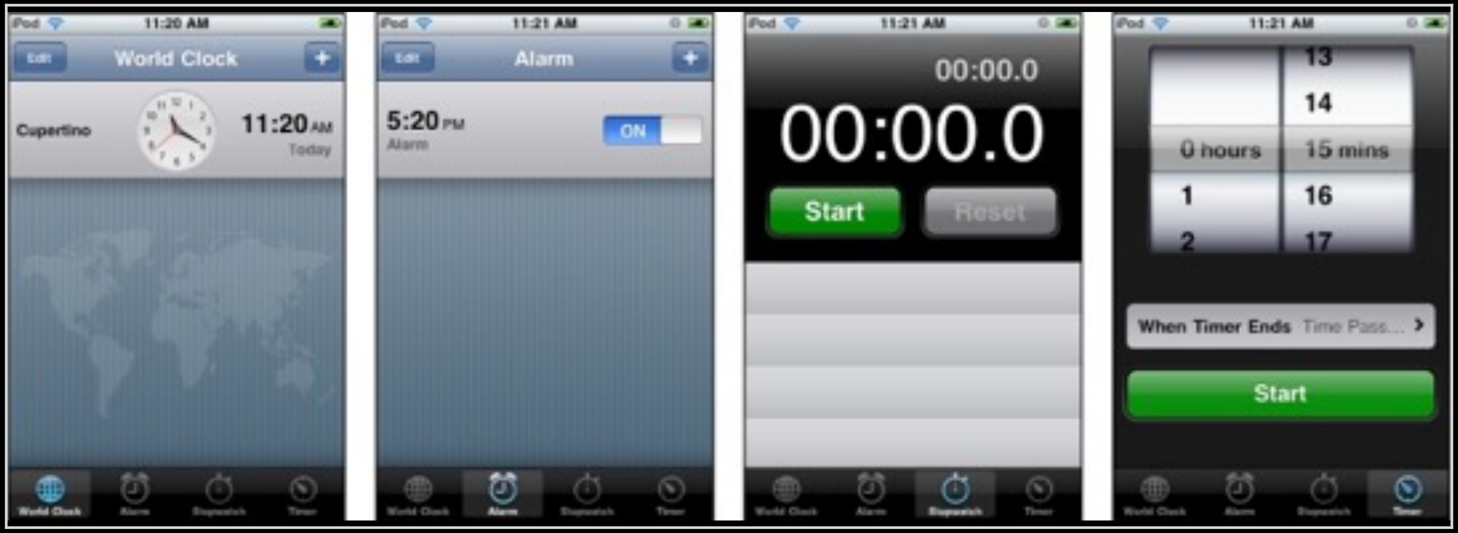
Guest Lecture

By: Adam Wallraff

UITabBarController, UICollectionView, and Basic Animations

UITabBarController

- UITabBarController is similar to UINavigationController in that it is meant to control, or hold, many other ViewControllers.
- To create an new instance of a UITabBarController with storyboards is almost identical to UINavigationController.
- Once created in storyboard, You connect each nested viewController by Ctrl +Dragging from the UITabBarController to the appropriate ViewController



UITabBarController

- Apple Docs - “To configure the tabs of a tab bar controller, you assign the view controllers that provide the root view for each tab to the viewControllers property. The order in which you specify the view controllers determines the order in which they appear in the tab bar. When setting this property, you should also assign a value to the selectedViewController property to indicate which view controller is selected initially. (You can also select view controllers by array index using the selectedIndex property.)”
- UITabBarControllers also have Delegate methods for further customization:

Managing Tab Bar Selections

- tabBarController:shouldSelectViewController:
- tabBarController:didSelectViewController:

Managing Tab Bar Customizations

- tabBarController:willBeginCustomizingViewControllers:
- tabBarController:willEndCustomizingViewControllers:changed:
- tabBarController:didEndCustomizingViewControllers:changed:

Overriding View Rotation Settings

- tabBarControllerSupportedInterfaceOrientations:
- tabBarControllerPreferredInterfaceOrientationForPresentation:

Supporting Custom Tab Bar Transition Animations

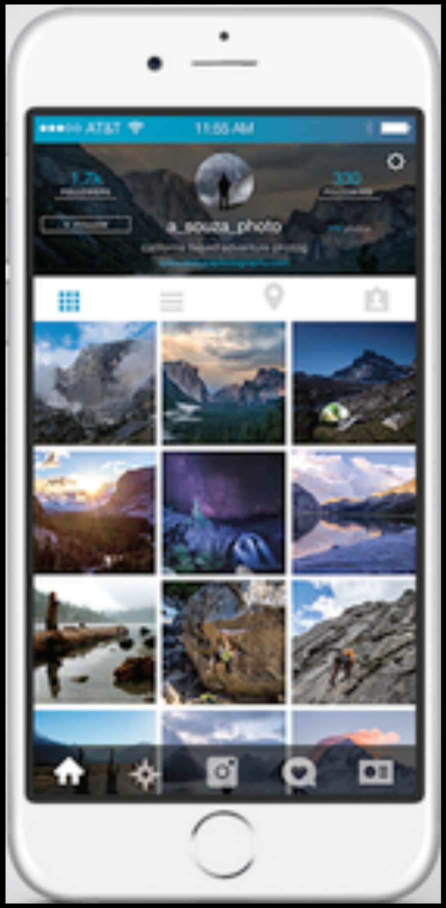
- tabBarController:animationControllerForTransitionFromViewController:toViewController:
- tabBarController:interactionControllerForAnimationController:

UITabBarController

DEMO

UICollectionView

- Apple - “Collection views provide the same general function as table views except that a collection view is able to support more than just single-column layouts.”
- So, now that we know how to use UITableView, UICollectionView is very similar.
- UICollectionView is a collection of UICollectionViewCell’s. Unlike UITableView, UICollectionView can scroll horizontally or vertically and instead of Rows we have Items.
- To use UICollectionView, we must conform to the proper UICollectionViewDataSource protocol. In this case cellForItemAtIndexPath and numberOfItemsInSection



```
func collectionView(collectionView: UICollectionView,
cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell

func collectionView(collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int
```

UICollectionView

- There are lots of additional ways to customize a UICollectionView by becoming the UICollectionViewDelegate.

- For today, we will focus on the previous 2 DataSource methods.

- One other big difference between table views and collection views is that with table views you are given some basic cell attributes upon creation. This is NOT the case with collection view cell's.

- Because of this, we will ALWAYS have to subclass UICollectionViewCell's.

UICollectionView

UICollectionViewDelegate Methods:

Managing the Selected Cells

collectionView(_:shouldSelectItemAtIndexPath:)
collectionView(_:didSelectItemAtIndexPath:)
collectionView(_:shouldDeselectItemAtIndexPath:)
collectionView(_:didDeselectItemAtIndexPath:)

Managing Cell Highlighting

collectionView(_:shouldHighlightItemAtIndexPath:)
collectionView(_:didHighlightItemAtIndexPath:)
collectionView(_:didUnhighlightItemAtIndexPath:)

Tracking the Addition and Removal of Views

collectionView(_:willDisplayCell:forItemAtIndexPath:)
collectionView(_:willDisplaySupplementaryView:forElementKind:atIndexPath:)
collectionView(_:didEndDisplayingCell:forItemAtIndexPath:)
collectionView(_:didEndDisplayingSupplementaryView:forElementOfKind:atIndexPath:)

Providing a Transition Layout

collectionView(_:transitionLayoutForOldLayout:newLayout:)

Managing Actions for Cells

collectionView(_:shouldShowMenuForItemAtIndexPath:)
collectionView(_:canPerformAction:forItemAtIndexPath:withSender:)
collectionView(_:performAction:forItemAtIndexPath:withSender:)

UICollectionView

DEMO

Basic Animations

- Animations add “Delight” to your apps.
- UIView’s have specific properties that can be animated.
- In addition, each UIView has a CALayer that has additional properties that you can animate.
- The most common way of animating is by using the corresponding class methods on UIView.
- There are more options, but these are the most common that I use. Feel free to check out the others on your own.

```
UIView.animateWithDuration(duration: NSTimeInterval, animations: () -> Void() -> Void)

UIView.animateWithDuration(duration: NSTimeInterval, animations:() -> Void() -> Void, completion:((Bool) ->
Void))

UIView.animateWithDuration(duration: NSTimeInterval, delay: NSTimeInterval, usingSpringWithDamping: CGFloat,
initialSpringVelocity: CGFloat, options: UIViewAnimationOptions, animations: () -> Void() -> Void, completion:
((Bool) -> Void))
```

Basic Animations

- The UIView properties that are animatable include:

.frame

.bounds

.center

.transform

.alpha

.backgroundColor

.contentStretch

Basic Animations

- **CALayer** - Each UIView is backed by a CALayer that has additional properties that can be animated. But First, what is a CALayer and how is it different then UIView?
- CALayer(Core Animation Layer) - CALayer functions as a backing layer for each UIView. The UIView then manages the layer within the view hierarchy. The View’s layer is responsible for managing the display and animation of its corresponding UIView.
- The primary features that the backing layer doesn't handle are touch events and user interaction. These are handled by UIView.

Basic Animations

DEMO