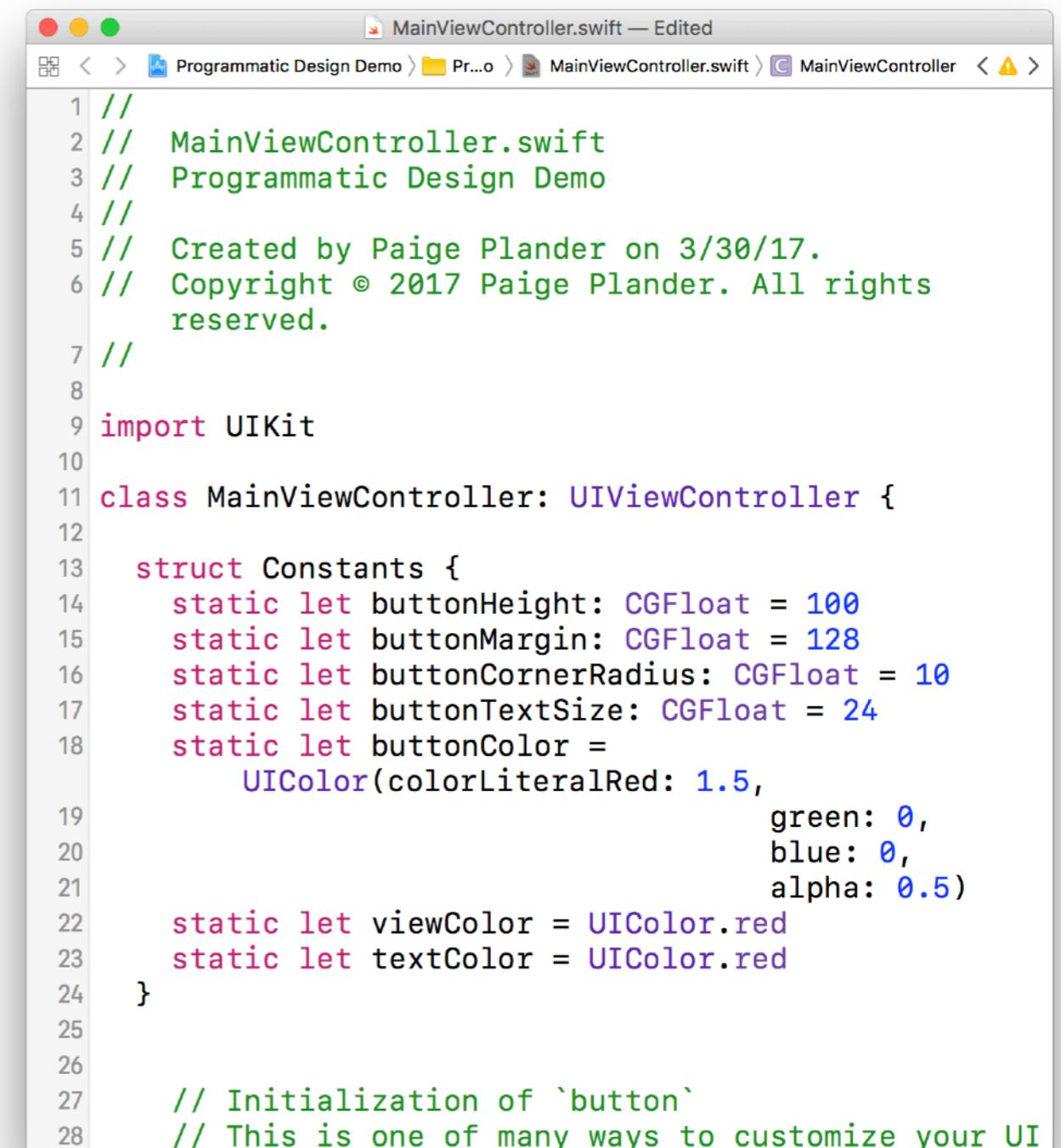
Storyboard : Problems

Cons of using Storyboard

Easy to get cluttered for larger scale applications

XML files are prone to merge conflicts when using version control (git)

No way to define layout constants or easy way to reuse UI layouts



The screenshot shows a Xcode editor window with the file 'MainViewController.swift' open. The code is written in Swift and defines a class 'MainViewController' that inherits from 'UIViewController'. It includes a 'Constants' struct with static let properties for button height, margin, corner radius, text size, and colors. The code also includes a note about button initialization and customization.

```
// MainViewController.swift
// Programmatic Design Demo
//
// Created by Paige Plander on 3/30/17.
// Copyright © 2017 Paige Plander. All rights reserved.

import UIKit

class MainViewController: UIViewController {

    struct Constants {
        static let buttonHeight: CGFloat = 100
        static let buttonMargin: CGFloat = 128
        static let buttonCornerRadius: CGFloat = 10
        static let buttonTextSize: CGFloat = 24
        static let buttonColor =
            UIColor(colorLiteralRed: 1.5,
                    green: 0,
                    blue: 0,
                    alpha: 0.5)

        static let viewColor = UIColor.red
        static let textColor = UIColor.red
    }

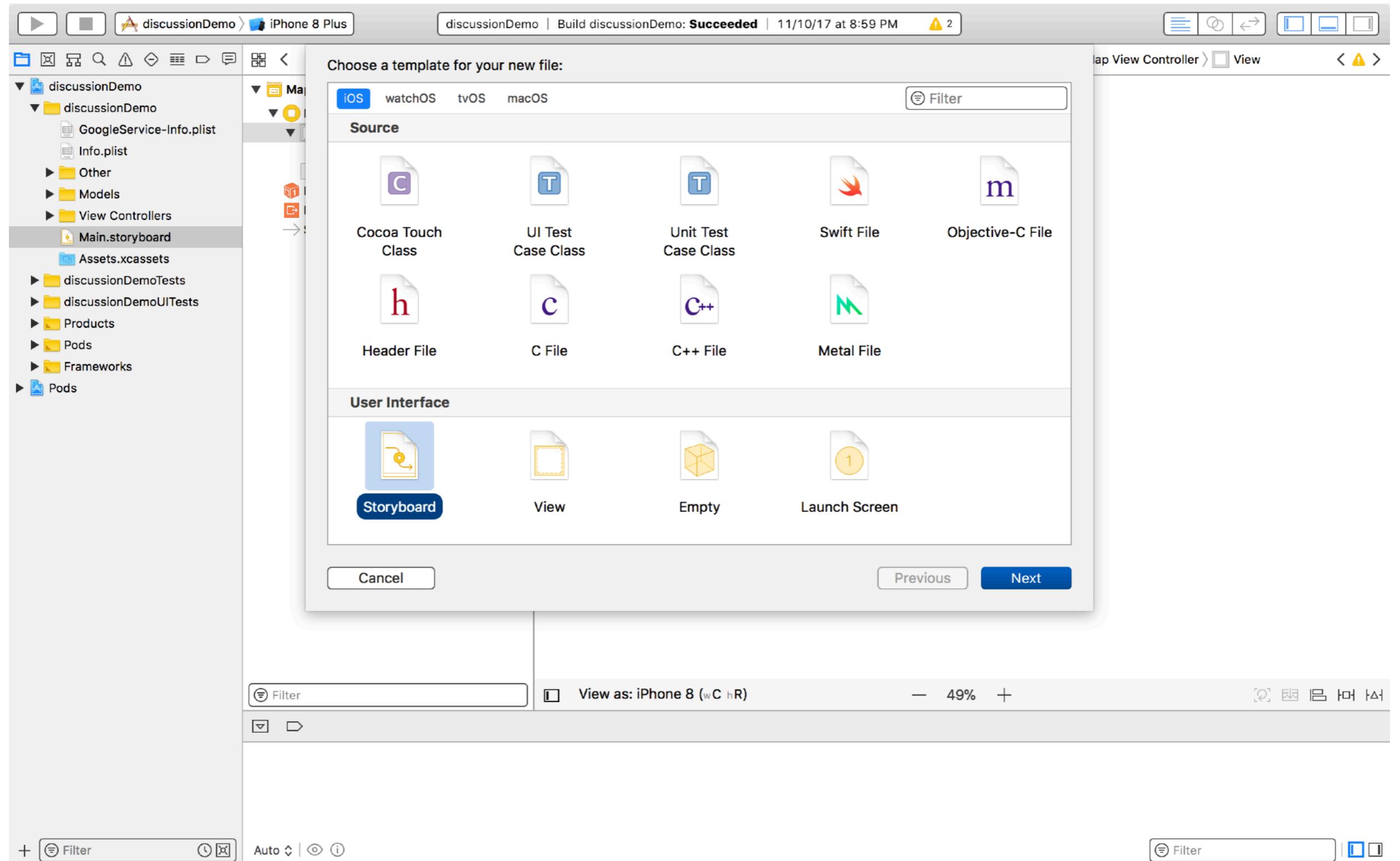
    // Initialization of `button`
    // This is one of many ways to customize your UI
}
```

Storyboards References

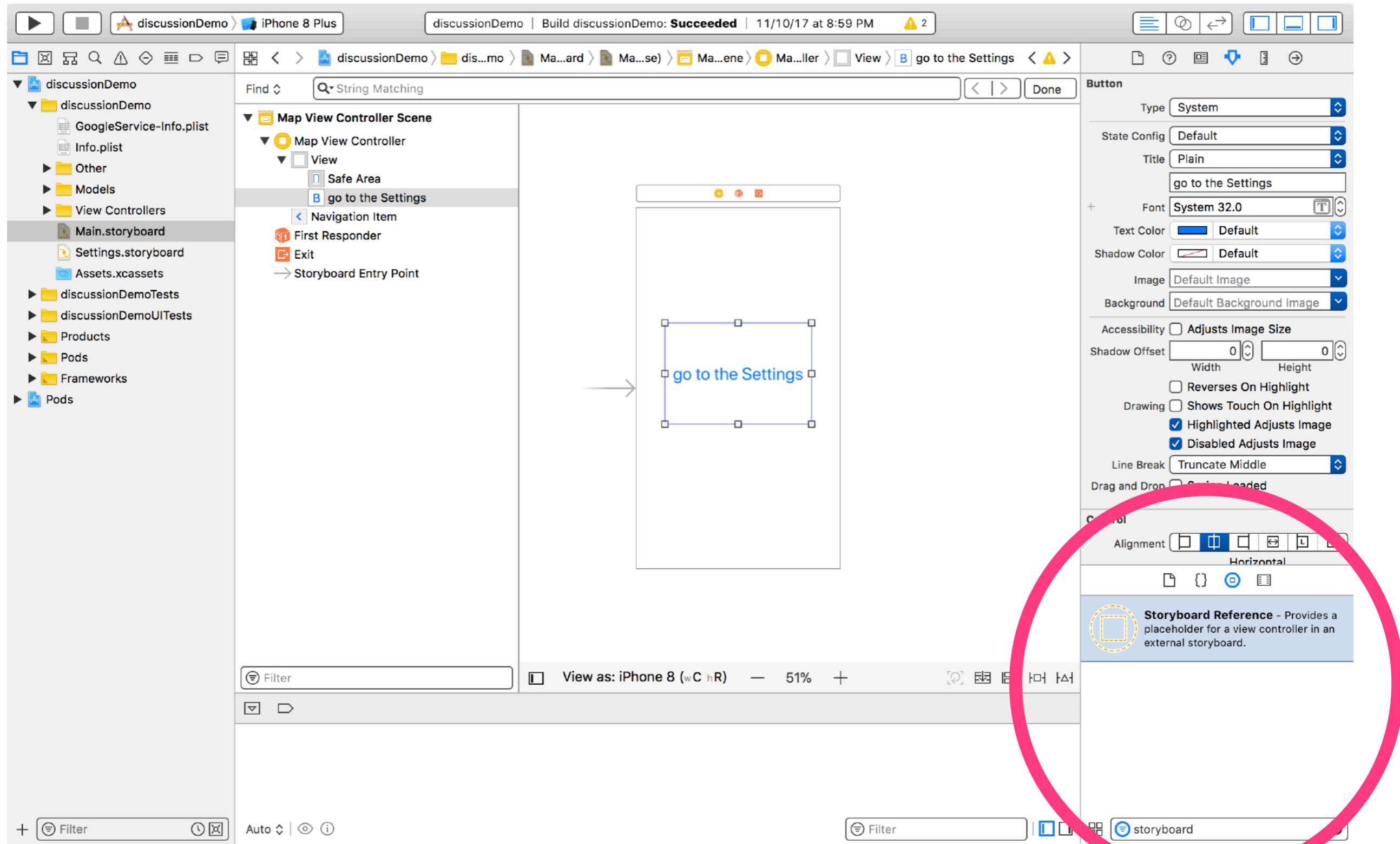
If you have a large project, break it up into several different Storyboard files.

Use “Storyboard References” to transition between Storyboards

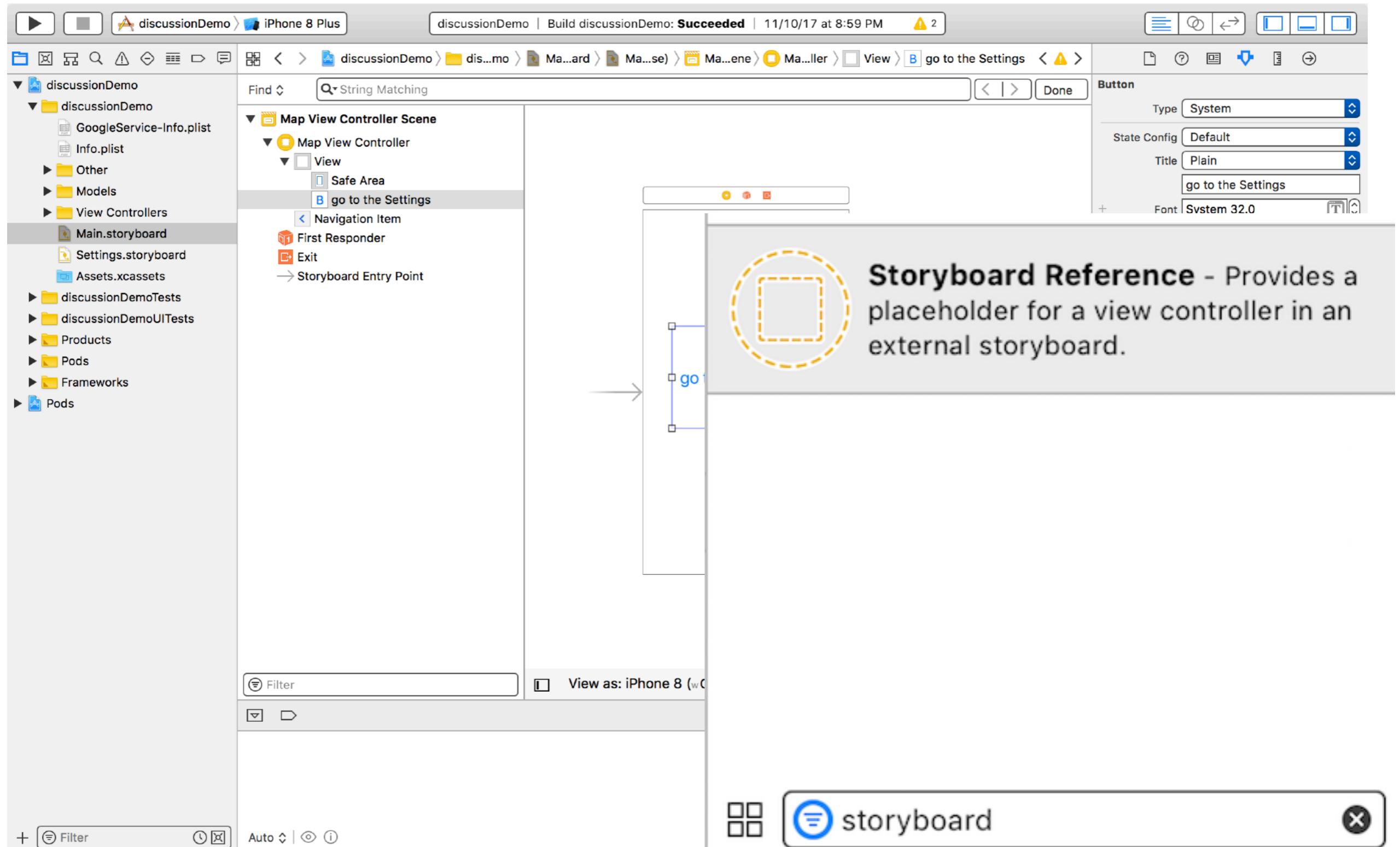
Create the storyboard file



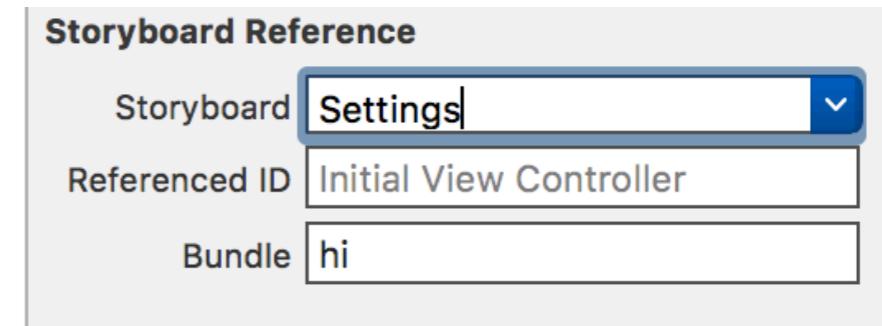
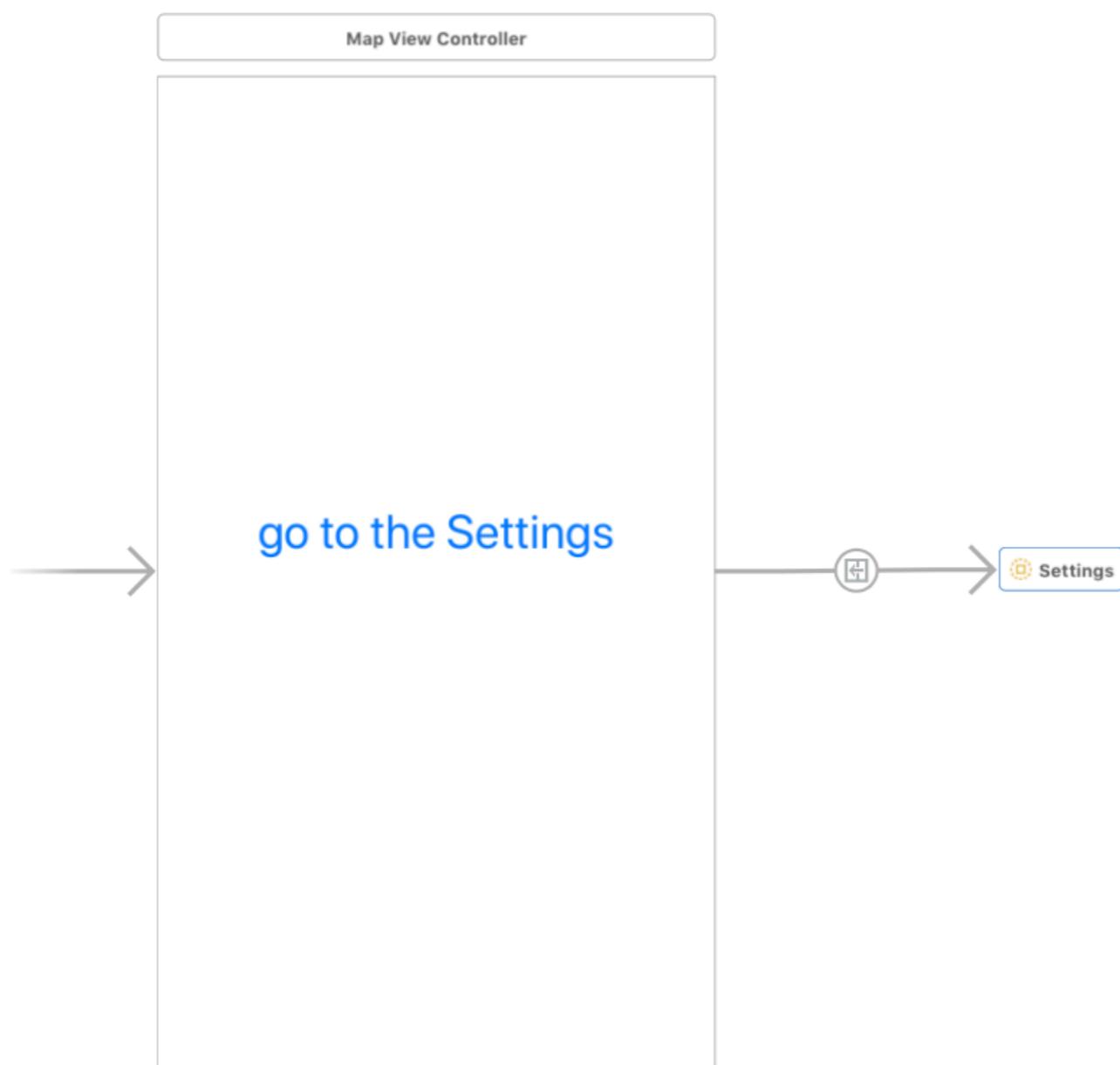
Create a Storyboard Reference



Create a Storyboard Reference



Create a Storyboard Reference



create a
segue as
you would
for view
controllers

Storyboards References

Storyboard references are powerful! You can create references to

- storyboards in different bundles,
- the same storyboard, and
- specific view controllers within other storyboards

Editor > Refactor Storyboard will automatically create a new storyboard file for selected view controllers

Programmatic Design

No Storyboard Needed

UI elements (buttons / labels / views) are instantiated in code and added as subviews

Pros

Better for version control

Scalable

Less limited

Cons

Steeper learning curve

Lots of boiler plate code

Not visual

Summary - iOS UI Implementation

Programmatic Design vs Storyboard

... so which one is better?

Depends on what you are creating

Often times, a combination of both may be
the best solution

(i.e - instantiate views in storyboard, edit
them programmatically)

Programmatic Design

Some important classes:

UIWindow - provides the backdrop for your app's content (usually only one per app)

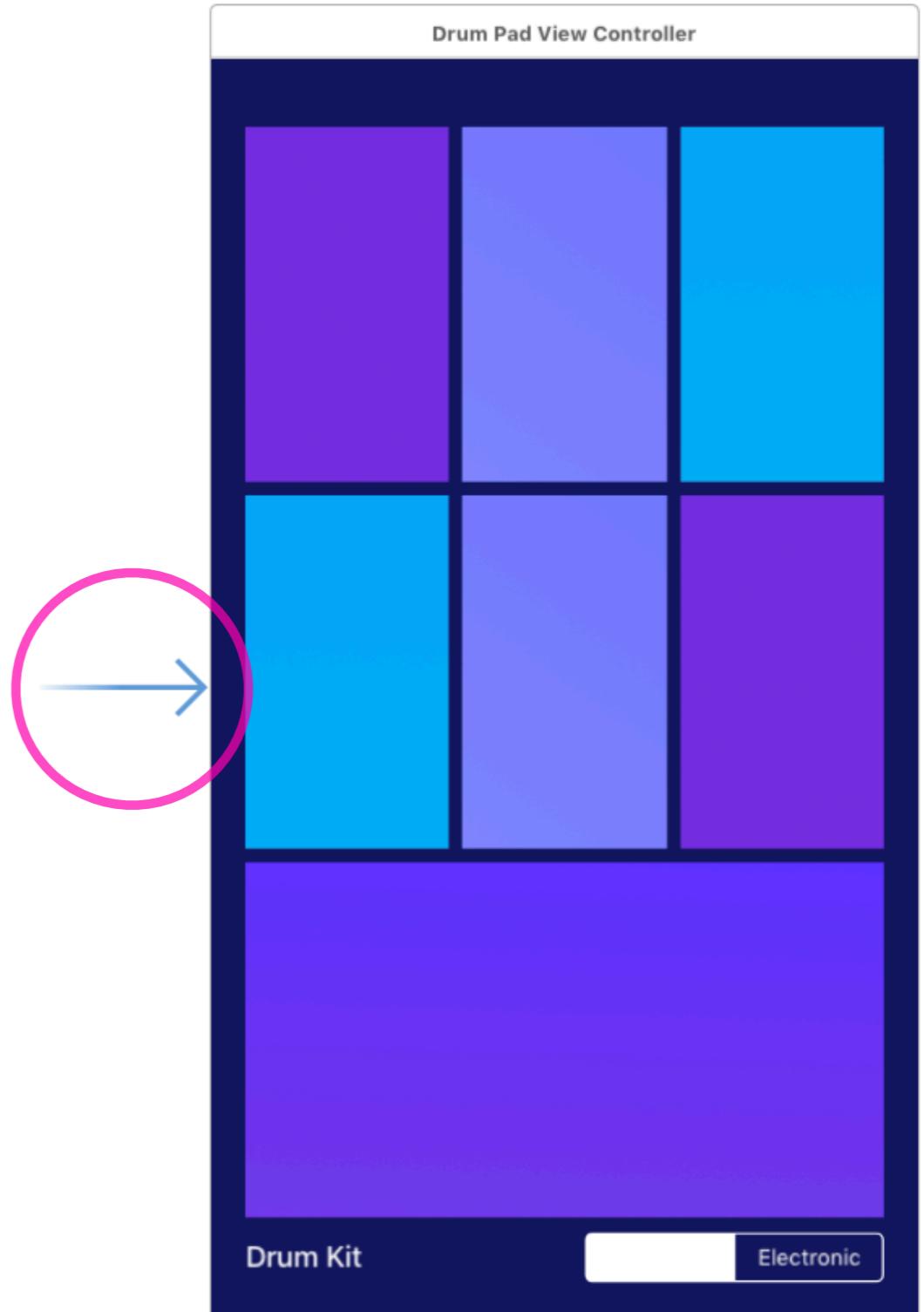
UIScreen - defines the properties of the user's device (get the bounds of user's device using `UIScreen.main.bounds`)

UIViewController - manages a set of UIView's

Programmatic Design (with no Storyboard)

To get rid of your storyboard, delete both the **Main.storyboard** file and it's reference in **Info.plist**

To programmatically set the initial view controller, you'll need to edit your **AppDelegate.swift**. This is equivalent to setting the "initial view controller" property in Storyboard (represented by the arrow icon)



Programmatic Design (with no Storyboard)

```
import UIKit  
  
@UIApplicationMain  
class AppDelegate: UIResponder, UIApplicationDelegate {  
  
    var window: UIWindow?  
  
    func application(_ application: UIApplication,  
                    didFinishLaunchingWithOptions launchOptions:  
                    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
  
        // this code executes when your app is opened for the  
        // first time  
  
        return true  
    }  
    ...
```

Found in AppDelegate.swift

Setting your initial View Controller
Programmatically (Example)

Programmatic Design (with no Storyboard)

```
import UIKit
```

Found in AppDelegate.swift

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
var window: UIWindow?
```

```
func application(_ application: UIApplication,  
                 didFinishLaunchingWithOptions launchOptions:  
                     [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
    window = UIWindow(frame: UIScreen.main.bounds)  
    let myViewController = MyViewController()  
    window?.rootViewController = myViewController  
    window?.makeKeyAndVisible()  
    return true  
}
```

```
...
```

The **window** displays the app's content on
the device's main screen.

Programmatic Design (with no Storyboard)

```
import UIKit
```

Found in AppDelegate.swift

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
    var window: UIWindow?
```

```
    func application(_ application: UIApplication,  
                    didFinishLaunchingWithOptions launchOptions:  
                    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
```

```
        window = UIWindow(frame: UIScreen.main.bounds)
```

```
        let myViewController = MyViewController()
```

```
        window?.rootViewController = myViewController
```

```
        window?.makeKeyAndVisible()
```

```
        return true
```

```
}
```

```
...
```

Set the window to be
the size of the user's screen

Programmatic Design (with no Storyboard)

```
import UIKit
```

Found in AppDelegate.swift

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
    var window: UIWindow?
```

```
    func application(_ application: UIApplication,
```

```
                  didFinishLaunchingWithOptions launchOptions:
```

```
                  [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
```

```
        window = UIWindow(frame: UIScreen.main.bounds)
```

```
        let myViewController = MyViewController()
```

```
        window?.rootViewController = myViewController
```

```
        window?.makeKeyAndVisible()
```

```
        return true
```

```
}
```

```
...
```

Instantiate a View Controller to be the
window's root view controller

Programmatic Design (with no Storyboard)

```
import UIKit
```

Found in AppDelegate.swift

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
    var window: UIWindow?
```

```
    func application(_ application: UIApplication,
```

```
                  didFinishLaunchingWithOptions launchOptions:
```

```
                  [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
```

```
        window = UIWindow(frame: UIScreen.main.bounds)
```

```
        let myViewController = MyViewController()
```

```
        window?.rootViewController = myViewController
```

```
        window?.makeKeyAndVisible()
```

```
        return true
```

```
}
```

```
...
```

Set the window's
root view controller property

Programmatic Design (with no Storyboard)

```
import UIKit
```

Found in AppDelegate.swift

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
    var window: UIWindow?
```

```
    func application(_ application: UIApplication,  
                    didFinishLaunchingWithOptions launchOptions:  
                    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
        window = UIWindow(frame: UIScreen.main.bounds)  
        let myViewController = MyViewController()  
        window?.rootViewController = myViewController  
        window?.makeKeyAndVisible()  
        return true  
    }
```

```
...
```

Make the window visible to the user

Programmatic Design (with no Storyboard)

```
import UIKit
```

Found in AppDelegate.swift

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
    var window: UIWindow?
```

```
    func application(_ application: UIApplication,
```

```
                  didFinishLaunchingWithOptions launchOptions:
```

```
                  [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
```

```
        window = UIWindow(frame: UIScreen.main.bounds)
```

```
        let myViewController = MyViewController()
```

```
        window?.rootViewController = myViewController
```

```
        window?.makeKeyAndVisible()
```

```
        return true
```

```
}
```

```
...
```

Now the user will see “myViewController”
upon opening this application

Programmatic Design

To create UI elements programmatically, you'll need to do the following:

1. Instantiate the UI element

i.e. `let myButton = UIButton()`

2. Add the view as a subview to your superview using `addSubview`

i.e. `superview.addSubview(myButton)`

3. Set the position and size of your view either using **frames** or **layout constraints**

Programmatic Design : Example

Suppose we wanted to add a button to our view

in Storyboard

Drag and drop a UIButton
into your storyboard from
the Object Library

Customize using
Attributes Inspector

Setup Constraints

Programmatically

```
let myBtn = UIButton(frame:  
                      CGRect(x: 50,  
                              y: 100,  
                              width: 200,  
                              height: 100))  
  
myBtn.setTitle("Click me!",  
              for: .normal)  
myBtn.backgroundColor = .red  
view.addSubview(myBtn)
```

Views and Geometry

Views / UIView

The `UIView` class defines a rectangular area on your user's screen

This area can be used for managing content, holding other views, registering touch events, etc.

Classes like `UIImageView` and `UILabel` are special types of `UIView's` (they both subclass `UIView`)

CGRect and CGPoint

CGRect - defines position and size

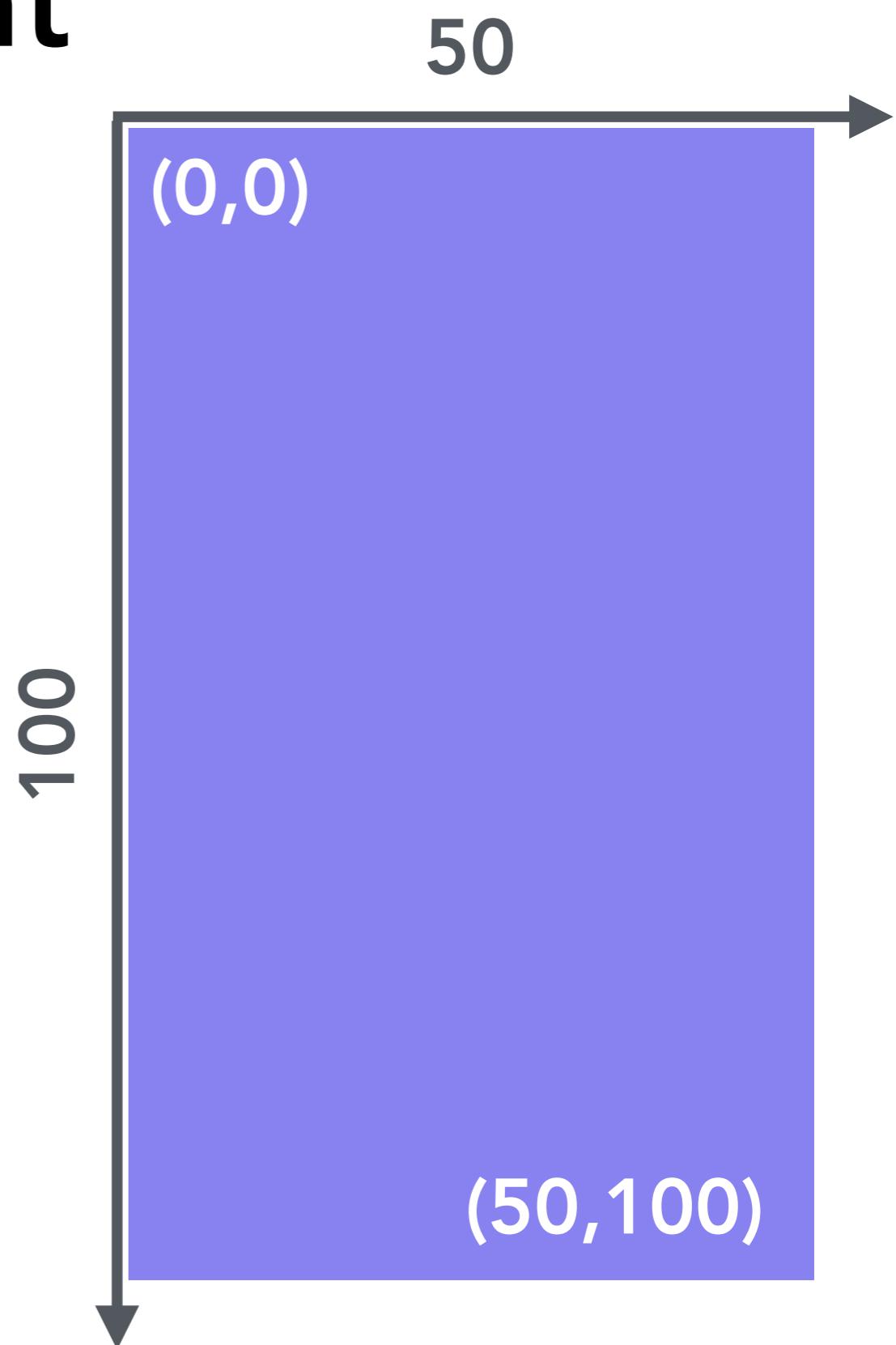
```
CGRect(x: 0, y: 0,  
       width: 100,  
       height: 100)
```

CGPoint - defines a position

```
CGPoint(x: 0, y: 0)
```

CGSize - defines a size

```
CGSize(width: 100,  
       height: 100)
```



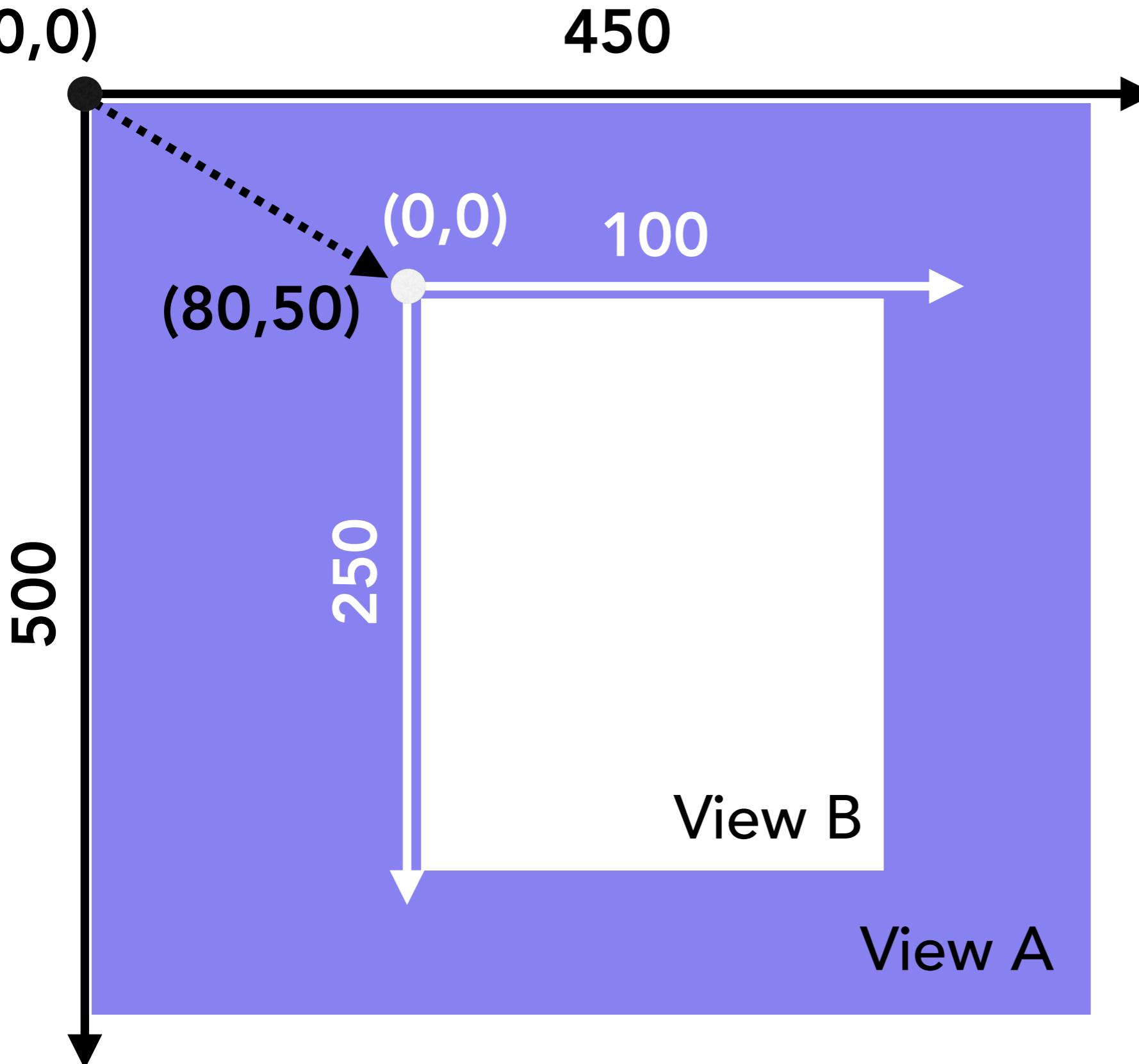
UIView : Geometry

A `UIView`'s geometry is defined by the view's `frame`, `bounds`, and `center` properties

`frame`: `CGRect` - the coordinates and dimensions of the view **in the coordinate system of its superview**

`bounds`: `CGRect` - the coordinates and dimensions of the view **relative to itself**

`center`: `CGPoint` - the center of the view
(used for positioning of the view)



View A frame

$x, y = (?,?)$

width = 450

height = 500

View A bounds

$x, y = (?,?)$

width = 450

height = 500

View B frame

$x, y = (?,?)$

width = 100

height = 250

View B bounds

$x, y = (?,?)$

width = 100

height = 250

frame: uses the **coordinate system of its superview**

bounds: uses coordinates **relative to itself**

Positioning / Sizing Views

Two ways of setting the size and position
of your views programmatically

1. Using **frames / bounds** (`initWithFrame`
`CGRect`, `CGPoint`)
2. Using **AutoLayout** (`NSLayoutConstraints`)

Positioning and Sizing Views Using Frames

```
let myFrame = CGRect(x: 0, y: 0,  
width: UIScreen.main.bounds.width - 16,  
height: 100)
```

```
let myButton = UIButton(frame: myFrame)
```

```
myButton.center = view.center
```

```
view.addSubview(myButton)
```

Positioning and Sizing Views with AutoLayout

```
btn.translatesAutoresizingMaskIntoConstraints = false

// constraints to center the button horizontally in the view
let myConstraints = [
    btn.centerXAnchor.constraint(equalTo: view.centerXAnchor),
    btn.centerYAnchor.constraint(equalTo: view.centerYAnchor),
    btn.leadingAnchor.constraint(equalTo: view.leadingAnchor,
                                 constant: 8),
    btn.trailingAnchor.constraint(equalTo: view.trailingAnchor,
                                 constant: 8),
    btn.heightAnchor.constraint(equalToConstant: 100)
]

NSLayoutConstraint.activate(myConstraints)
```

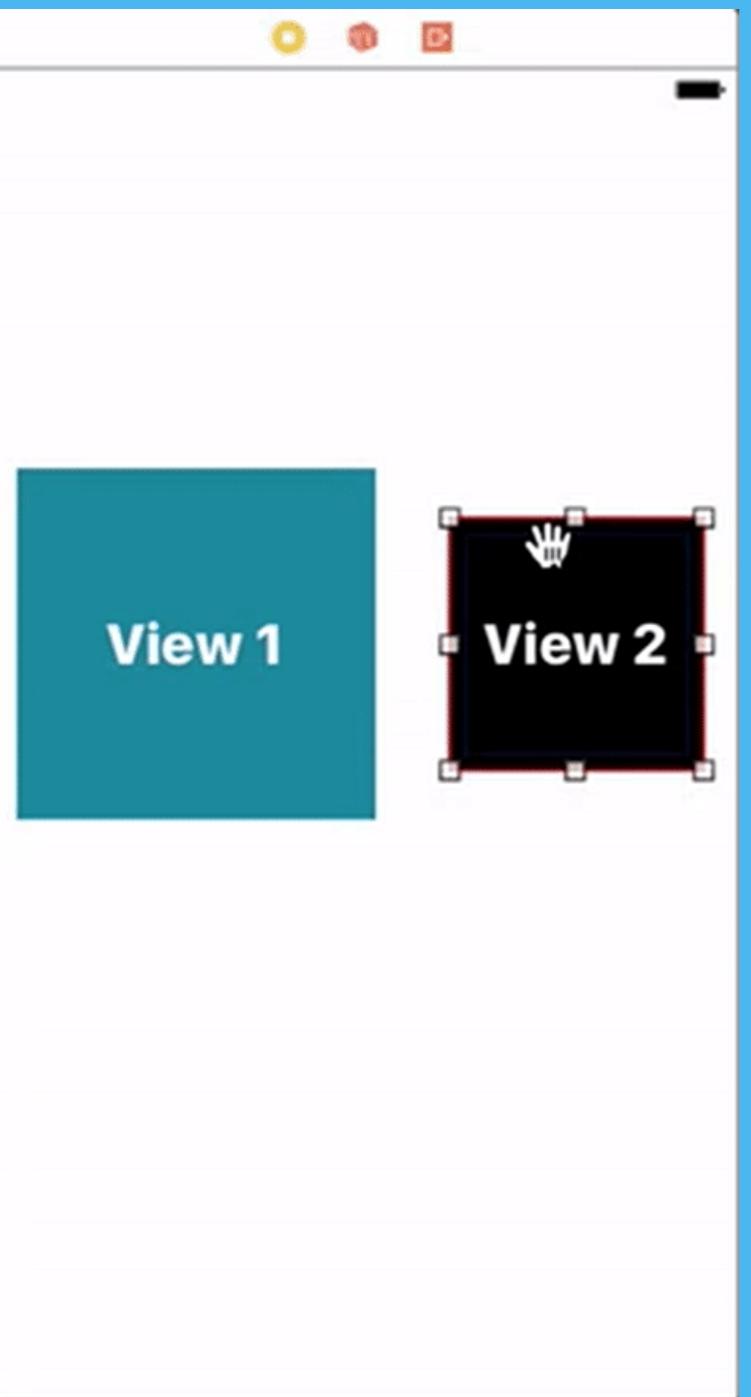
In this example, we create a list of constraints, then batch activate them (rather than doing it one by one)

Programmatic AutoLayout

Layout Anchors

```
let constraint =  
    view2.leadingAnchor.constraint(  
        equalTo: view1.trailingAnchor,  
        constant: 8)
```

```
constraint.isActive = true
```



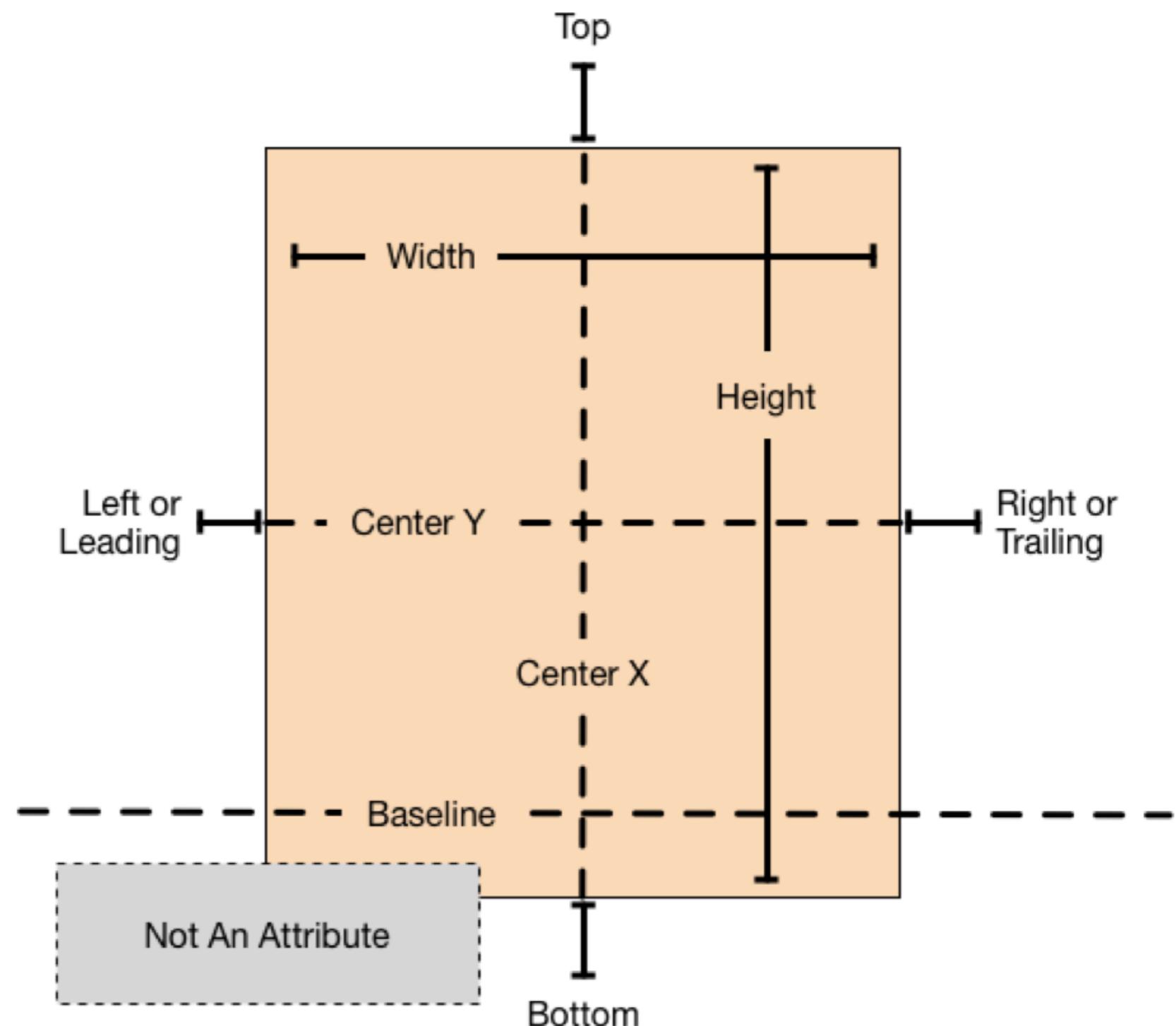
Equivalent
Storyboard Example

In both of these examples,
the spacing between
views is set to 8 points

Programmatic AutoLayout

Layout Anchor Properties

Use these properties to create relationships between views



Presenting view controllers

To present new view controllers (i.e. show segue)

```
func present(_ viewControllerToPresent:  
             UIViewController,  
             animated flag: Bool,  
             completion: ((()) -> Void)? = nil)
```

Presenting VC's (with nav controllers)

To add vc's to a navigation controller stack

- instantiate a navigation controller
- add viewcontrollers to the nav controllers “viewcontrollers” array or set a root view controller
- navigate by “pushing” view controllers, or popping to the root view controller

Presenting VC's (with nav controllers)

adding a view controller to the stack (push)

```
let nextVC = ViewController2()  
navigationController?.pushViewController(  
    nextVC, animated: true)
```

Note: navigationController will be nil if the current view controller is not within it's stack

Presenting VC's (with nav controllers)

pop (go back) to the root of you navigation controller

```
navigationController?.popToRootViewController  
(animated: true)
```

Presenting VC's (with nav controllers)

pop (go back) to a different view controller,
that isn't the root

```
navigationController?.popToViewController(pre  
vVC, animated: true)
```

note: the VC you are navigating to must
already be in the nav controller's stack

Demo

<https://github.com/paigeplan/lec8>