



Additional Views and Controllers

Dr. Sarah Abraham

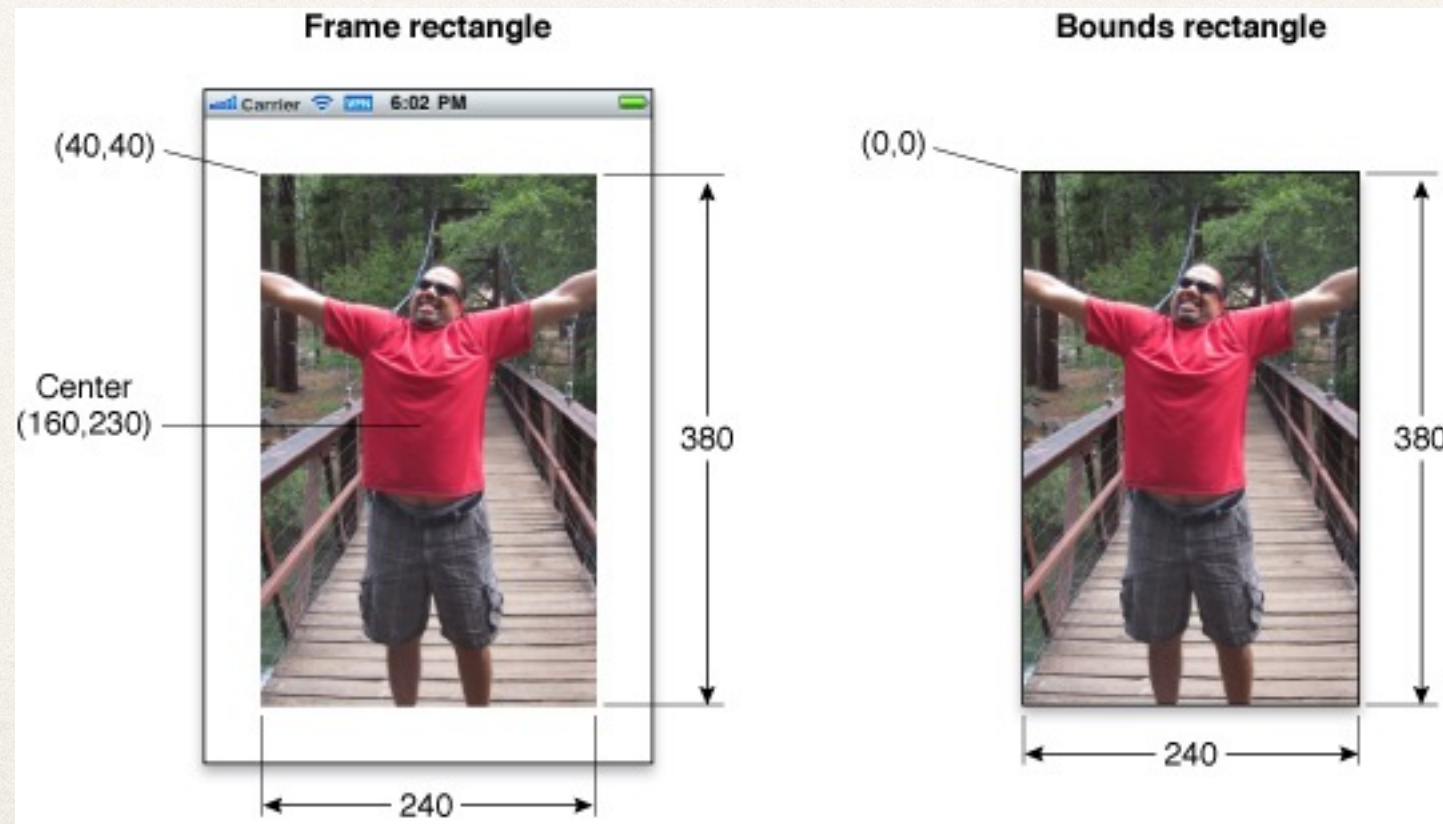
University of Texas at Austin
CS329e
Fall 2019

View Frames and Bounds

- ❖ All views have both a frame and a bounds structure
- ❖ Both defined as CGRects:
 - ❖ origin (x, y)
 - ❖ dimensions (width, height)
- ❖ A view's frame is the rectangle position within the superview's coordinate system
- ❖ A view's bounds is the view rectangle within the view's own coordinate system
- ❖ A view's center is the center point of the view within the superview's coordinate system

Relationship between Frames and Bounds

- ❖ $\text{frame.origin} = \text{center} - (\text{bounds.size} / 2.0)$
- ❖ $\text{center} = \text{frame.origin} + (\text{bounds.size} / 2.0)$
- ❖ $\text{frame.size} = \text{bounds.size}$

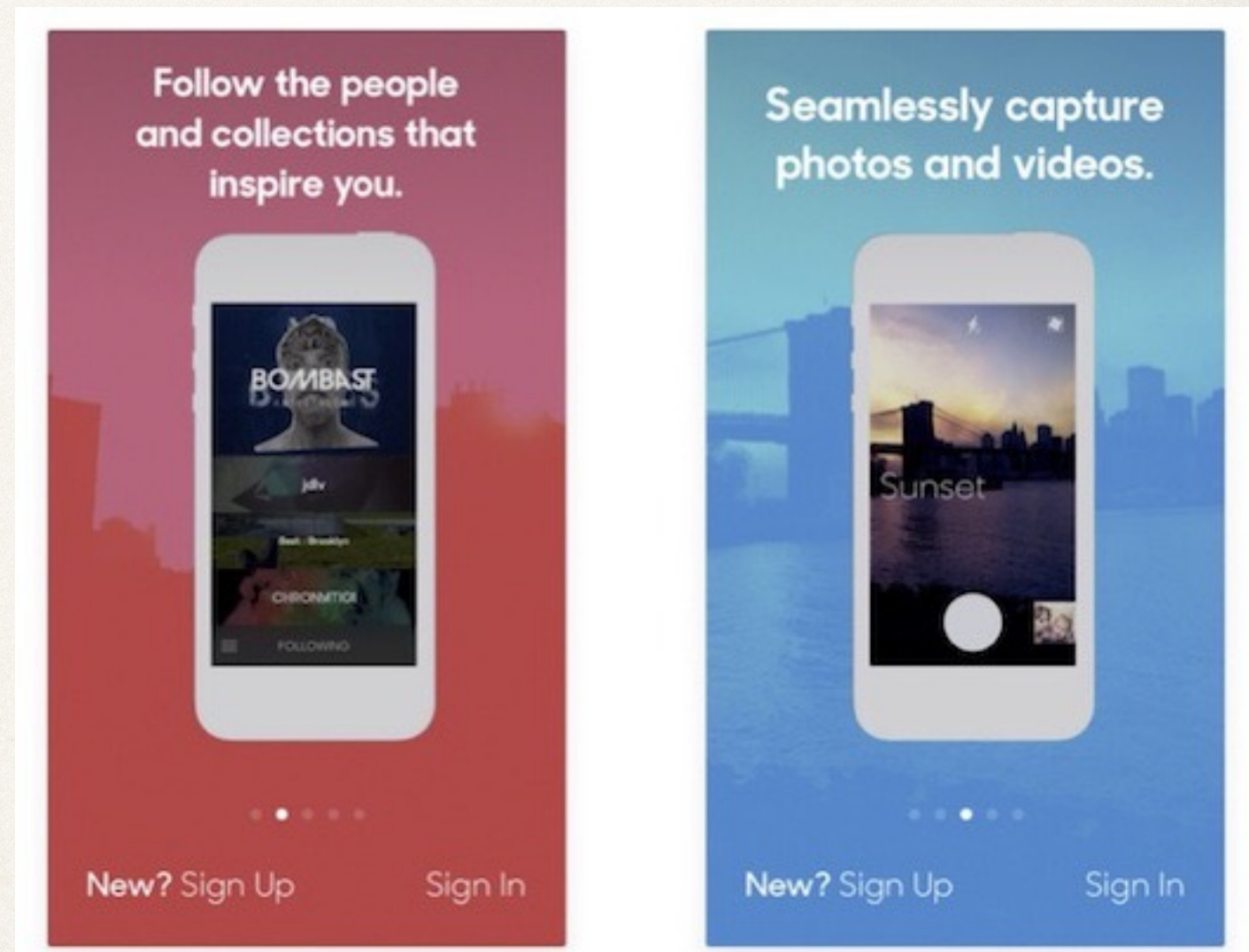


Views and Controllers

- ❖ Lots of different types and functionalities!
- ❖ Similar in concept to other views and controllers
- ❖ But the devil's in the details...
- ❖ Choose the view and controller based on desired functionality

Page Views

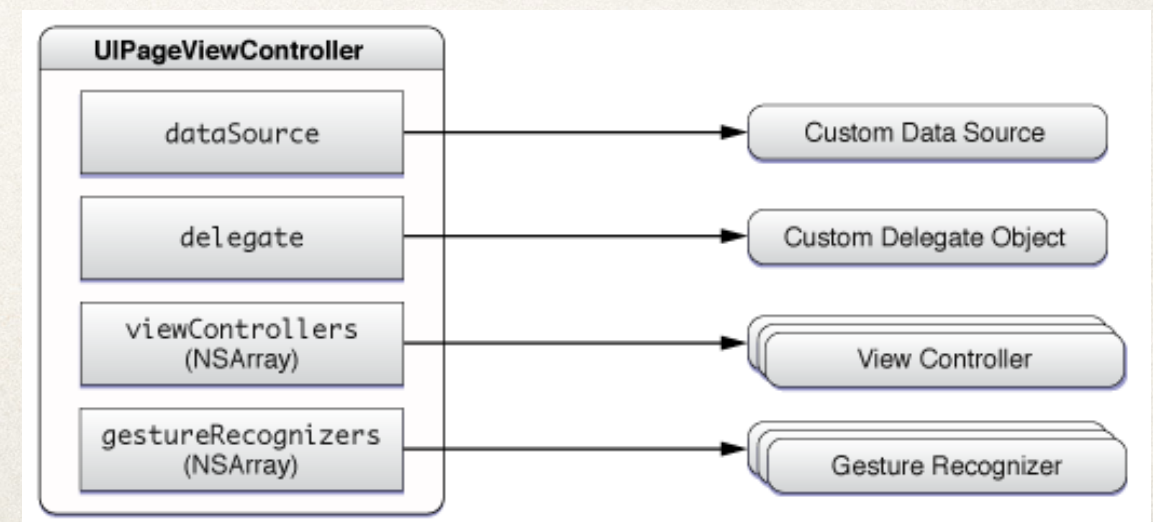
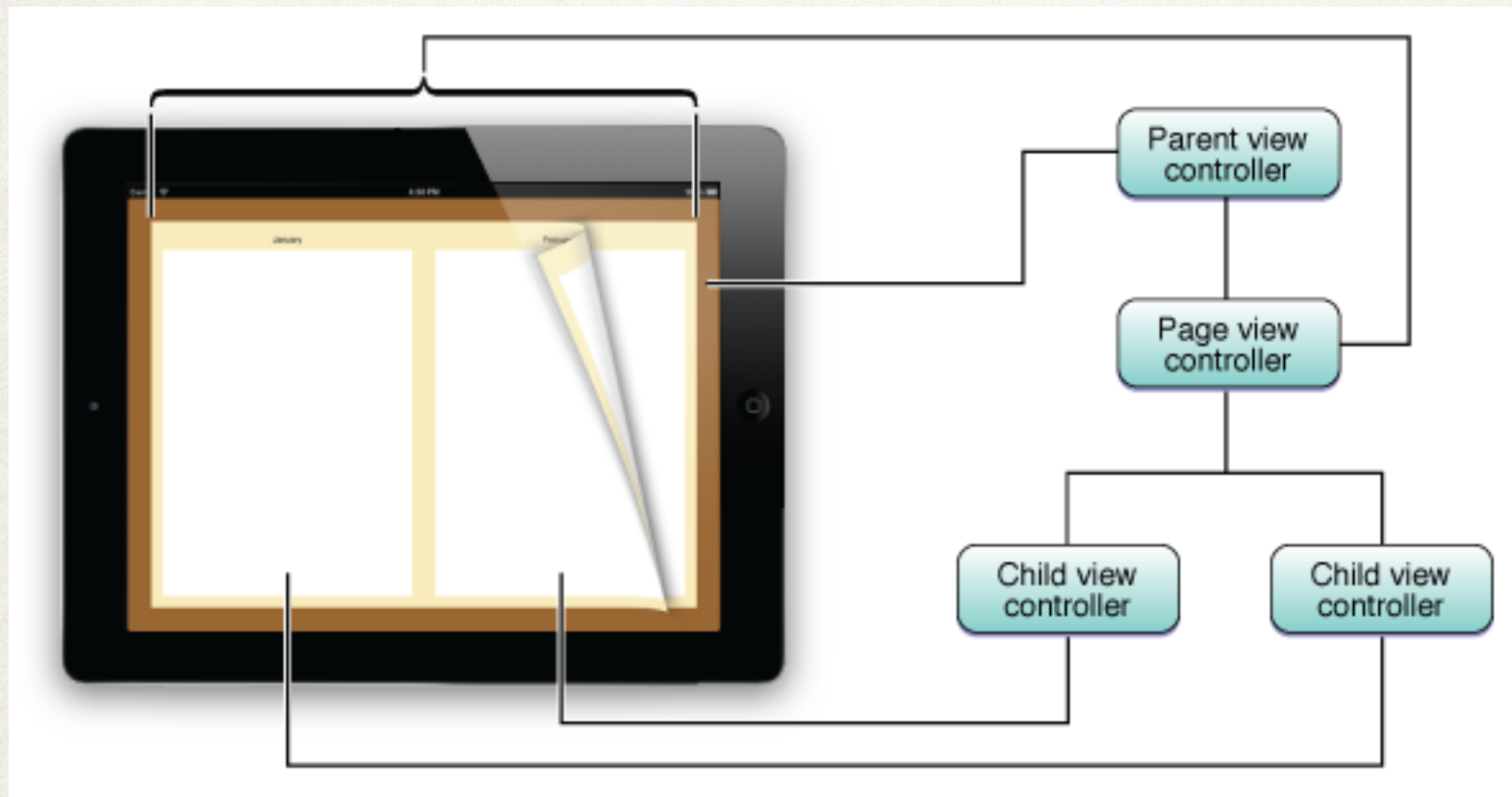
- ❖ Presents content in a page-by-page manner
- ❖ Ideal for linear content
- ❖ Ideal for content with natural page breaks



UIPageViewController

- ❖ Container controller to manage multiple controllers
 - ❖ View hierarchy
 - ❖ Child views managed by their own content view controllers
- ❖ Configurable appearance and definitions:
 - ❖ Orientation of page views (vertical or horizontal)
 - ❖ Transition style (page curl or scrolling)
 - ❖ Visual layout (location of spine or page spacing)

Page View Elements



Tab Views

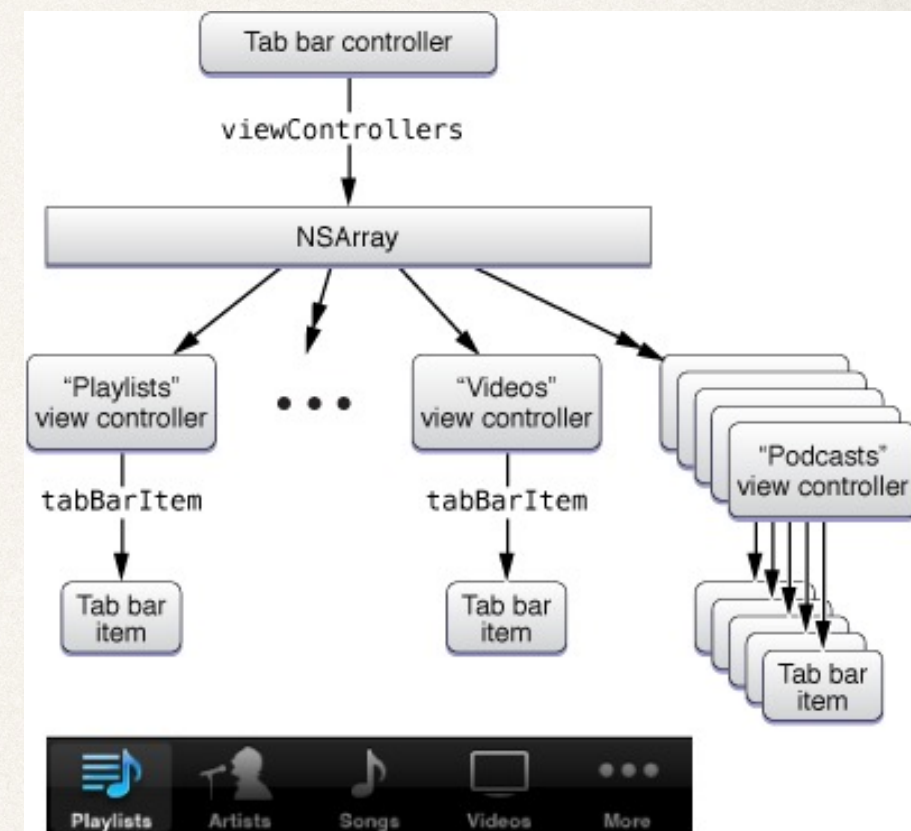
- ❖ Presents tab bar to switch between content
- ❖ Organizes content across multiple modes
- ❖ Provides different perspectives for same content



Tab Bar Interface

- ❖ Container for all necessary tab bar objects:
 - ❖ `UITabBarController`
 - ❖ Content view controllers for each tab
 - ❖ Optional delegate
- ❖ Six or more custom views automatically generates “More” button for display

Tab Bar Hierarchy



Segmented Controls



- ❖ Same idea as the tab bar but used within a single view controller
- ❖ Horizontal bar with segments functioning as buttons
- ❖ Register control event and perform action when value changes:
 - ❖ `segmentedControl.addTarget(self, action: "action:", forControlEvents: .ValueChanged);`

Segmented Controls: Target

- ❖ addTarget associates segment selection with an action

```
segmentedControl.addTarget(self, action: "action:",  
forControlEvents: .ValueChanged);
```

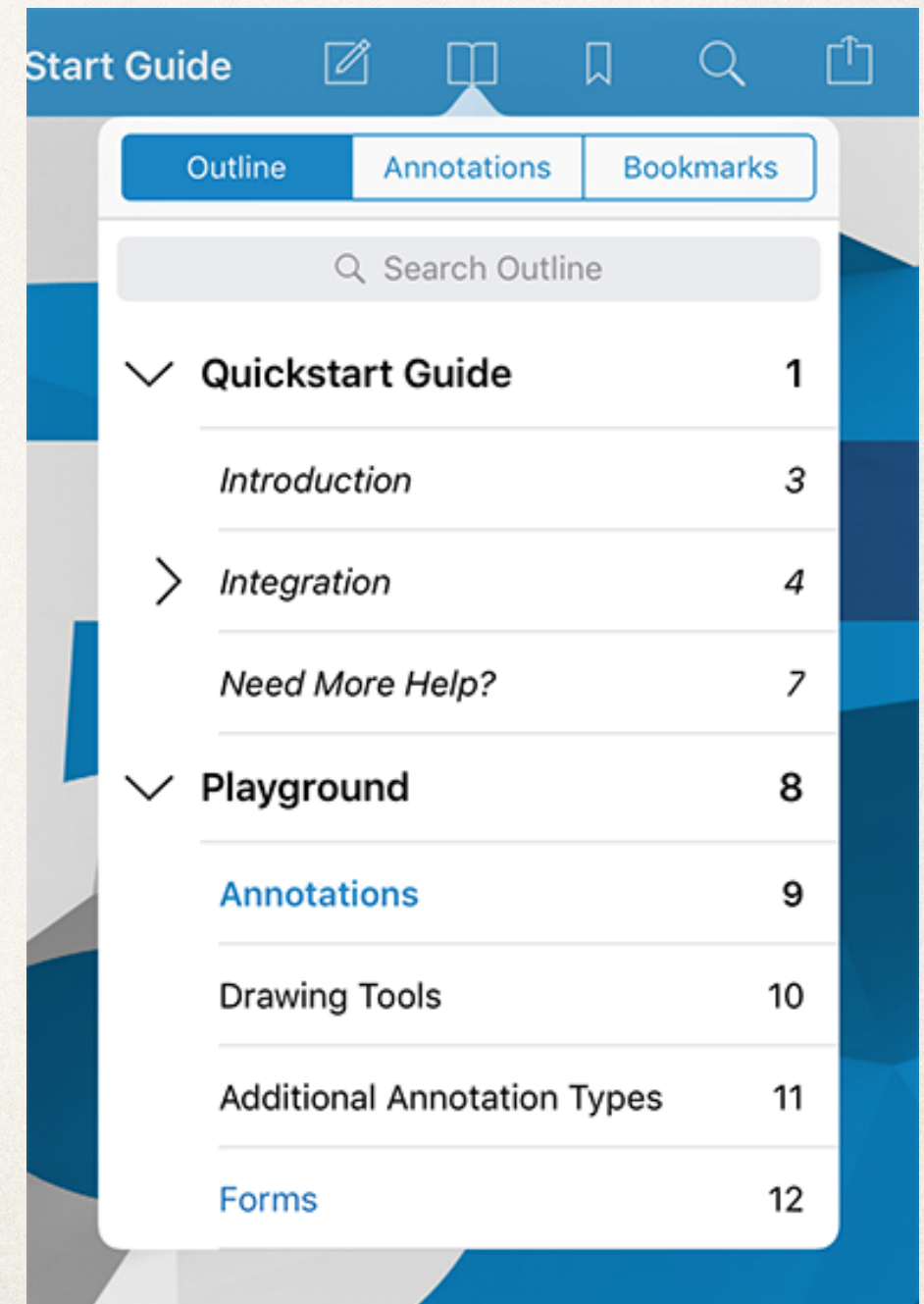
```
func action(sender: UISegmentedControl) {  
    switch sender.selectedSegmentIndex {  
        case 1: //do something  
        case 2: //do something  
        default: //do something  
    }  
}
```


Scroll Views

- ❖ Provides support for displaying content larger than the screen size
- ❖ Handles scroll functionality across views that support scrolling
- ❖ Handles zooming and panning of screen content
- ❖ Uses UIScrollViewDelegate

Popover Controllers

- ❖ Manages the presentation of content as a popover
- ❖ Information presented is temporary
- ❖ Popover only visible till user taps outside of it or explicitly dismisses it



Popover Behavior

- ❖ Implemented as a view controller using `UIPopoverPresentationController`
 - ❖ Set popover appearance
 - ❖ Set popover anchor point
 - ❖ Set popover arrow direction
- ❖ `UIPopoverPresentationControllerDelegate` used to access and dismiss the popover

Popover Delegate

- ❖ Popover delegate actions:
 - ❖ `prepareForPopoverPresentation`
 - ❖ `popoverPresentationControllerShouldDismissPopover`
 - ❖ `PopoverPresentationControllerDidDismissPopover`
- ❖ View controller must be presented using `UIModalPresentationStyle.popover`

Quiz Question!

- ❖ What defines a view's bounds?
 - A. The rectangle position within the superview's coordinate system
 - B. The rectangle position within the view's own coordinate system
 - C. The center position within the superview's coordinate system

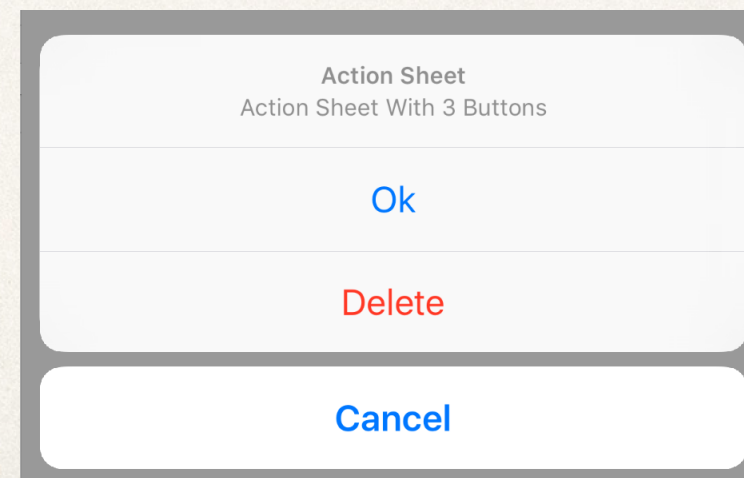
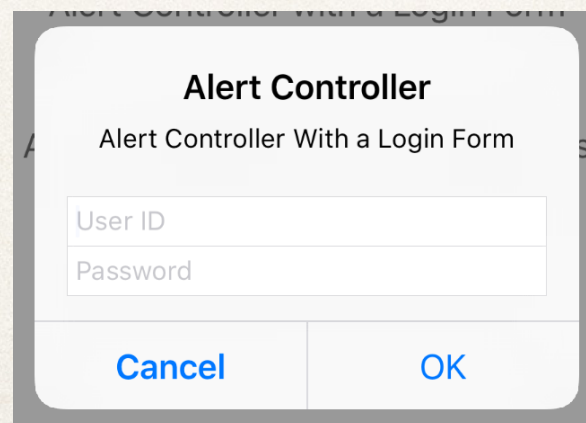
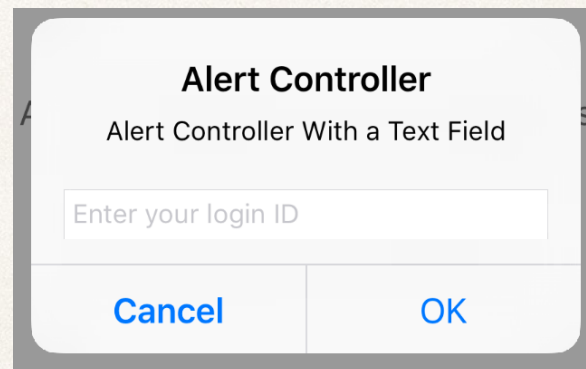
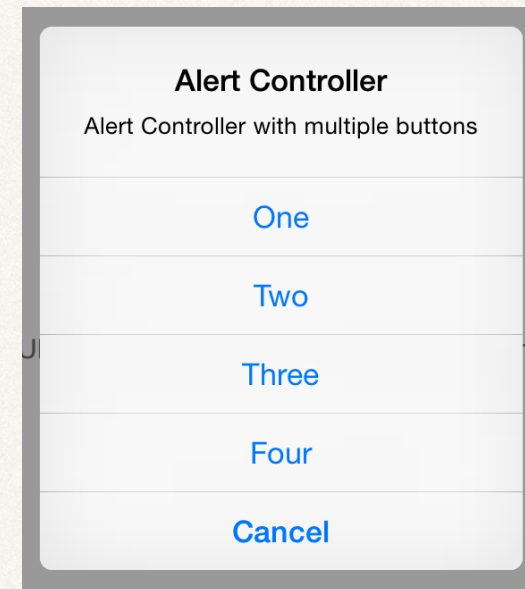
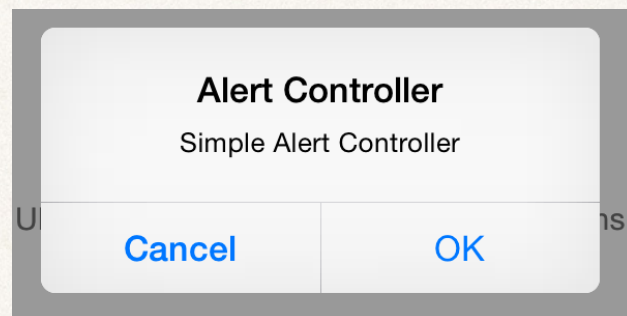
Alert Controllers

- ❖ Easy way to display info to user
- ❖ Uses `UIAlertController` and `UIAlertAction`
- ❖ Alert controller displays the message
- ❖ Alert action allows user to respond to message
 - ❖ Action has code that executes when button selected

Action Style Settings

- ❖ Defines the visual style of the alert's actions
- ❖ Default
 - ❖ Standard presentation of action button
 - ❖ Normal text for general customization
- ❖ Cancel
 - ❖ Style implies action will cancel operation
 - ❖ Only one button allowed
- ❖ Destructive
 - ❖ Style implies action with change or delete data

Alert View Examples



Presenting an Alert

```
let alert = UIAlertController(title: "My  
Alert", message: "This is an alert.",  
preferredStyle: .alert)
```

```
alert.addAction(UIAlertAction(title:  
NSString("OK", comment: "Default  
action"), style: .default, handler: { _ in  
NSLog("The \"OK\" alert occurred.") }))
```

```
self.present(alert, animated: true,  
completion: nil)
```


Presenting an Alert

```
let alert = UIAlertController(title: "My  
Alert", message: "This is an alert.",  
preferredStyle: .alert)
```

```
alert.addAction(UIAlertAction(title:  
NSString("OK", comment: "Default  
action"), style: .default, handler: { _ in  
NSLog("The \"OK\" alert occurred.") } ) )
```

```
self.present(alert, animated: true,  
completion: nil)
```


Swift and Closures

- ❖ Closures are self-contained blocks of functionality
 - ❖ This concept will come up in other areas of Swift development!
 - ❖ Numerous features expect the use of closures
- ❖ `handler: { _ in NSLog("The \"OK\" alert occurred.") }`
 - ❖ Code inside `{ }` contains the functionality that executes when an alert happens
 - ❖ Equivalent to:

```
alert.addAction(UIAlertAction(title: NSLocalizedString("OK",  
comment: "Default action"), style: .default, handler:  
myHandler))
```

```
func myHandler() { NSLog("The \"OK\" alert occurred.") }
```