

iOS DeCal : Lecture 8

UI / UX and Programmatic Design

April 4, 2017

Announcements - 4/4

Snapchat Clone Part 2 due next Tuesday

Custom App Checkins (this Thurs during lab)

Be prepared to talk to your assigned TA about your progress

Remember to check your Attendances

Can be found on Piazza

Let us know if we made a mistake!

Overview : Today's Lecture

iOS Human Interface Guidelines (HIG)

Programmatic Design

iOS Human Interface Guidelines

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

Overview ▾

Design Principles

What's New in iOS 10

Interface Essentials

Interaction

Features

Visual Design

Graphics

UI Bars

UI Views

UI Controls

Extensions

Technologies

Resources



iOS Human Interface Guidelines

New in iOS 10

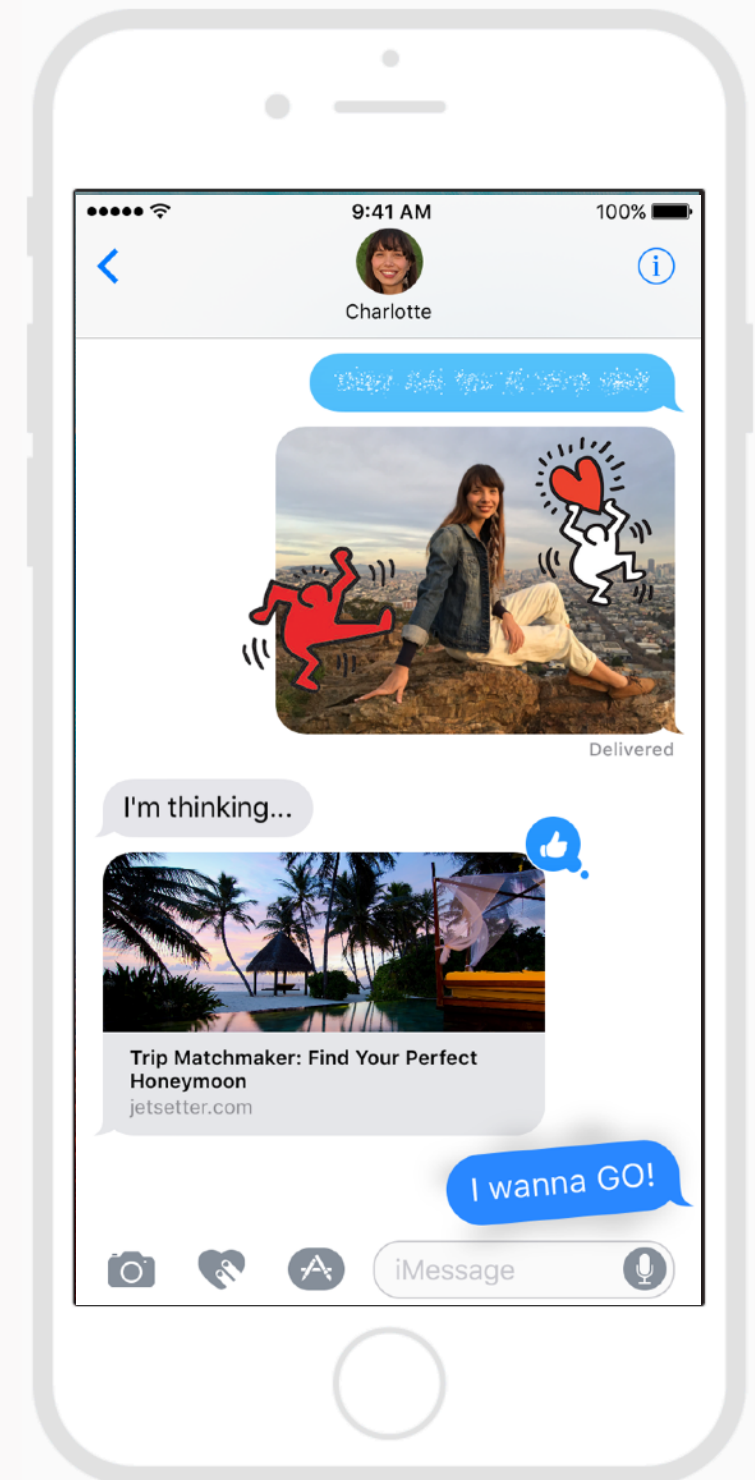
iMessage Integration

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

New in iOS 10

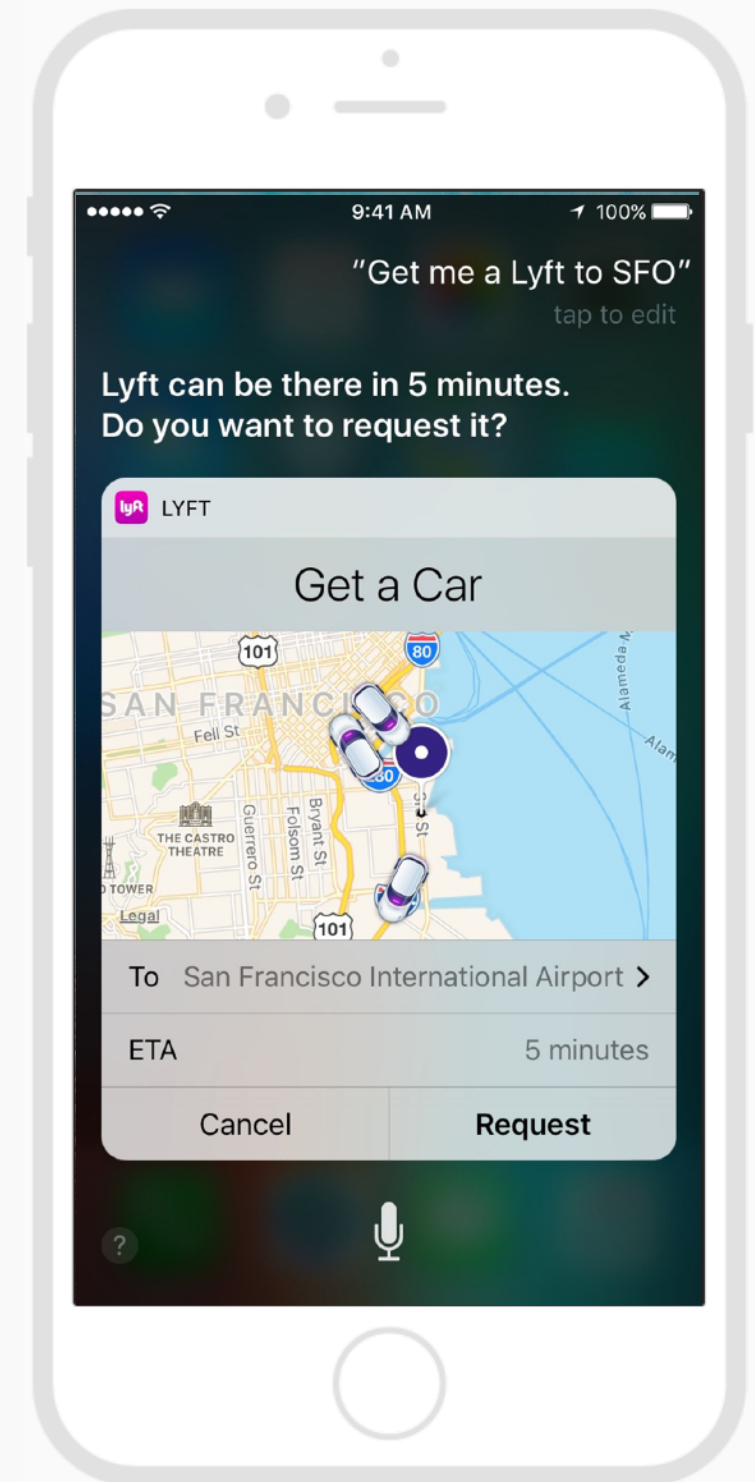
Integration with Siri

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

New in iOS 10

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

New in iOS 10

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

New in iOS 10

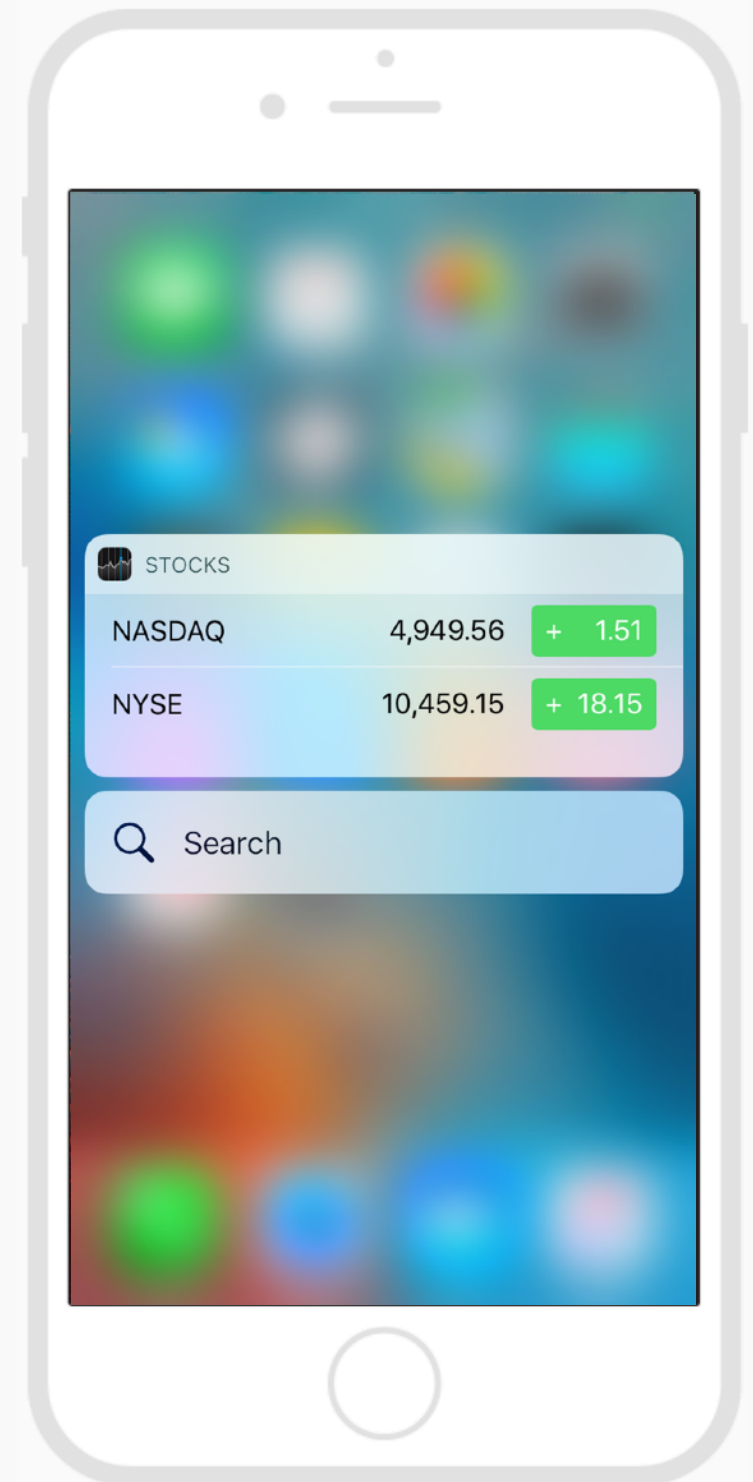
Search Widgets

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

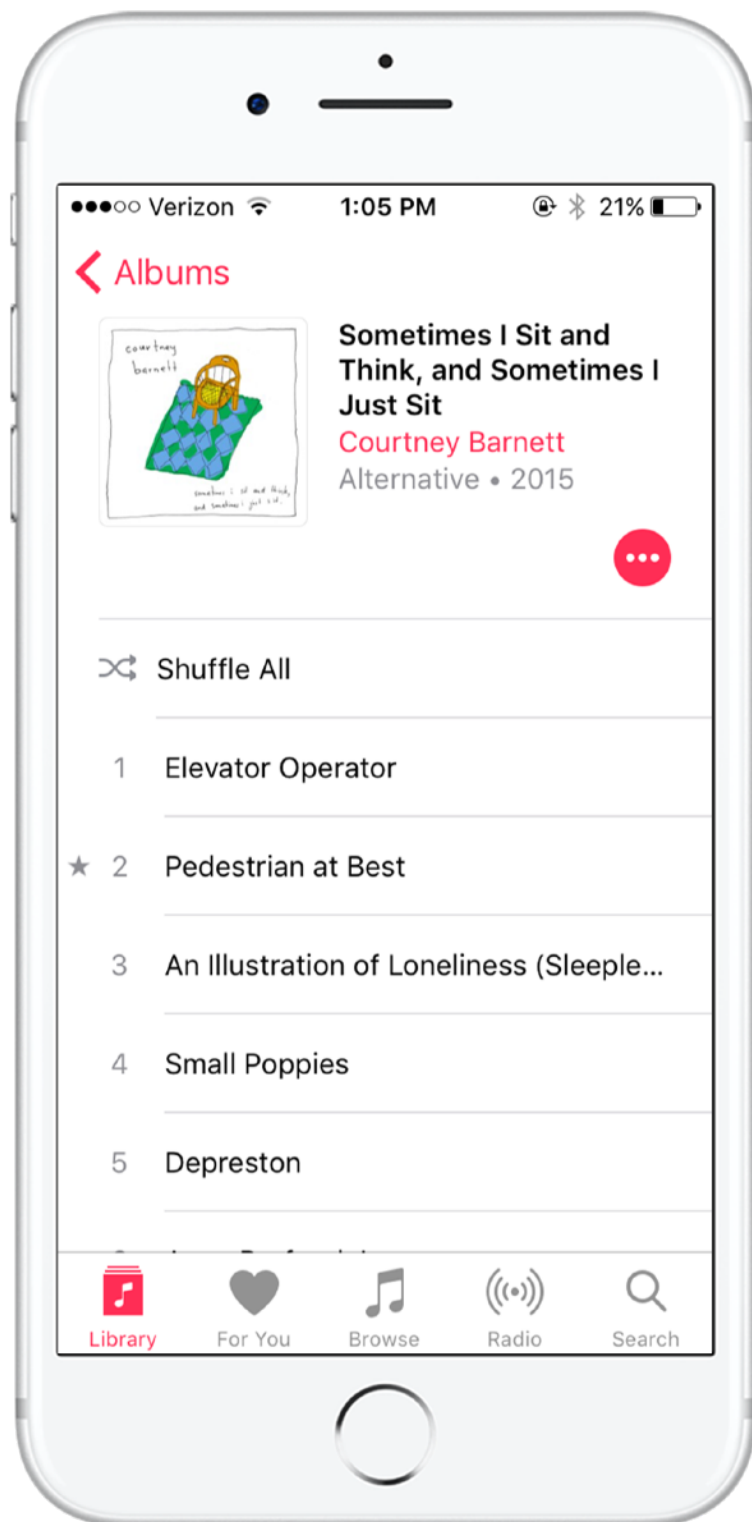
Very customizable (can add buttons, images, layout customization, etc.)

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

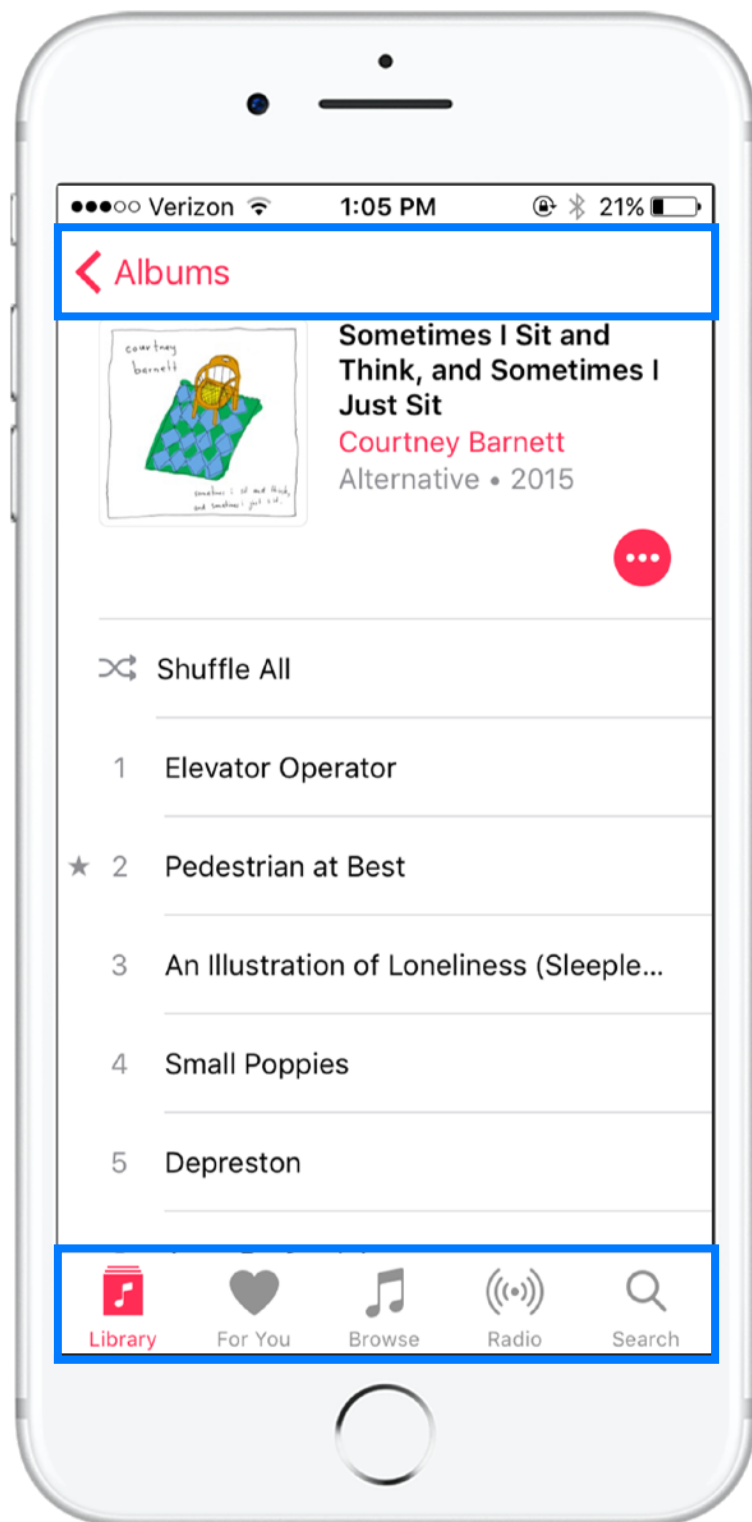
Views

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

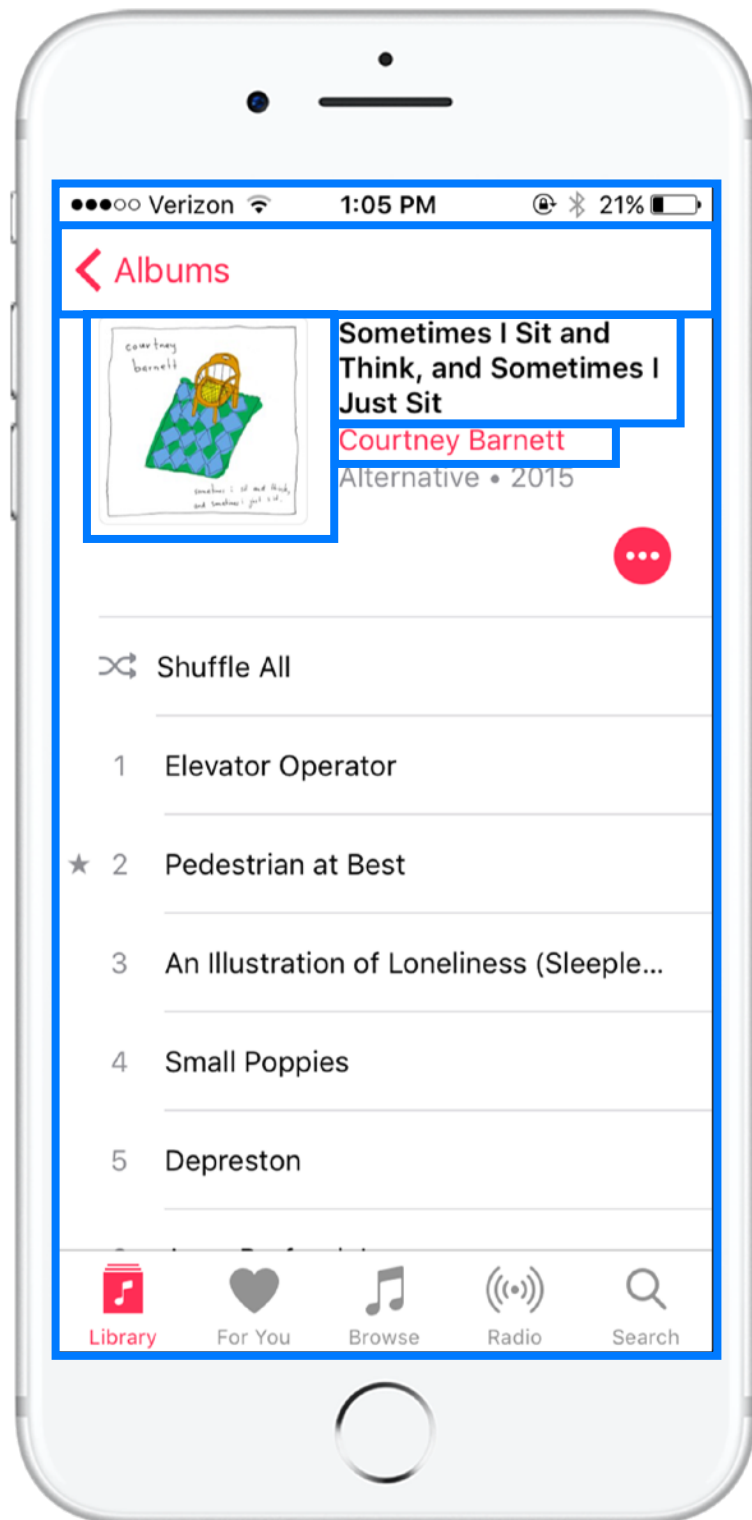
Views

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

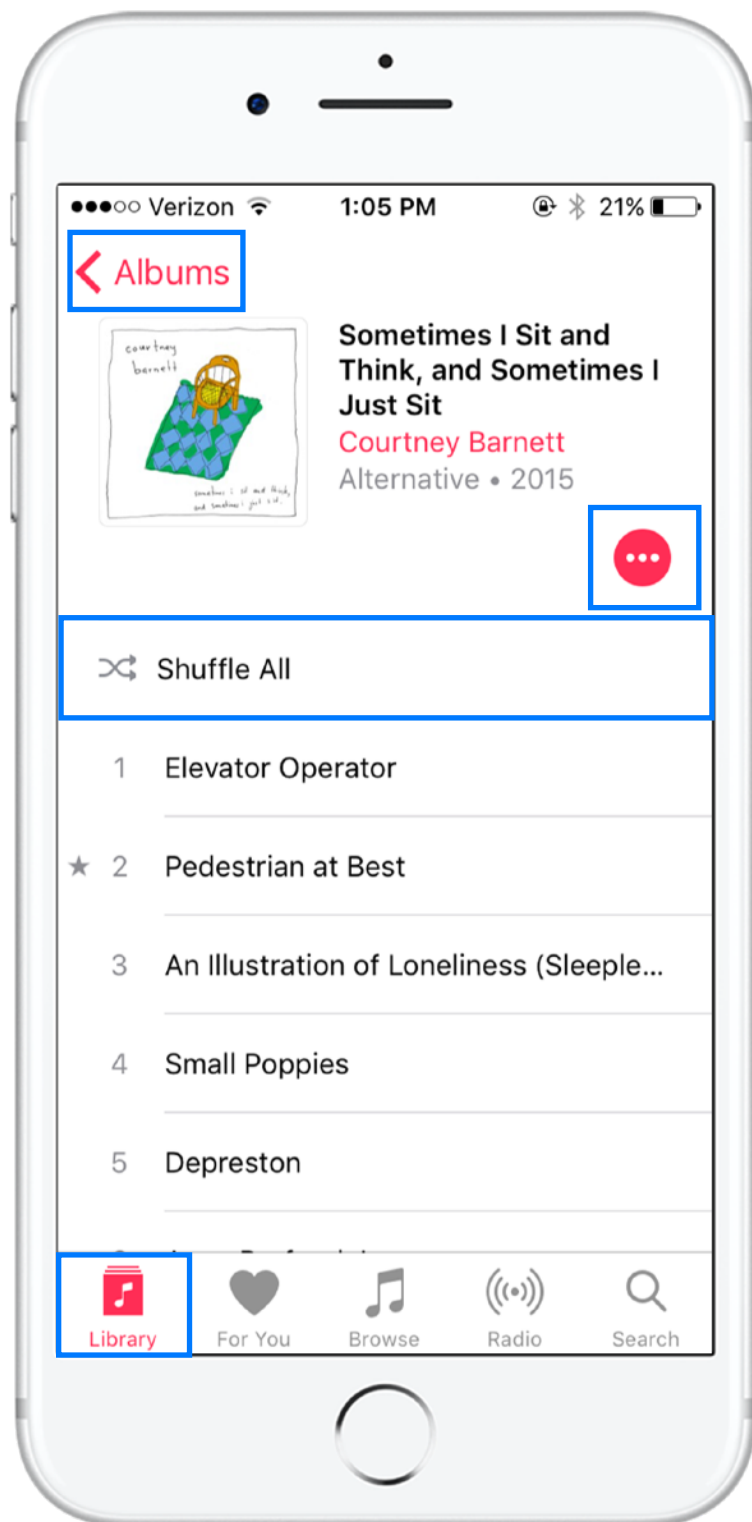
Views

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

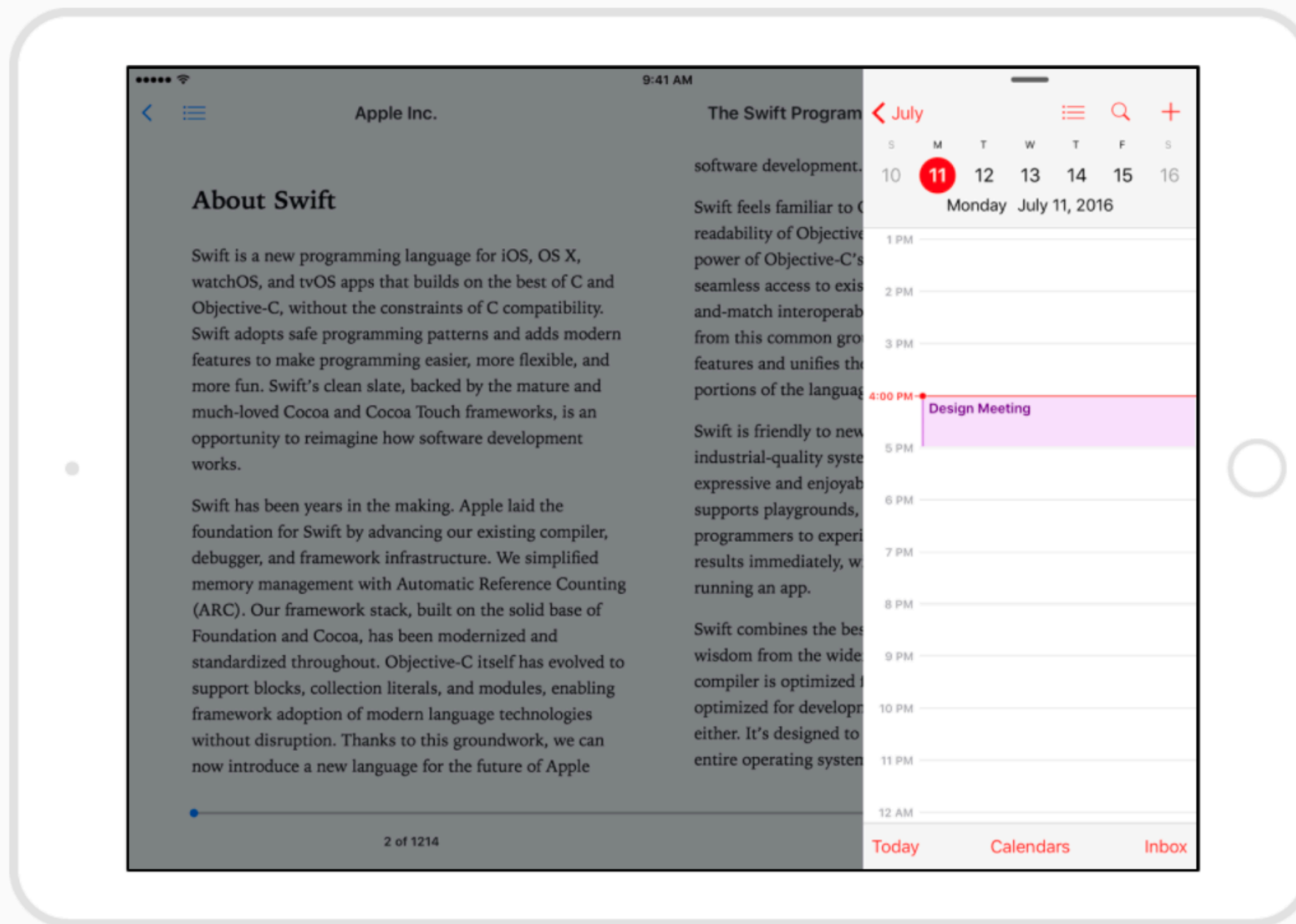
Views

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

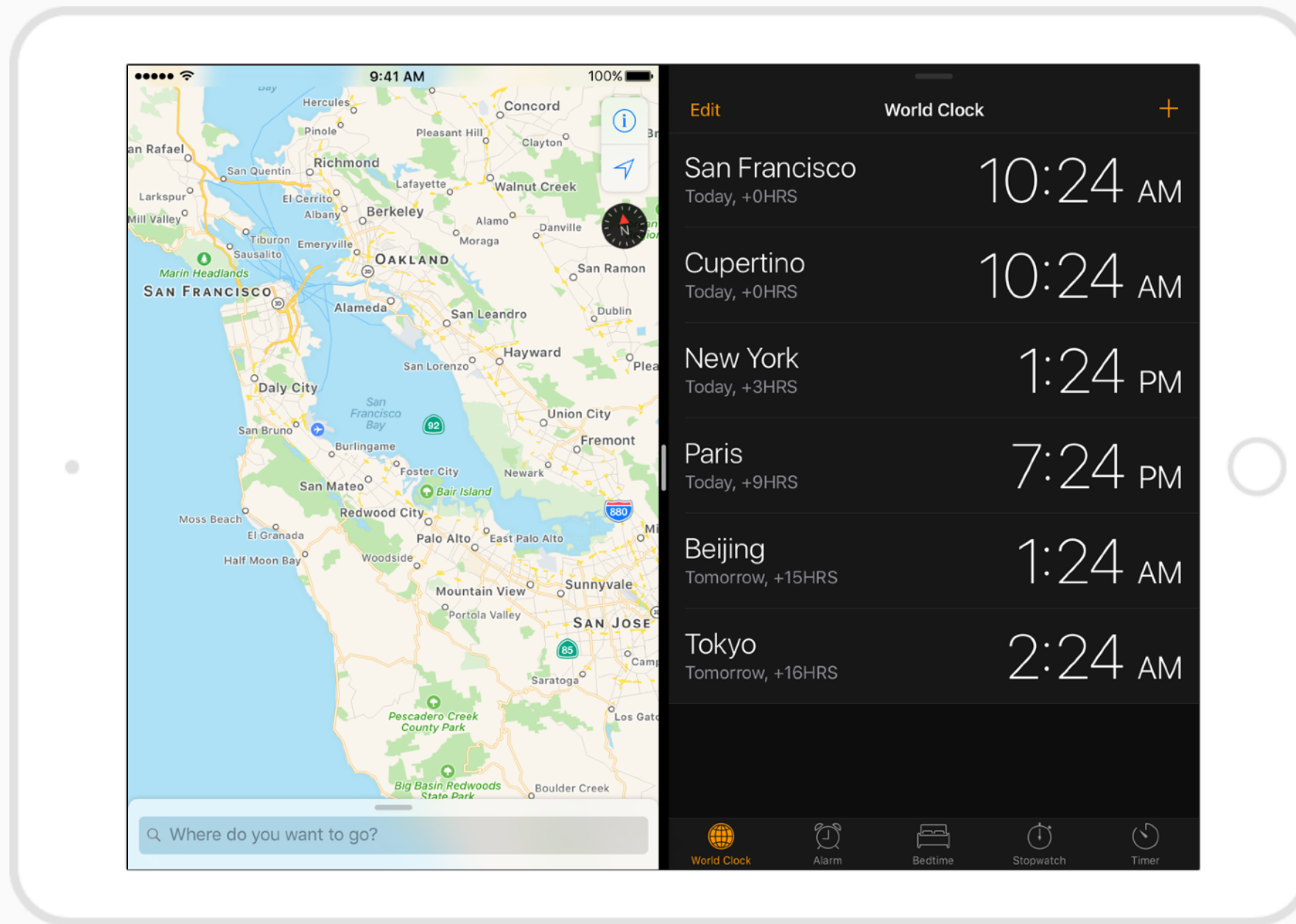
Buttons, text fields, segmented controls, etc.

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

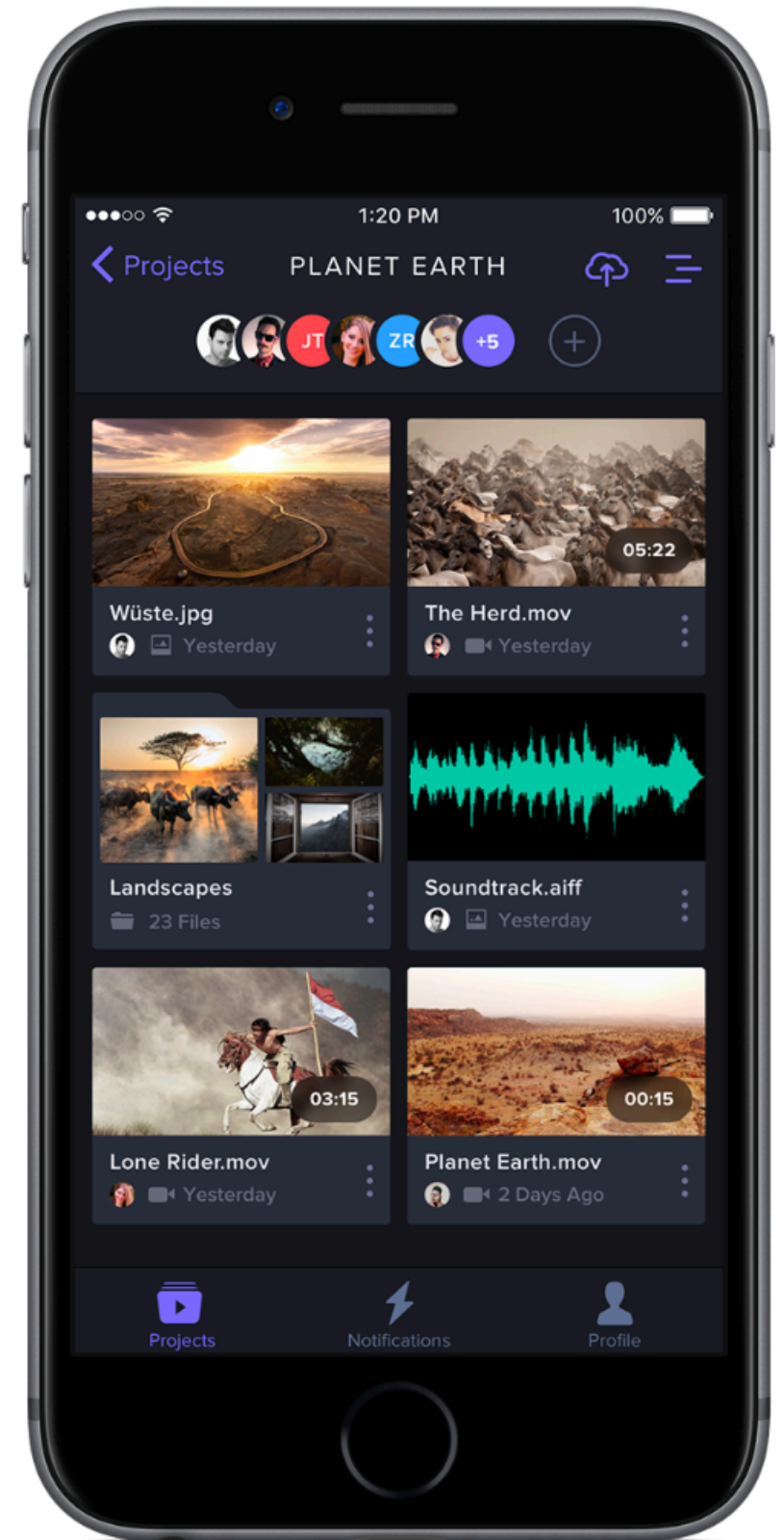
While it is important to have an app “image” or “brand”, avoid over-using logos, icon images, etc.

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

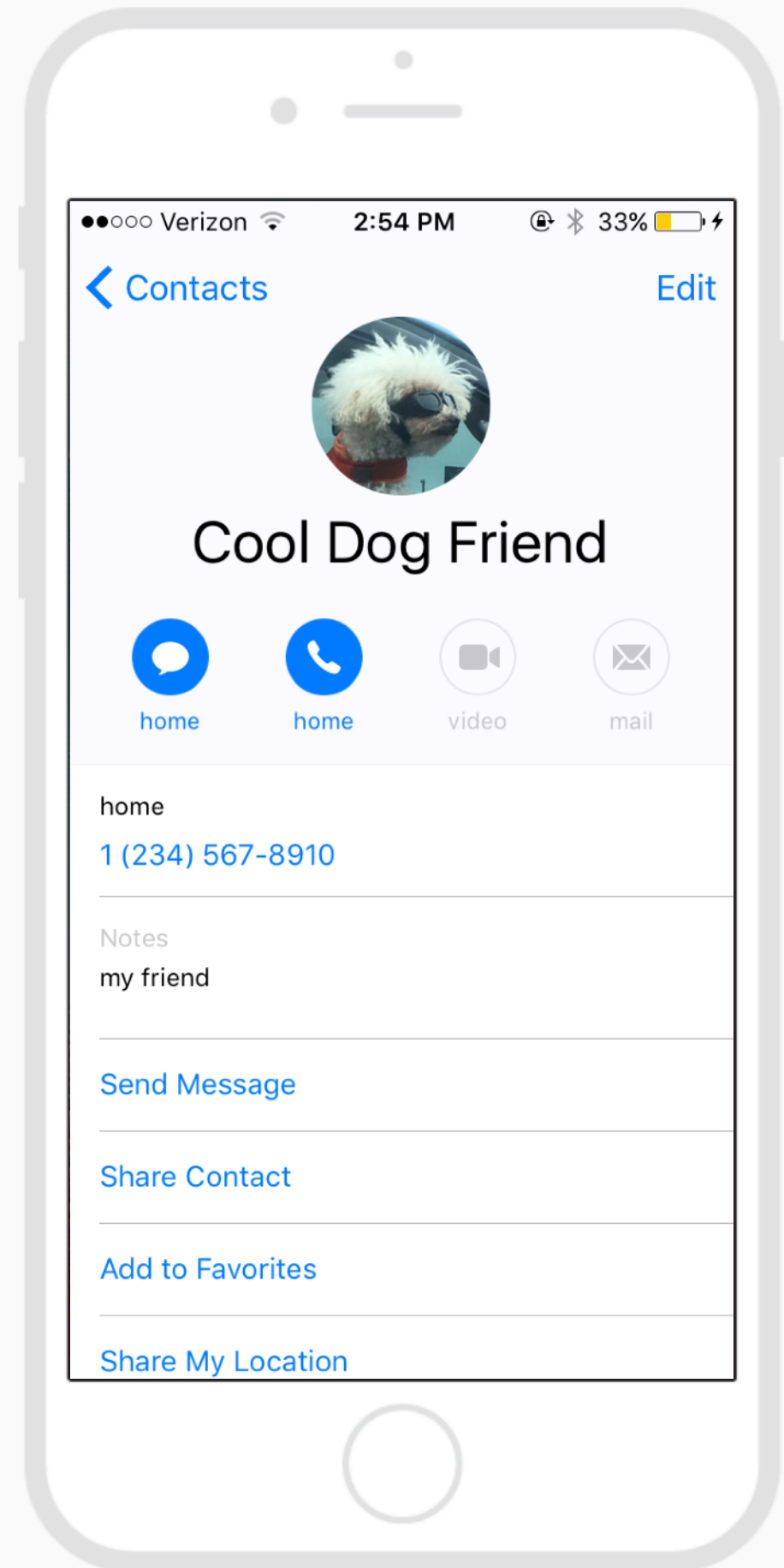
<div>R 255 G 59 B 48</div>	<div>R 255 G 149 B 0</div>
Red	Orange
<div>R 255 G 204 B 0</div>	<div>R 76 G 217 B 100</div>
Yellow	Green
<div>R 90 G 200 B 250</div>	<div>R 0 G 122 B 255</div>
Teal Blue	Blue
<div>R 88 G 86 B 214</div>	<div>R 255 G 45 B 85</div>
Purple	Pink

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

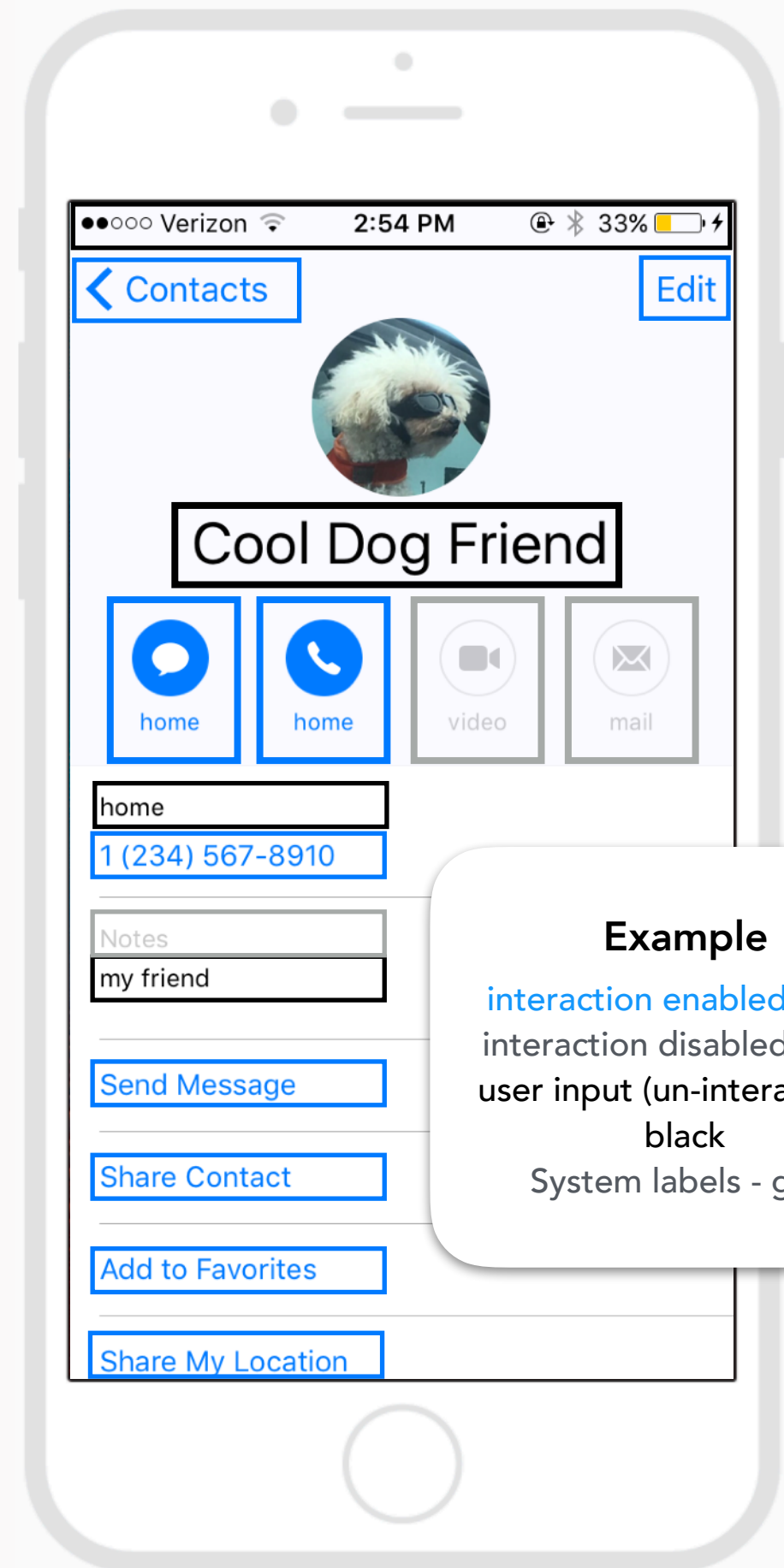


iOS HIG : Color

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

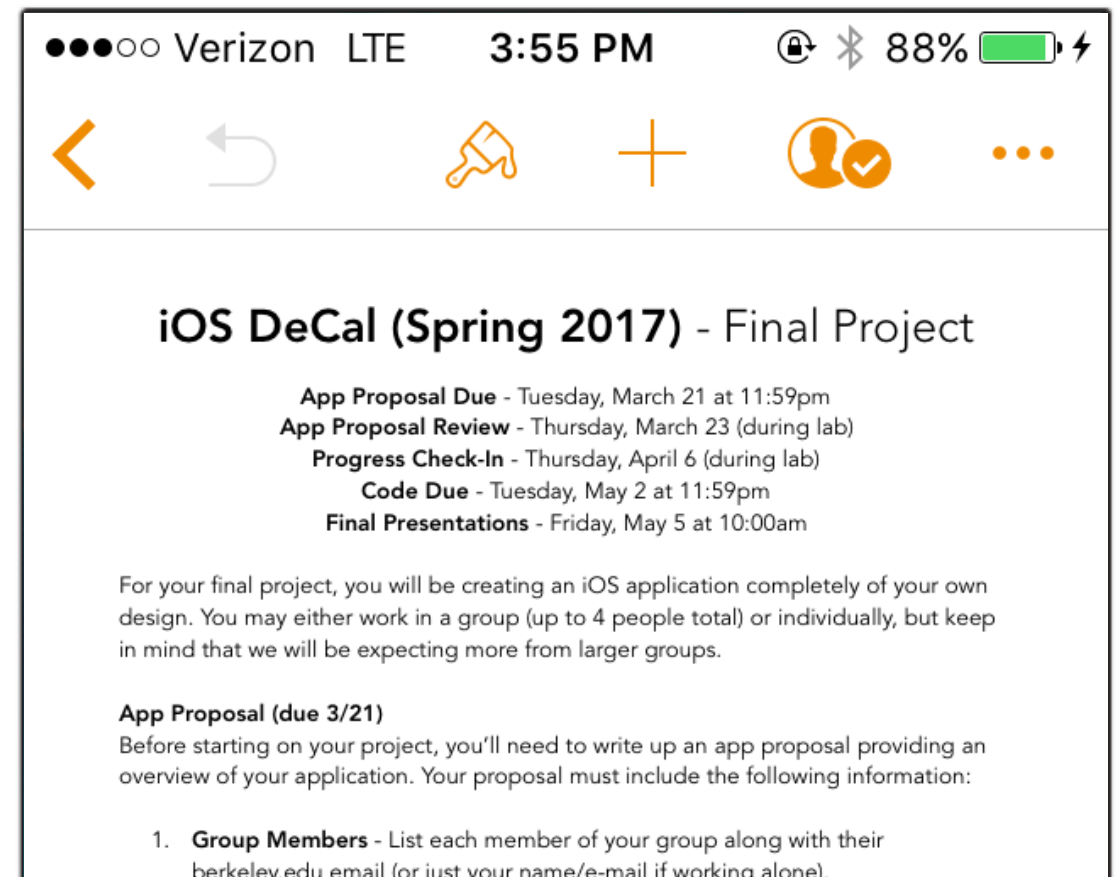
Distinguish between interactive and un-interactive UI elements using color

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



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)
```

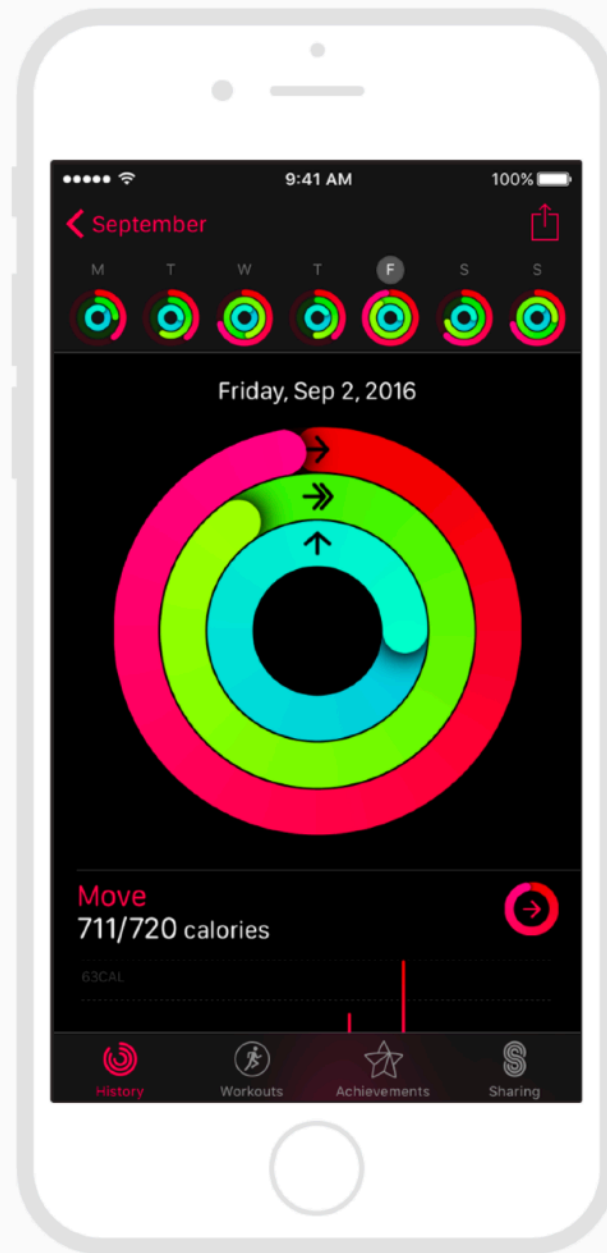
Initializes and returns a color object using the specified opacity and RGB component values.

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

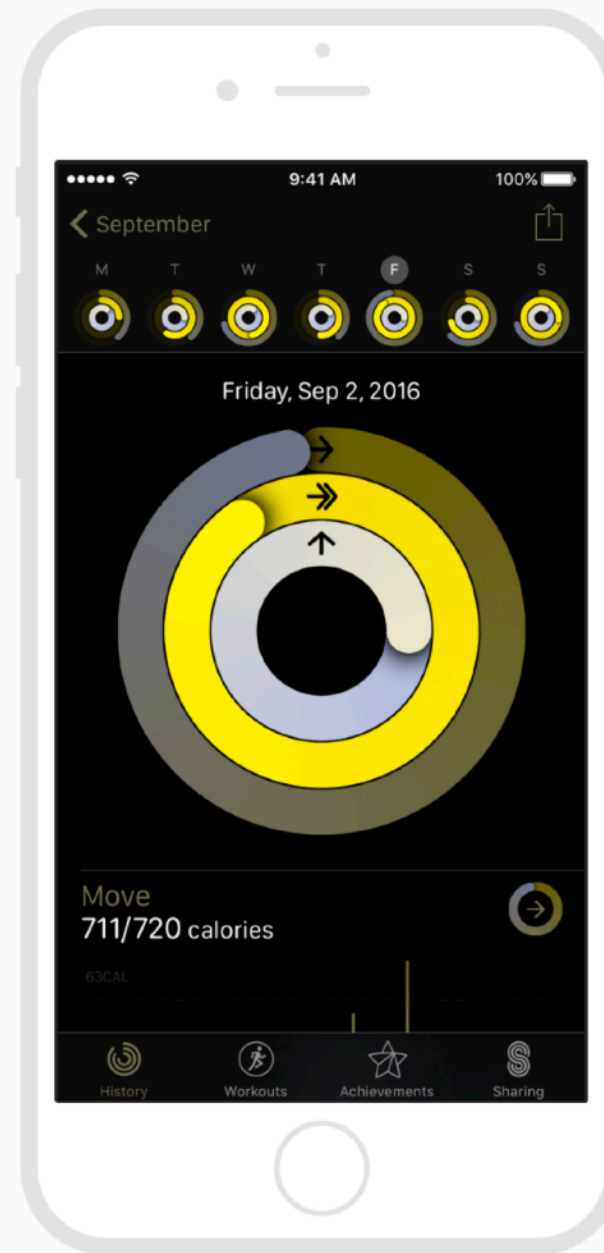
Initializes and returns a color object using the specified opacity and RGB component values in the Display P3 color space.

iOS HIG : Color

App Colors
(standard)



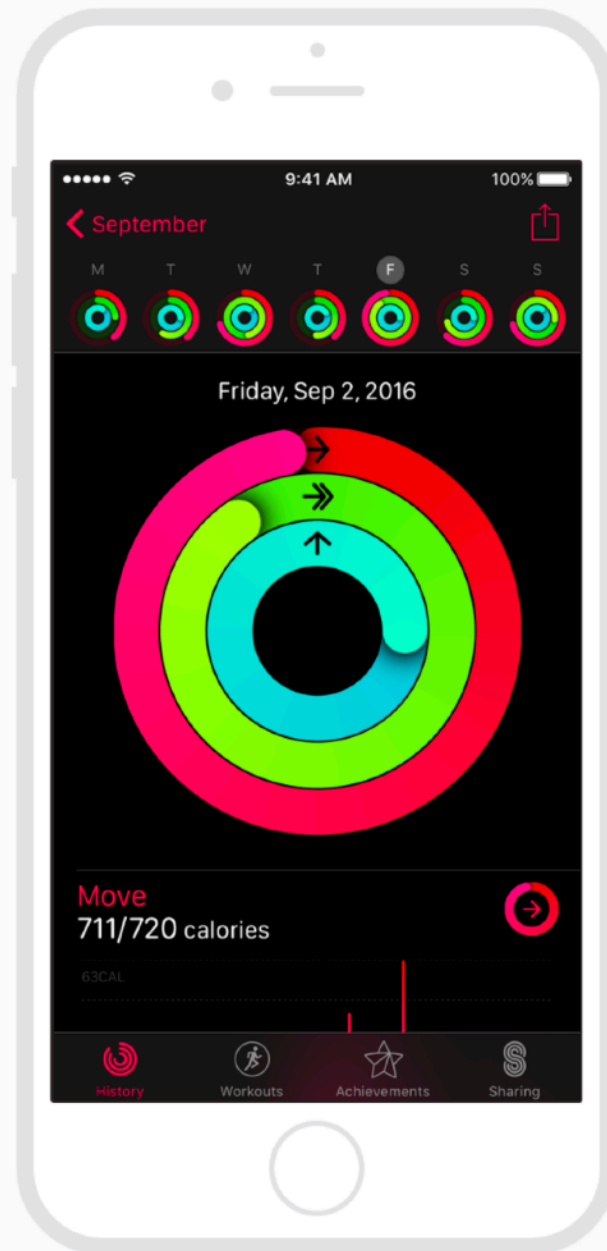
App Colors
(without red-
green)



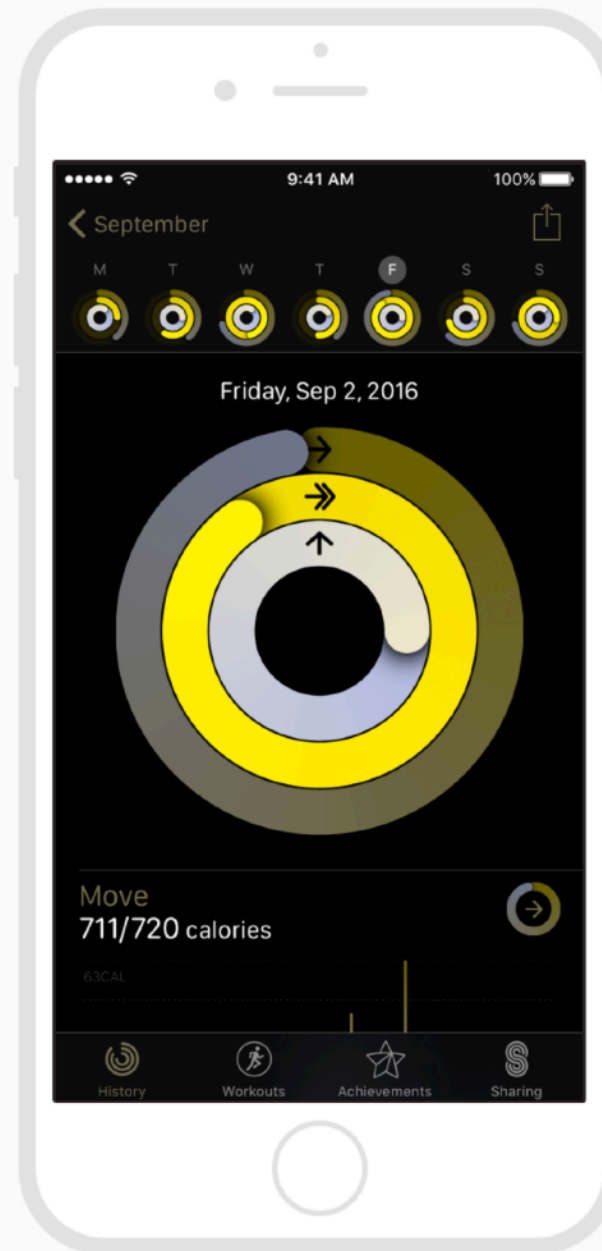
Keep in mind what your app will look like for users with various types of color vision impairment

iOS HIG : Color

App Colors
(standard)



App Colors
(without red-
green)

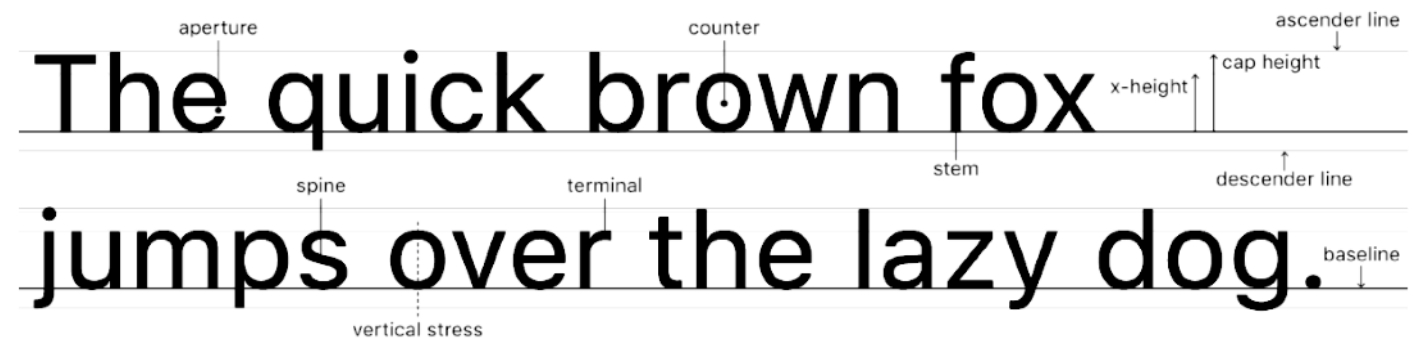


Photoshop has accessibility color filters to help you do this
<http://www.adobe.com/accessibility/products/photoshop.html>

iOS HIG : Fonts and Typography

SF

San Francisco



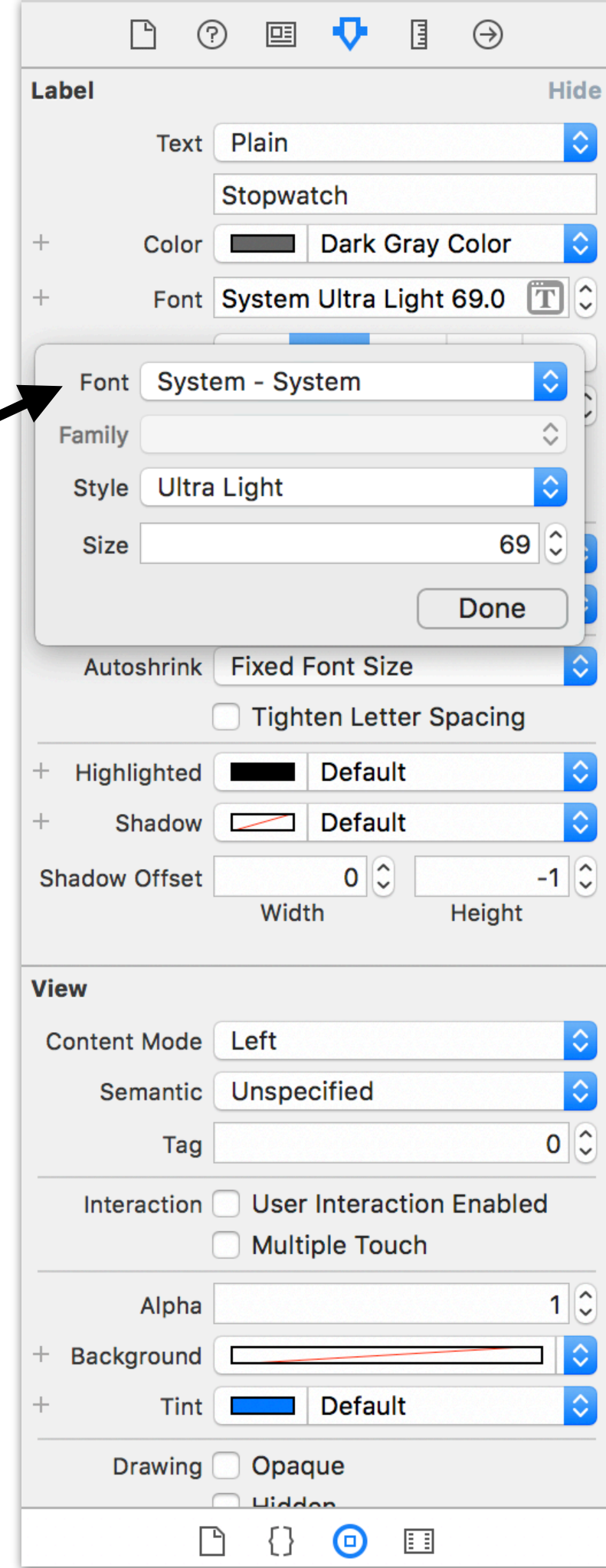
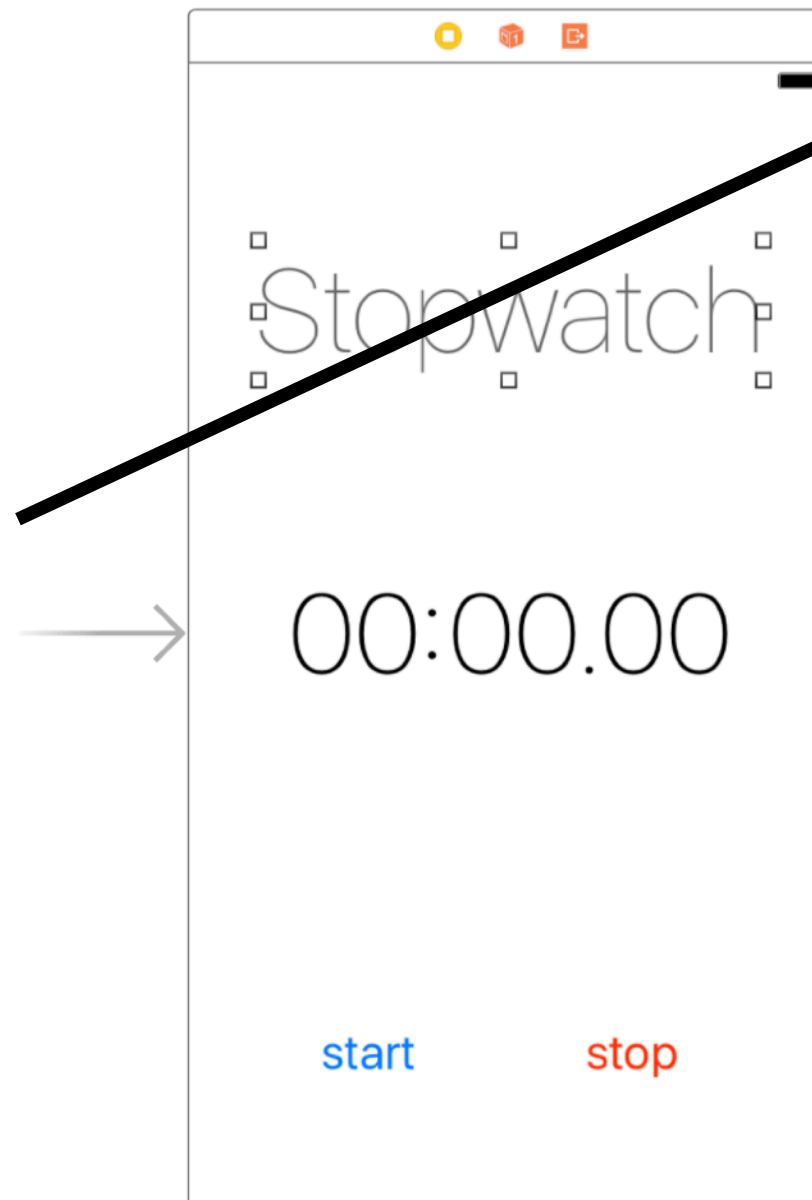
San Francisco

The System Font for iOS

Created by a team at Apple in 2014

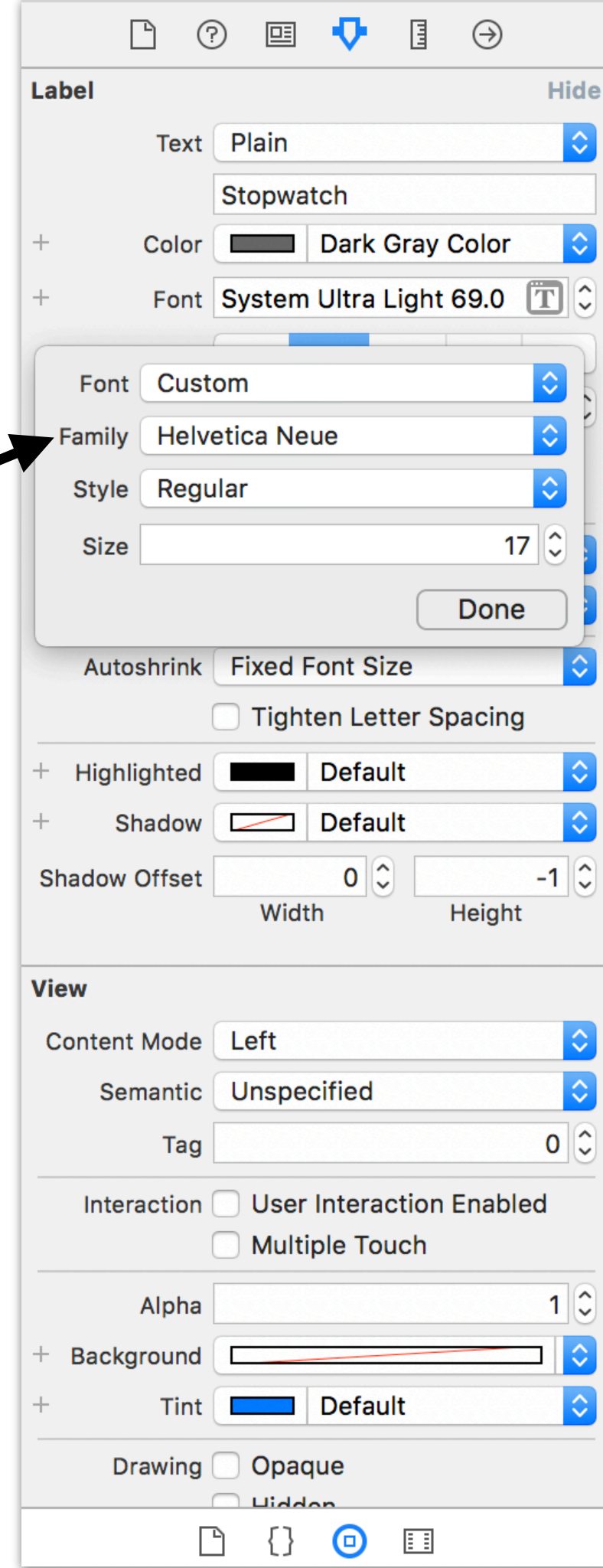
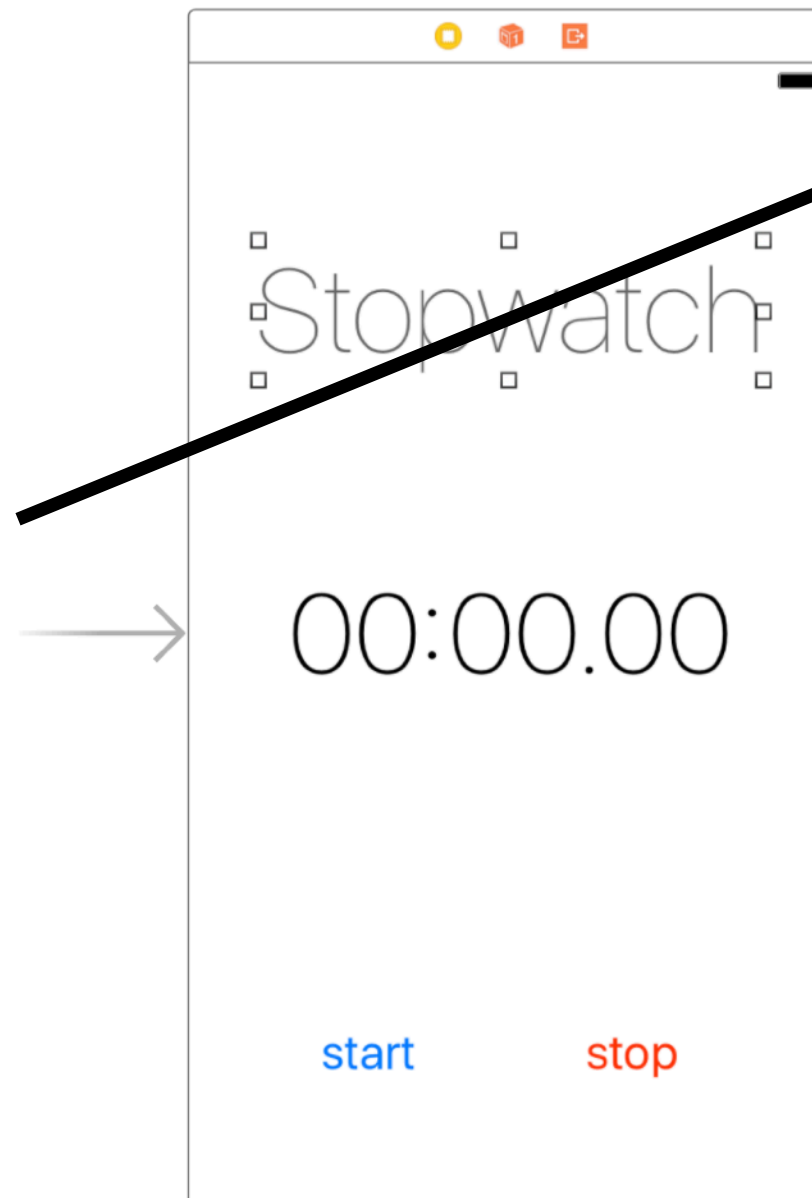
iOS HIG : Fonts

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



iOS HIG : Fonts

Set Font to
"Custom" to
change to a
different Font
Family



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

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

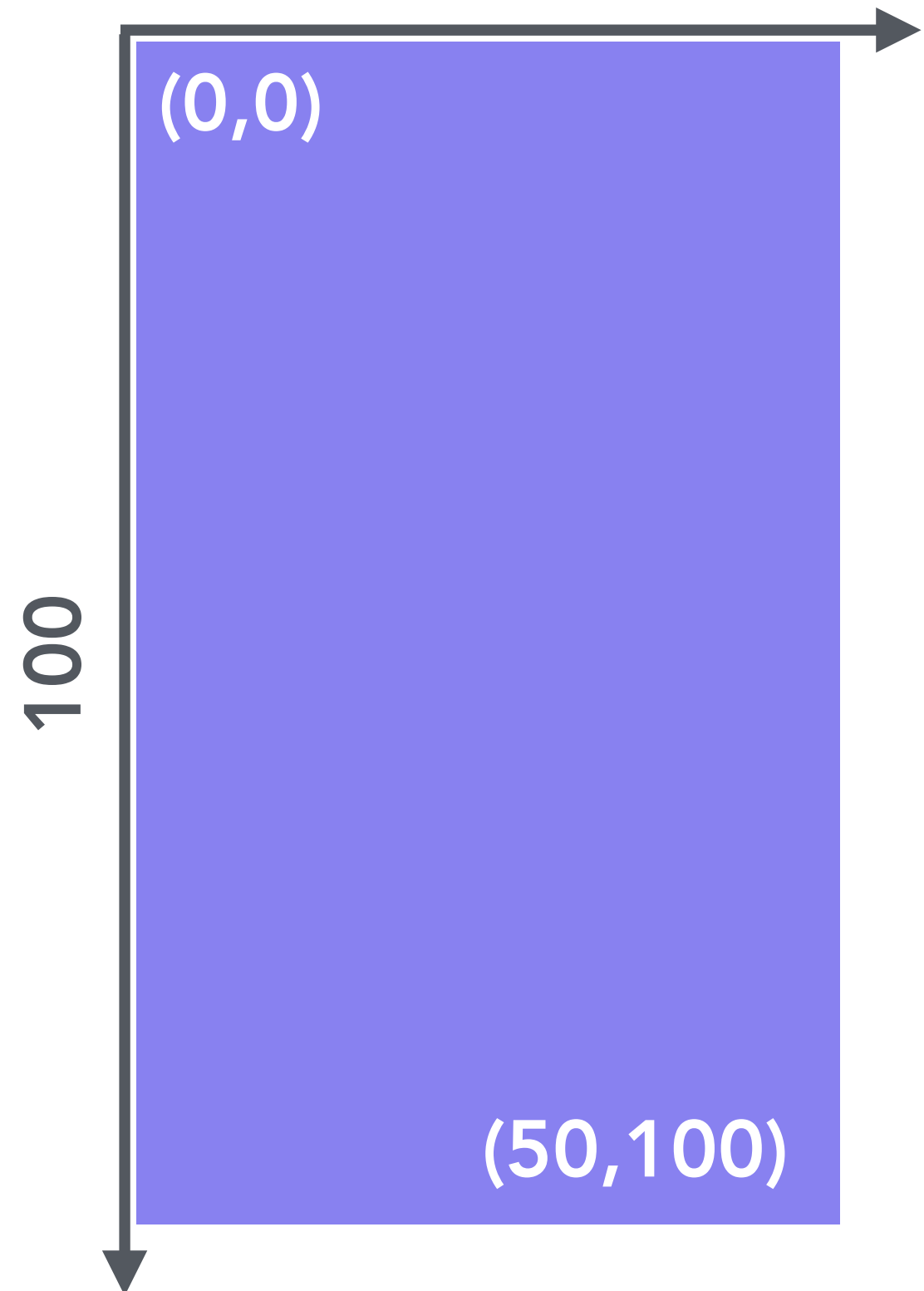
```
CGRect(x: 0, y: 0,  
       width: 100,  
       height: 100)
```

CGPoint - defines a position

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

CGSize - defines a size

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



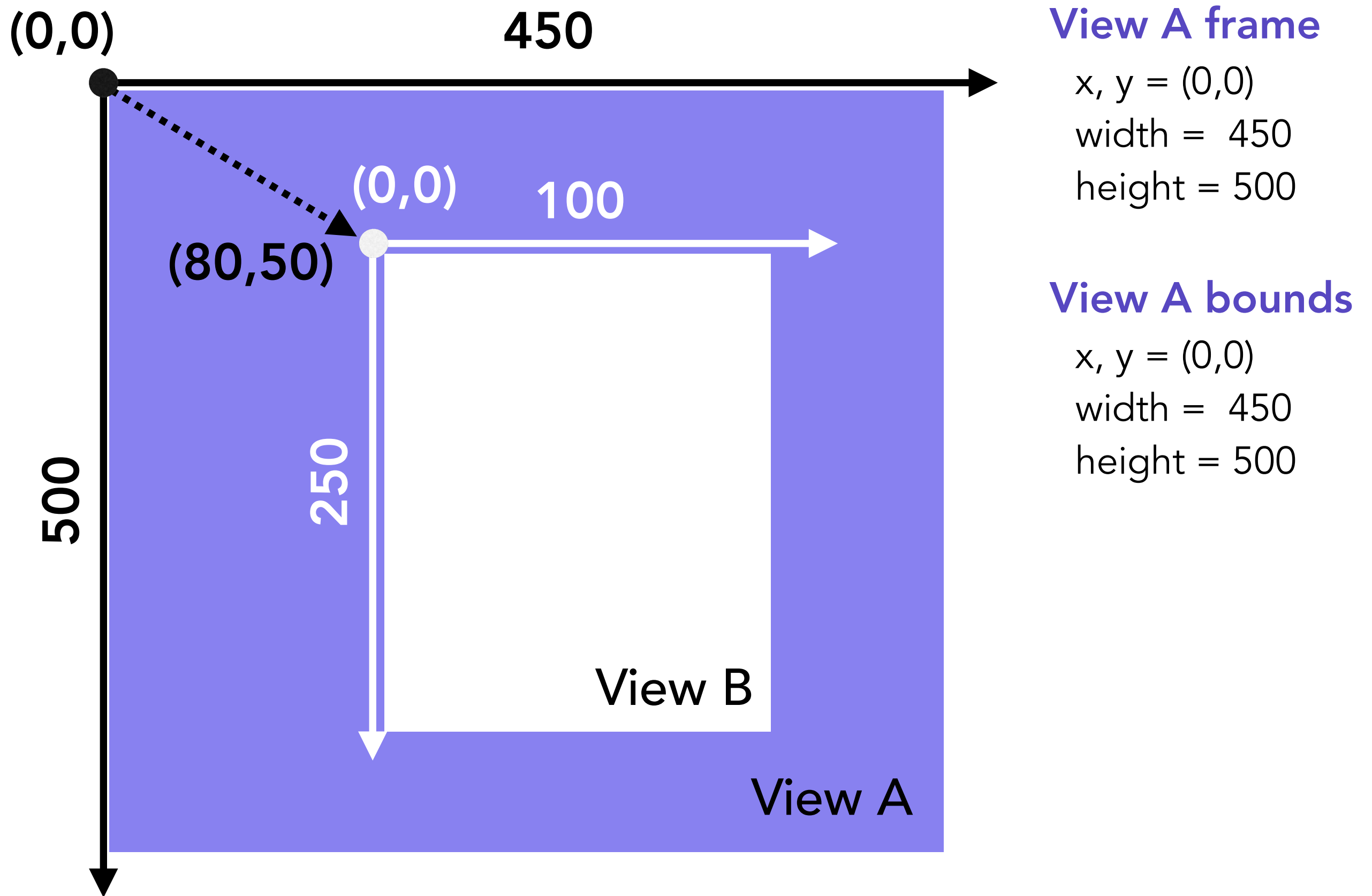
UIView : Geometry

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

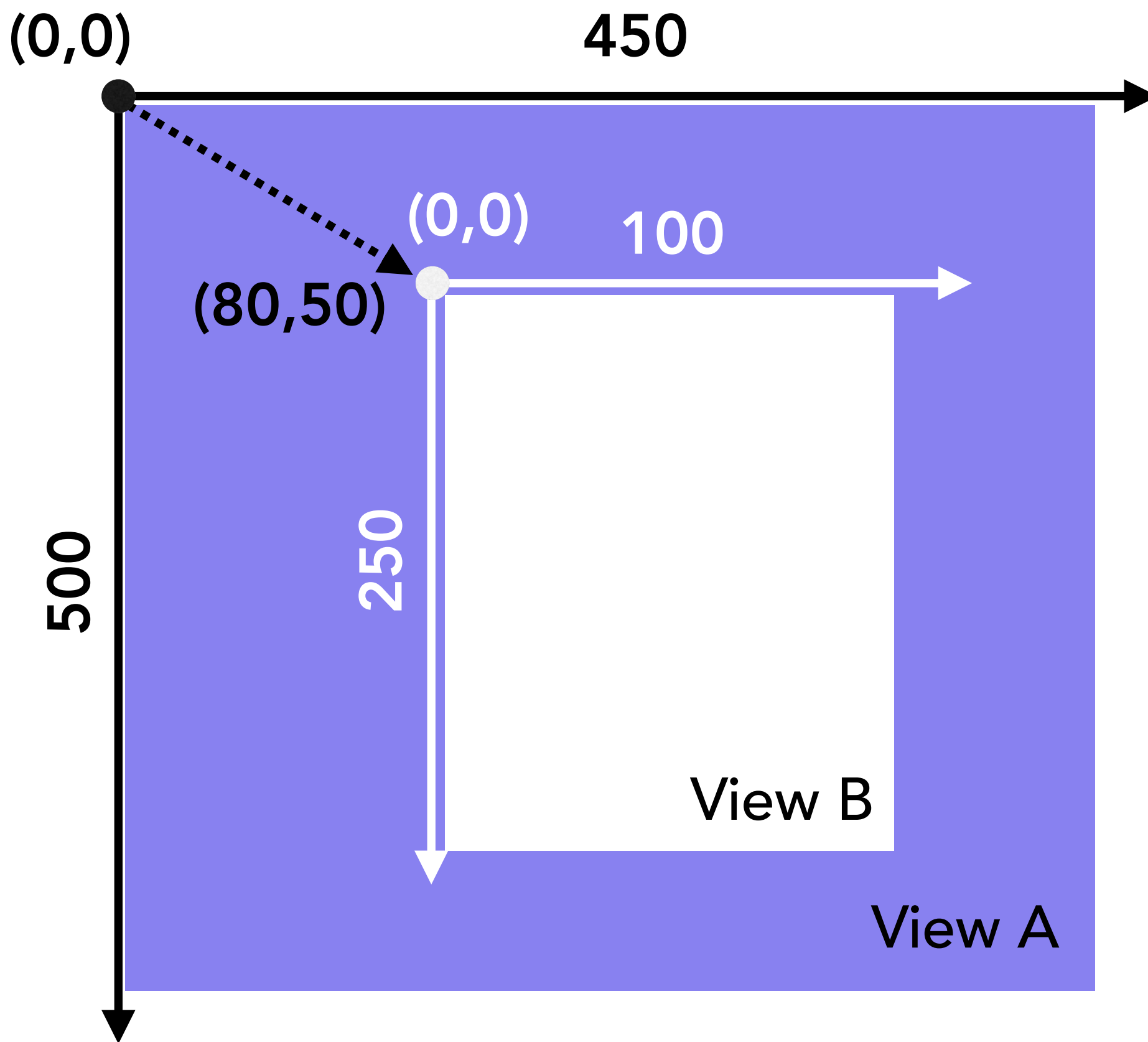
`frame`: `CGRect` - the coordinates and dimensions of the view **in the coordinate system of its superview**

`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)



Frame vs. Bounds



View A frame

$x, y = (0,0)$

width = 450

height = 500

View A bounds

$x, y = (0,0)$

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**

Programmatic Design

Storyboard : Review

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

Pros of Storyboard

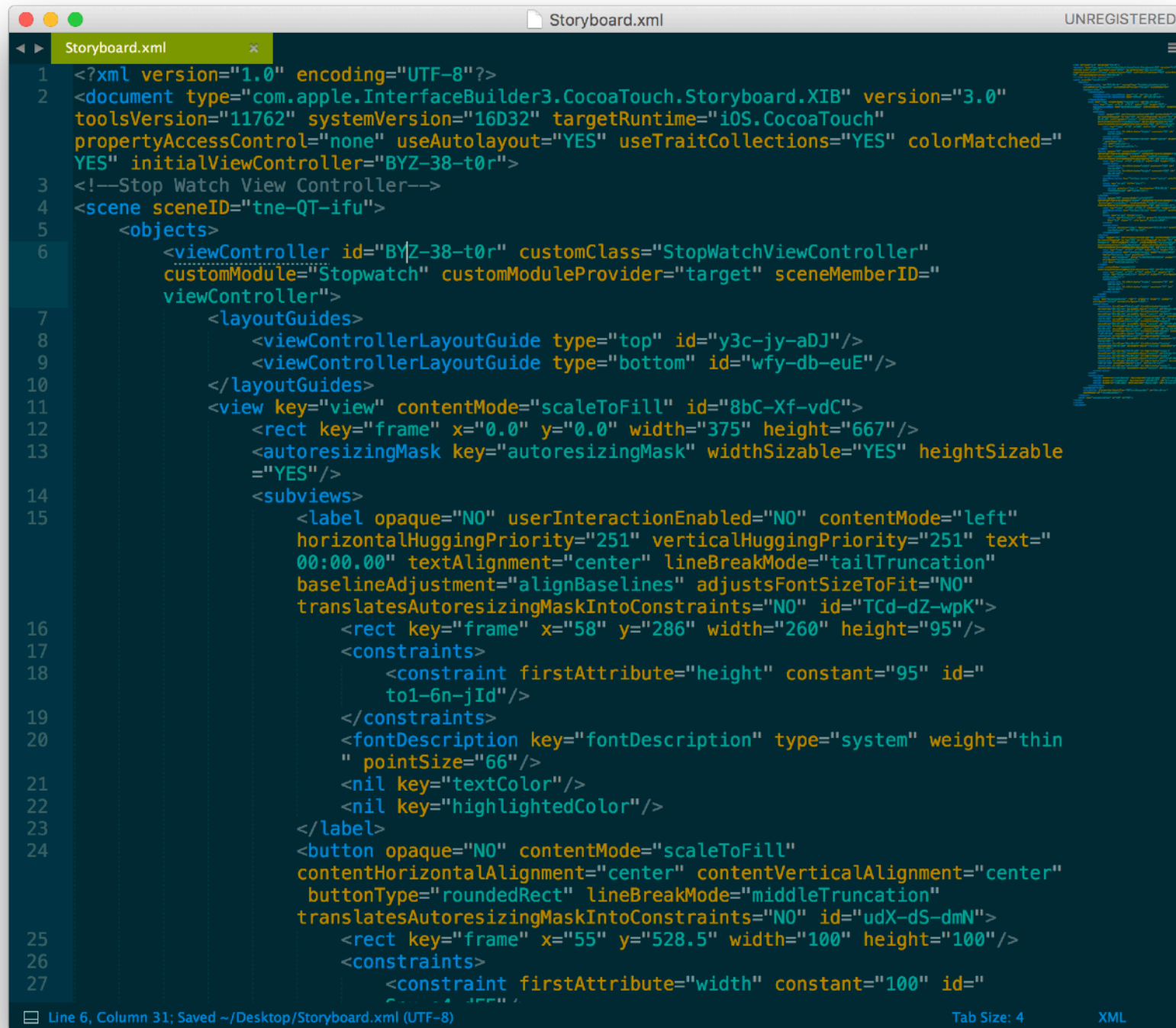
Drag and drop interface makes it really easy to visualize your application immediately

Relatively low learning curve

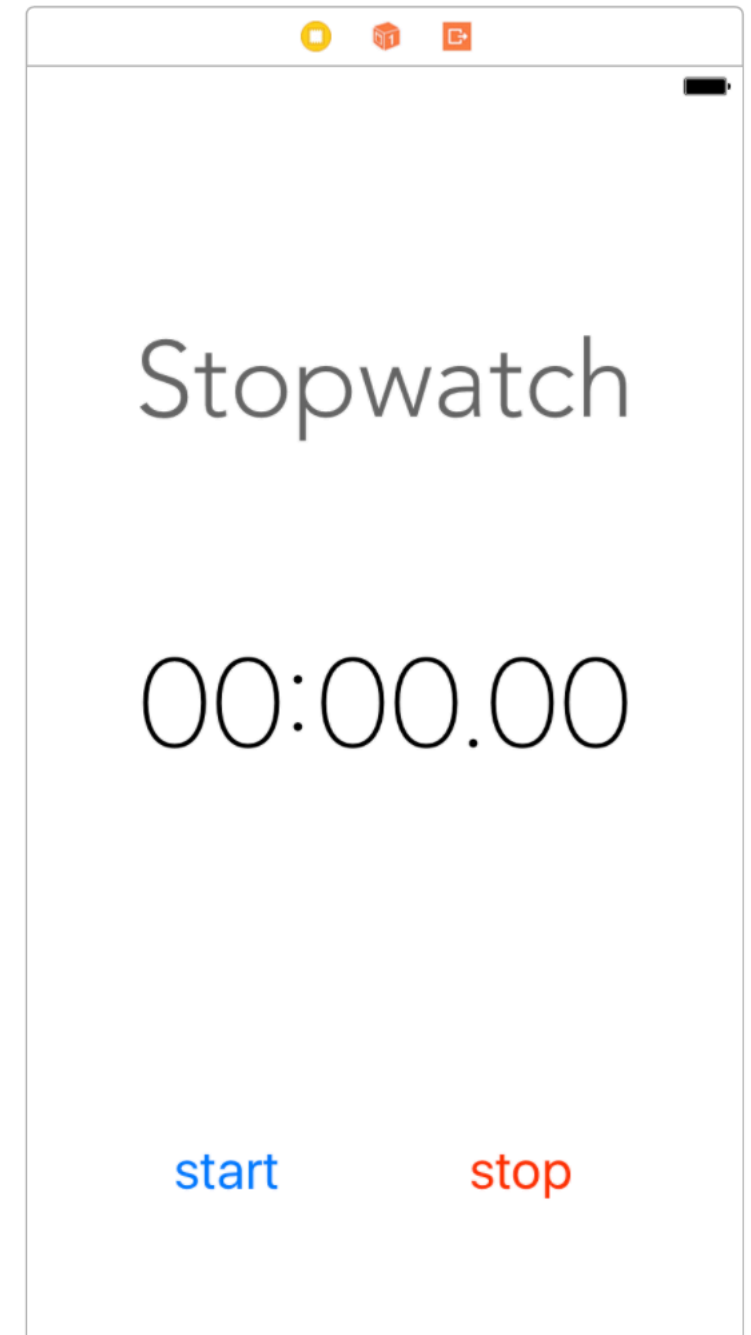
Great for small projects

The future of User Interface programming?

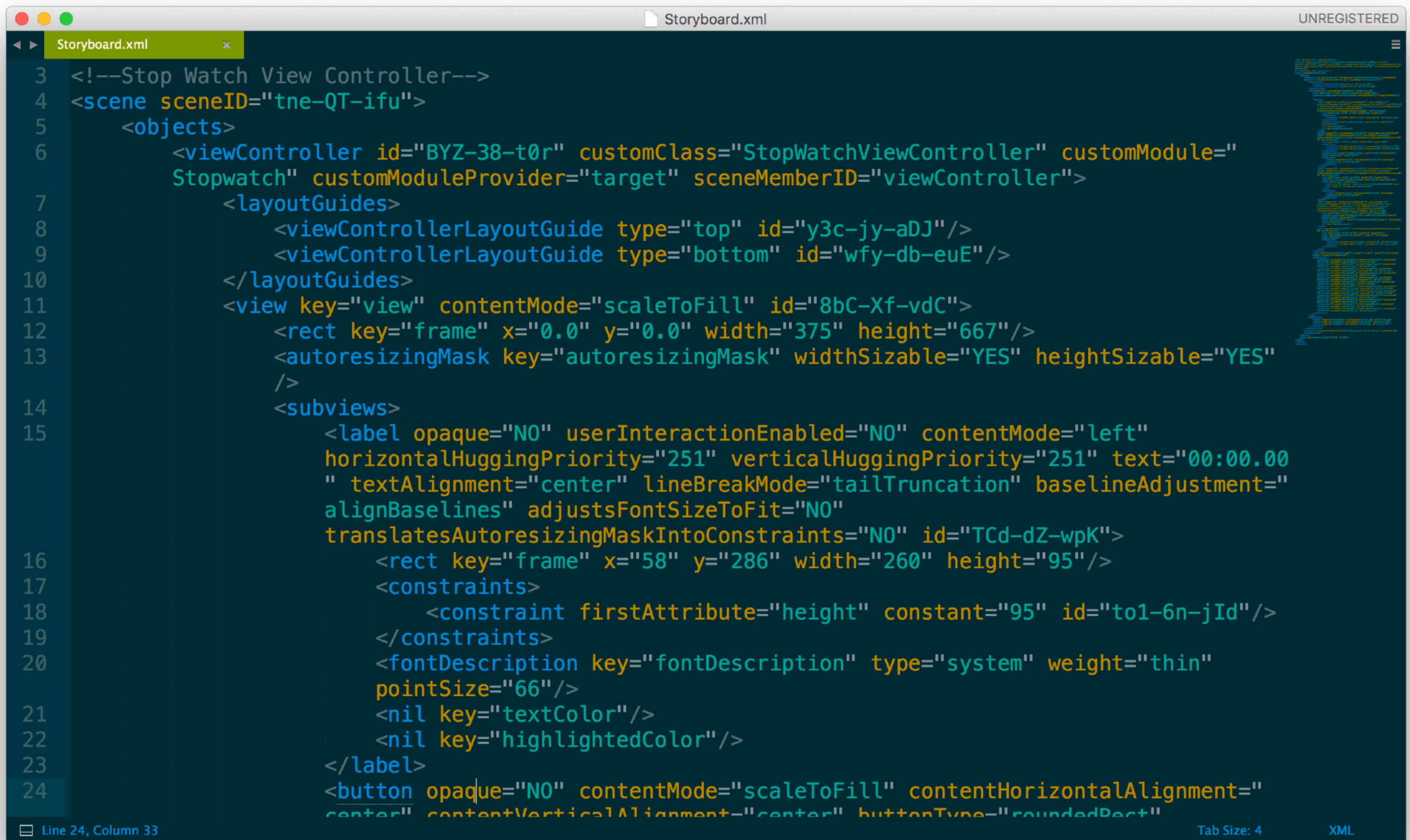
Storyboard : Beneath the hood



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0"
3   toolsVersion="11762" systemVersion="16D32" targetRuntime="iOS.CocoaTouch"
4   propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" colorMatched="
5   YES" initialViewController="BYZ-38-t0r">
6   <!--Stop Watch View Controller-->
7   <scene sceneID="tne-QT-ifu">
8     <objects>
9       <viewController id="BYZ-38-t0r" customClass="StopWatchViewController"
10        customModule="Stopwatch" customModuleProvider="target" sceneMemberID="
11        viewController">
12         <layoutGuides>
13           <viewControllerLayoutGuide type="top" id="y3c-jy-aDJ"/>
14           <viewControllerLayoutGuide type="bottom" id="wfy-db-euE"/>
15         </layoutGuides>
16         <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
17           <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
18           <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable
19            ="YES"/>
20           <subviews>
21             <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
22              horizontalHuggingPriority="251" verticalHuggingPriority="251" text="
23              00:00.00" textAlignment="center" lineBreakMode="tailTruncation"
24              baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
25              translatesAutoresizingMaskIntoConstraints="NO" id="TCd-dZ-wpK">
26               <rect key="frame" x="58" y="286" width="260" height="95"/>
27               <constraints>
28                 <constraint firstAttribute="height" constant="95" id="
29                 tol-6n-jId"/>
30               </constraints>
31               <fontDescription key="fontDescription" type="system" weight="thin"
32                pointSize="66"/>
33               <nil key="textColor"/>
34               <nil key="highlightedColor"/>
35             </label>
36             <button opaque="NO" contentMode="scaleToFill"
37              contentHorizontalAlignment="center" contentVerticalAlignment="center"
38              buttonType="roundedRect" lineBreakMode="middleTruncation"
39              translatesAutoresizingMaskIntoConstraints="NO" id="udX-dS-dmN">
40               <rect key="frame" x="55" y="528.5" width="100" height="100"/>
41               <constraints>
42                 <constraint firstAttribute="width" constant="100" id="
43                 ..."/>
```



Main.storyboard files are just XML files

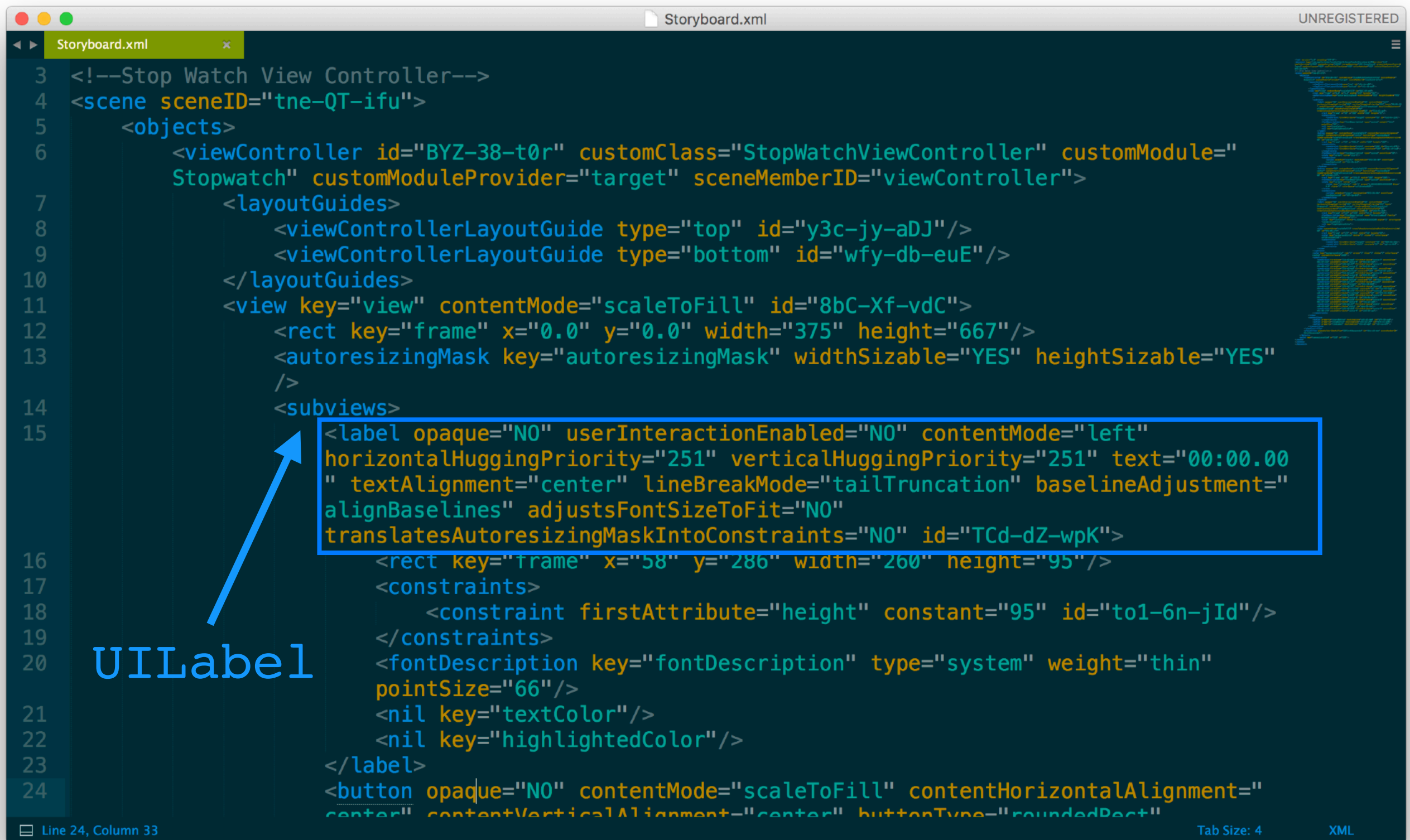
A screenshot of a text editor window titled 'Storyboard.xml' with a dark theme. The code is XML for a storyboard, showing a scene with a view controller, layout guides, a view, and subviews including a label and a button. The status bar at the bottom shows 'Line 24, Column 33', 'Tab Size: 4', and 'XML'.

```
3 <!--Stop Watch View Controller-->
4 <scene sceneID="tne-QT-ifu">
5   <objects>
6     <viewController id="BYZ-38-t0r" customClass="StopWatchViewController" customModule="
7       Stopwatch" customModuleProvider="target" sceneMemberID="viewController">
8       <layoutGuides>
9         <viewControllerLayoutGuide type="top" id="y3c-jy-aDJ"/>
10        <viewControllerLayoutGuide type="bottom" id="wfy-db-euE"/>
11      </layoutGuides>
12      <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
13        <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
14        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"
15        />
16        <subviews>
17          <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
18            horizontalHuggingPriority="251" verticalHuggingPriority="251" text="00:00.00"
19            textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="
20            alignBaselines" adjustsFontSizeToFit="NO"
21            translatesAutoresizingMaskIntoConstraints="NO" id="TCd-dZ-wpK">
22            <rect key="frame" x="58" y="286" width="260" height="95"/>
23            <constraints>
24              <constraint firstAttribute="height" constant="95" id="to1-6n-jId"/>
25            </constraints>
26            <fontDescription key="fontDescription" type="system" weight="thin"
27              pointSize="66"/>
28            <nil key="textColor"/>
29            <nil key="highlightedColor"/>
30          </label>
31          <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="
32            center" contentVerticalAlignment="center" buttonType="roundedRect"
33            />
34        </subviews>
35      </view>
36    </viewController>
37  </objects>
38</scene>
```

Line 24, Column 33

Tab Size: 4 XML

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



```
3 <!--Stop Watch View Controller-->
4 <scene sceneID="tne-QT-ifu">
5   <objects>
6     <viewController id="BYZ-38-t0r" customClass="StopWatchViewController" customModule="
7       Stopwatch" customModuleProvider="target" sceneMemberID="viewController">
8       <layoutGuides>
9         <viewControllerLayoutGuide type="top" id="y3c-jy-aDJ"/>
10        <viewControllerLayoutGuide type="bottom" id="wfy-db-euE"/>
11      </layoutGuides>
12      <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
13        <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
14        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"
15        />
16        <subviews>
17          <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
18            horizontalHuggingPriority="251" verticalHuggingPriority="251" text="00:00.00"
19            textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="
20            alignBaselines" adjustsFontSizeToFit="NO"
21            translatesAutoresizingMaskIntoConstraints="NO" id="TCd-dZ-wpK">
22            <rect key="frame" x="58" y="286" width="260" height="95"/>
23            <constraints>
24              <constraint firstAttribute="height" constant="95" id="to1-6n-jId"/>
25            </constraints>
26            <fontDescription key="fontDescription" type="system" weight="thin"
27              pointSize="66"/>
28            <nil key="textColor"/>
29            <nil key="highlightedColor"/>
30          </label>
31          <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="
32            center" contentVerticalAlignment="center" buttonType="roundedRect"
33            />
34        </subviews>
35      </view>
36    </viewController>
37  </objects>
38</scene>
```

UILabel

Line 24, Column 33

Tab Size: 4 XML

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

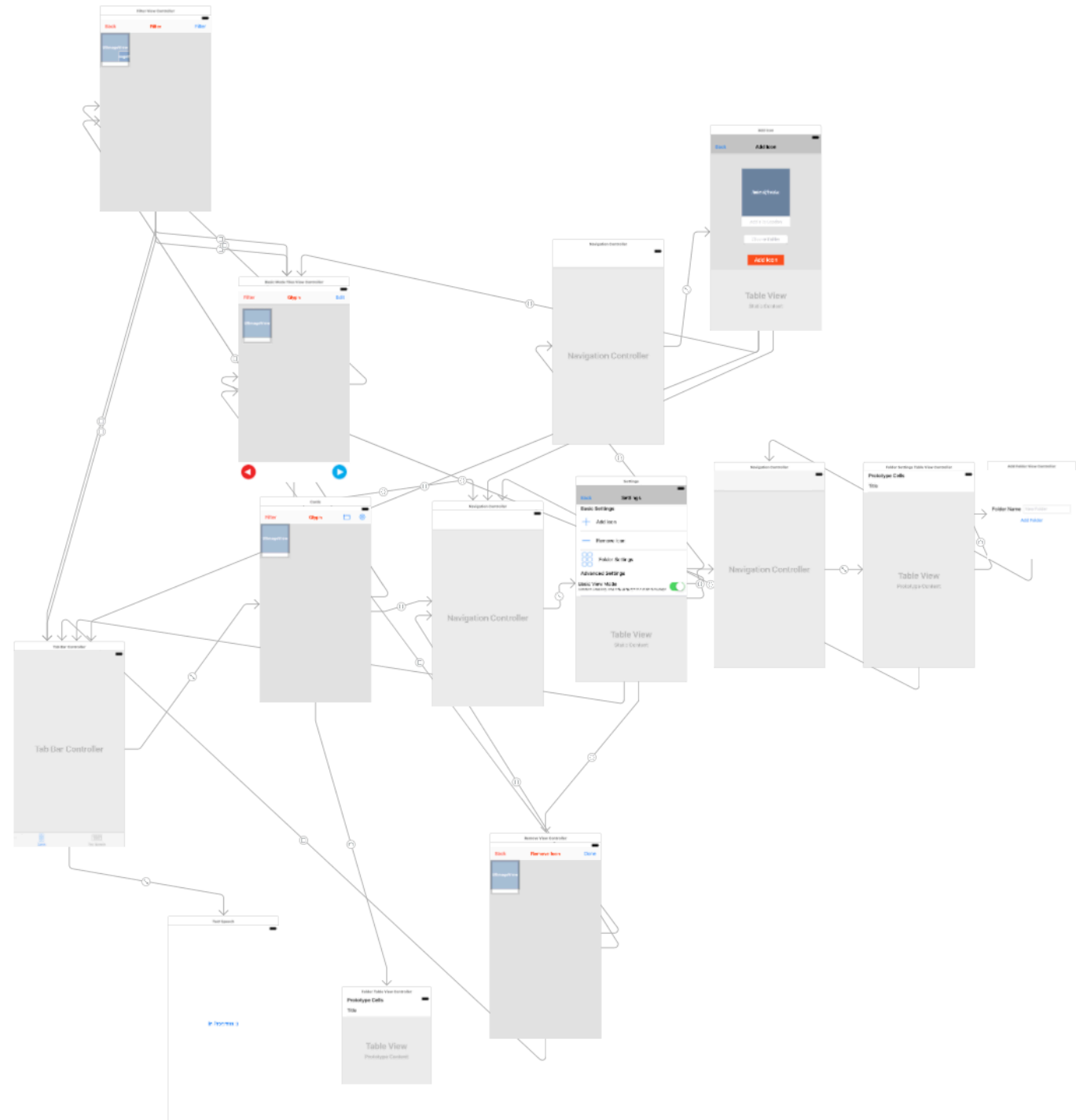
Storyboard : Problems

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 by user [neilinglis](#) with ID `gist:e238d5f22f85fa259ade`, created 3 years ago. It displays a merge conflict in a file named `gistfile1.txt`. The conflict is in a storyboard XML file, with the message "Storyboard Merge Conflict. Is there any sensible course of action for this?". The code shows a sequence of segue references, with a conflict marked by "=====" on line 8. The lines are numbered 1 through 15.

```
1 <<<<<<< HEAD
2     <segue reference="kXa-Mw-CAj"/>
3     <segue reference="TDo-1S-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-1S-nUS"/>
13    <segue reference="hJU-8t-Kde"/>
14    <segue reference="y7Z-qu-r0P"/>
15 >>>>>>> e9a57872e96f17a8d2d785e4de0132e75229a262
```

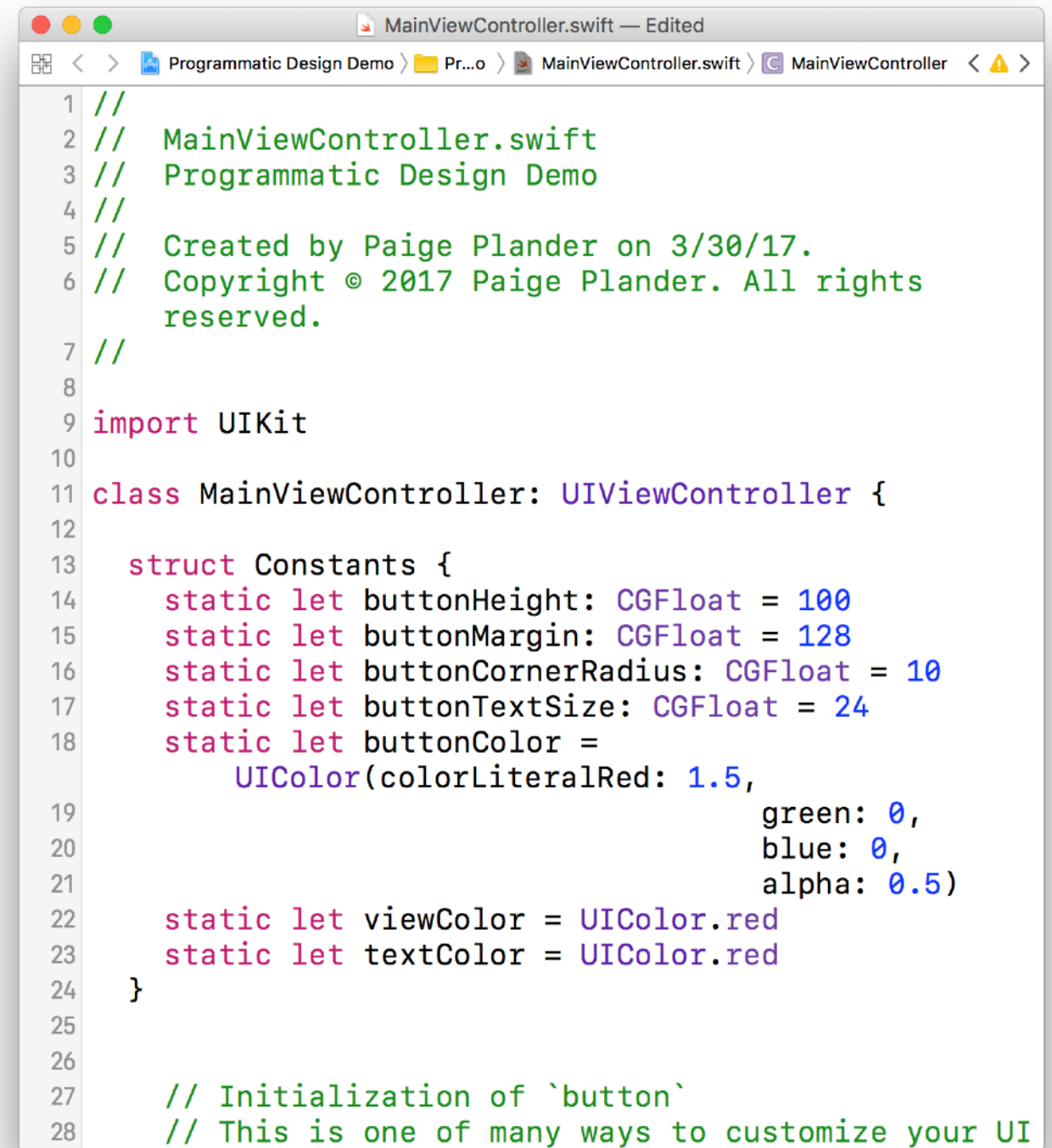
Storyboard : Problems

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



```
1 //
2 // MainViewController.swift
3 // Programmatic Design Demo
4 //
5 // Created by Paige Plander on 3/30/17.
6 // Copyright © 2017 Paige Plander. All rights reserved.
7 //
8
9 import UIKit
10
11 class MainViewController: UIViewController {
12
13     struct Constants {
14         static let buttonHeight: CGFloat = 100
15         static let buttonMargin: CGFloat = 128
16         static let buttonCornerRadius: CGFloat = 10
17         static let buttonTextSize: CGFloat = 24
18         static let buttonColor =
19             UIColor(colorLiteralRed: 1.5,
20                     green: 0,
21                     blue: 0,
22                     alpha: 0.5)
23         static let viewColor = UIColor.red
24         static let textColor = UIColor.red
25     }
26
27     // Initialization of `button`
28     // This is one of many ways to customize your UI
```


Programmatic Design

No Storyboard Needed

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

Pros

Better for version control

Scalable

Industry standard

Less limited

Cons

Steeper learning curve

Slower to get started

Code gets long

(especially when using constraints)

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

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)
```

Summary - iOS UI Implementation

Programmatic Design vs Storyboard

... so which one is better?

Depends on what you are creating

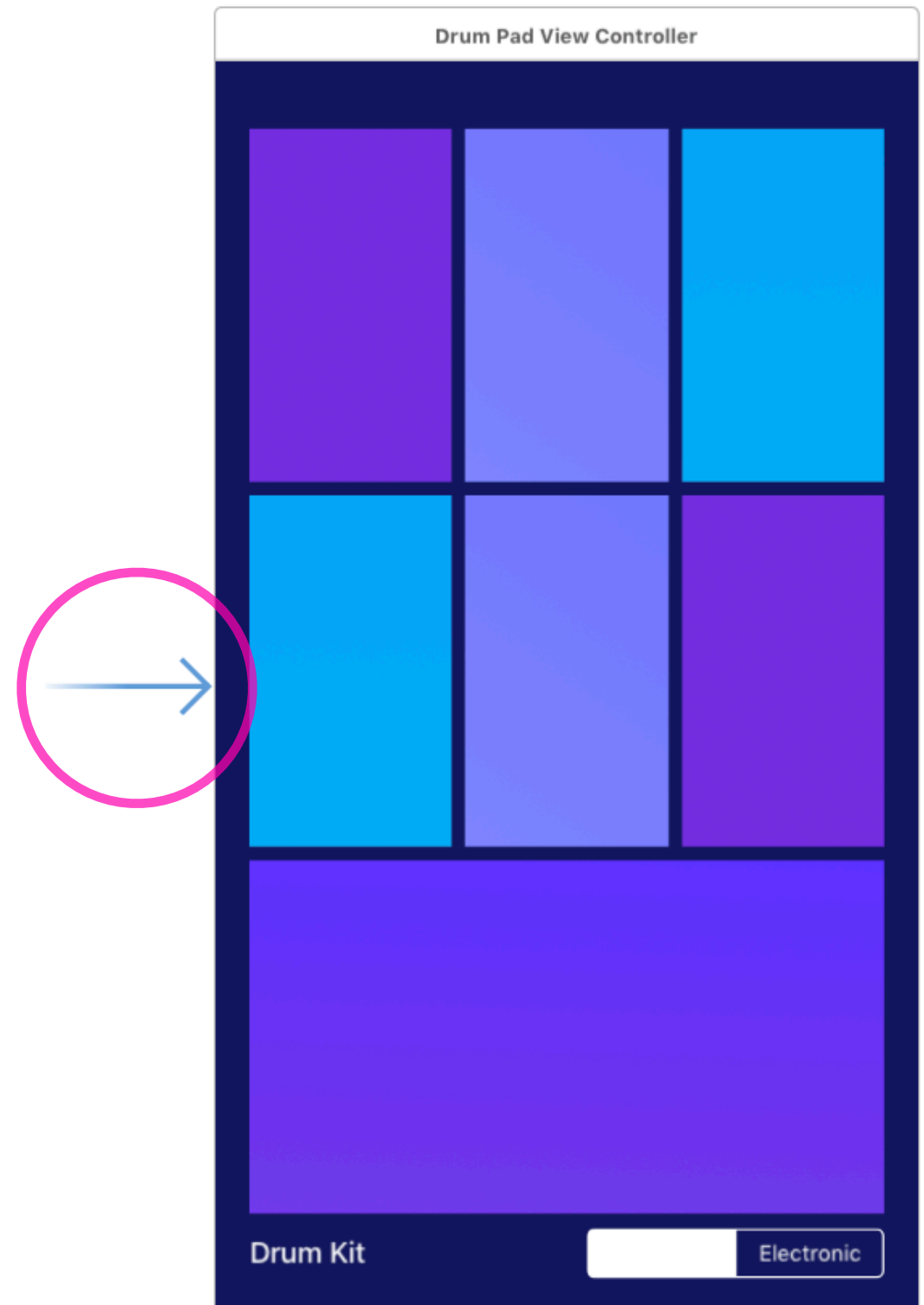
Often times, a combination of both may be the best solution

One idea - control navigation via Storyboard, add UI elements programmatically

Programmatic Design (with no Storyboard)

To get rid of your storyboard, delete both the **Main.storyboard** file and its 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
```

Found in AppDelegate.swift

```
@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
```

```
    }
```

```
    ...
```

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

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** (`NSLayoutConstraint`)

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)

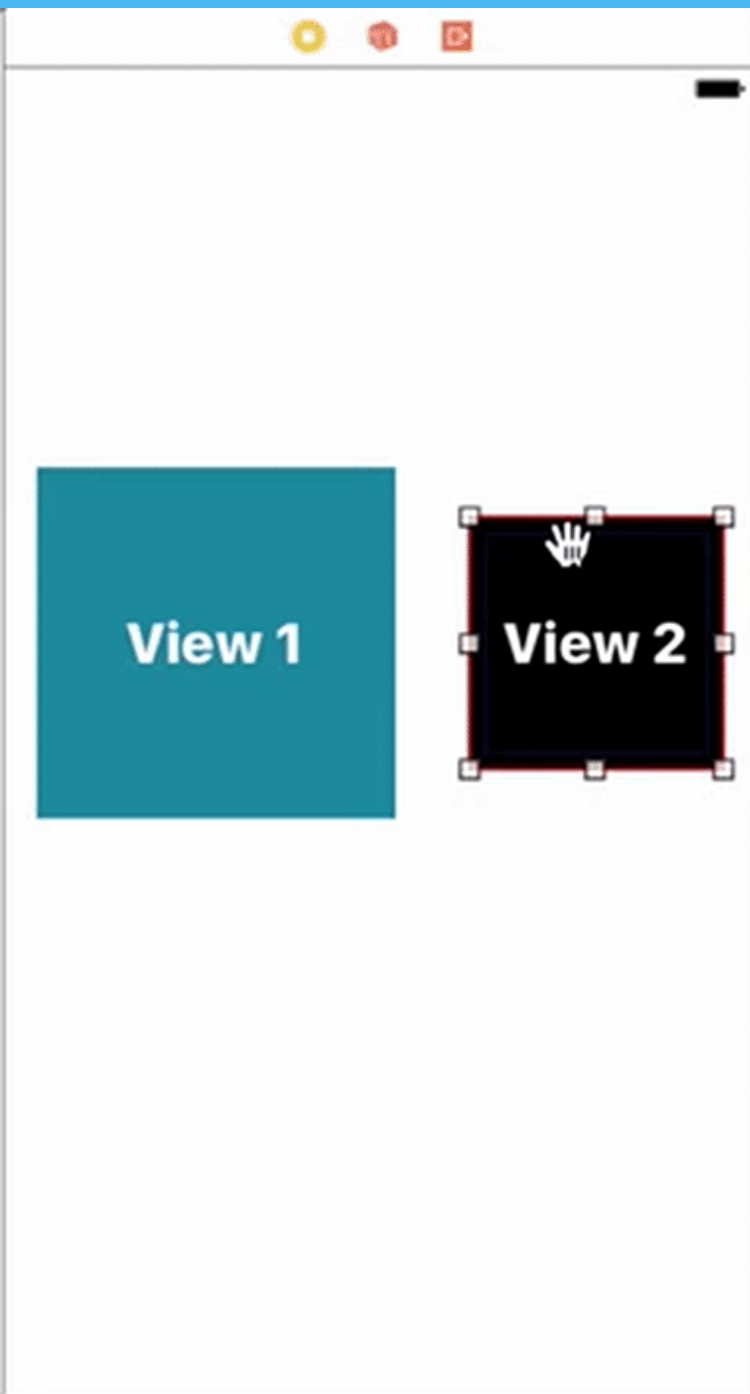
Programmatic AutoLayout

Layout Anchors

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

```
constraint.isActive = true
```

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

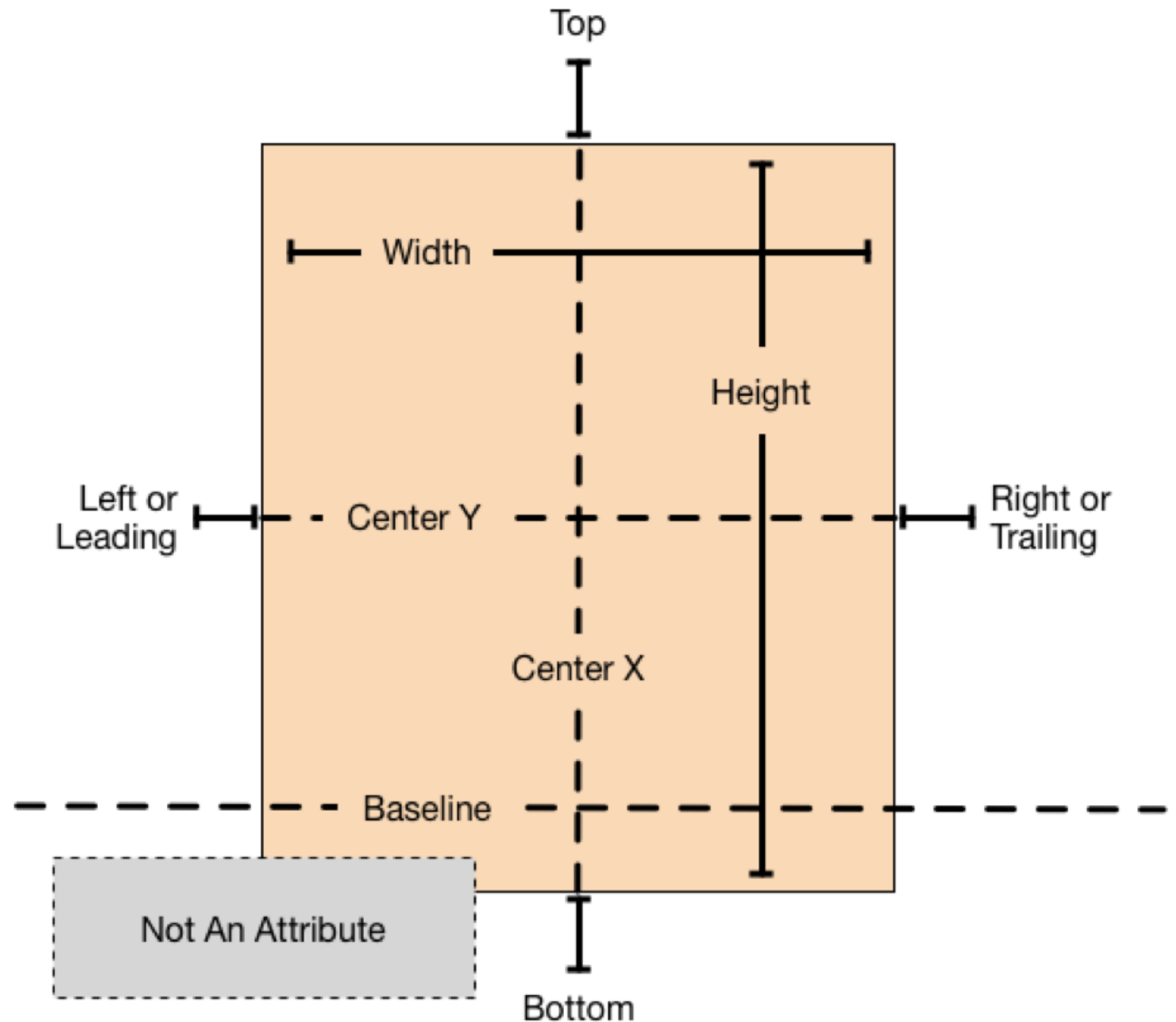


Equivalent
Storyboard Example

Programmatic AutoLayout

Layout Anchor Properties

Use these properties to create relationships between views



Programmatic Design Demo

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

Snapchat Project Part 2

Due **Tuesday** at 11:59pm

Next Lecture : Delegates, Protocols, and
Advanced Swift