Model-View-Controller

Dr. Sarah Abraham

University of Texas at Austin
CS329e
Spring 2020

MVC

- Pattern of development to modularize features and design
- Objects have one of three roles:
  - Model
  - Viewer
  - Controller
- Object types separated by abstract boundaries and communicate across these boundaries

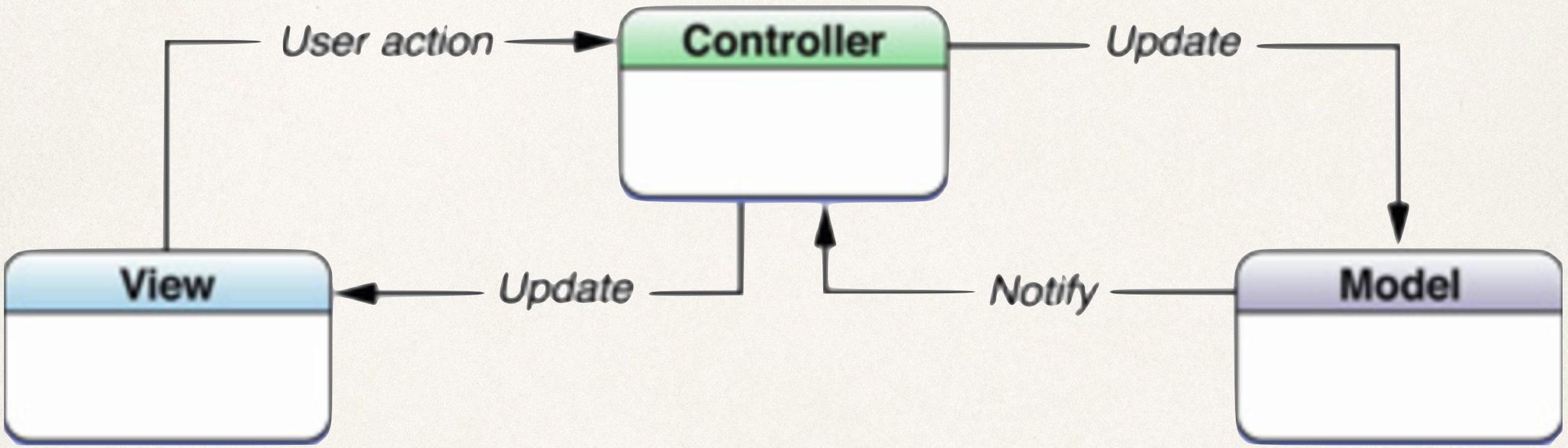
MVC Benefits

- Objects more reusable
- Interfaces better defined
- Applications more extensible
- Common pattern for interactive applications with GUI (graphical user interface)

MVC and Cocoa

- Cocoa designed around MVC model
- Good understanding of MVC leads to good design for Cocoa applications
- Custom objects in Cocoa applications must follow one of the MVC roles

MVC Pattern Flow



- How does the application receive and respond to this flow of events?

Event-driven Programming

- Events are triggered occurrences that the program receives and can respond to
- Event-driven programming allows for efficient handling of:
  - Device input
  - Timers
  - Event loops
- Events determine flow of the program based on user input, sensor output, or messages from other programs

Model Layer

- Defines logic and computation of the program
- Model objects encapsulate data specific to the application
  - Contains data loaded into app
  - Handles state of persistent data within the app
- Avoids explicit connection to view objects
  - No concerns about user-interface or presentation
  - Does not directly respond to user-input

Model Layer Communication

- User interfaces with view layer
  - Changes communicated via controller object to model layer
    - Based on event info, model object updates
- Backend database updates model object
  - Changes communicated via controller object to view layer
  - Based on event info, view objects update

View Layer

- Displays data from model objects to allow user to interact and modify this information
- View objects that are visible to the user
  - Draw themselves on the screen
  - Respond to user input
- UIKit and AppKit frameworks provide collections of view classes
- Interface Builder provides many view objects for building app GUI

View Layer Communication

- Controller objects notify view object about changes to model data

- User-initiated changes (buttons pressed, text-fields entered) passed from view layer to model layer via controller objects

Controller Layer

- Intermediary between one or more view objects and one or more model objects

- Conduits that allow view objects to learn about changes in model objects and vice versa

- Perform setup and coordinating tasks for an application

- Manage the life-cycles of other objects

Controller Layer Communication

- Interprets user actions made in view objects and communicates changes or new information to model layer

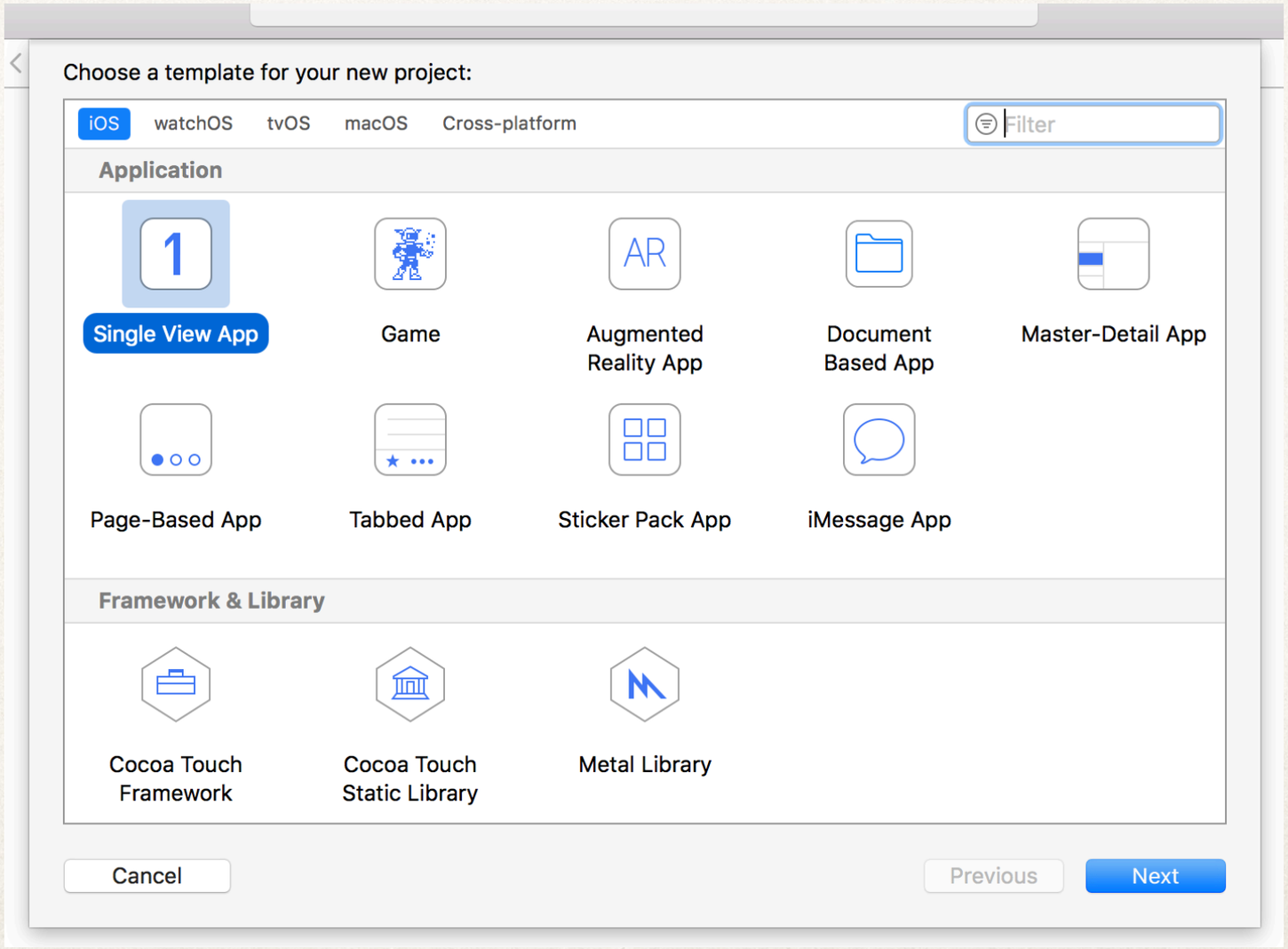
- Notified about changes to model objects and communicates new or updated data to the view objects for display

Using MVC with iOS

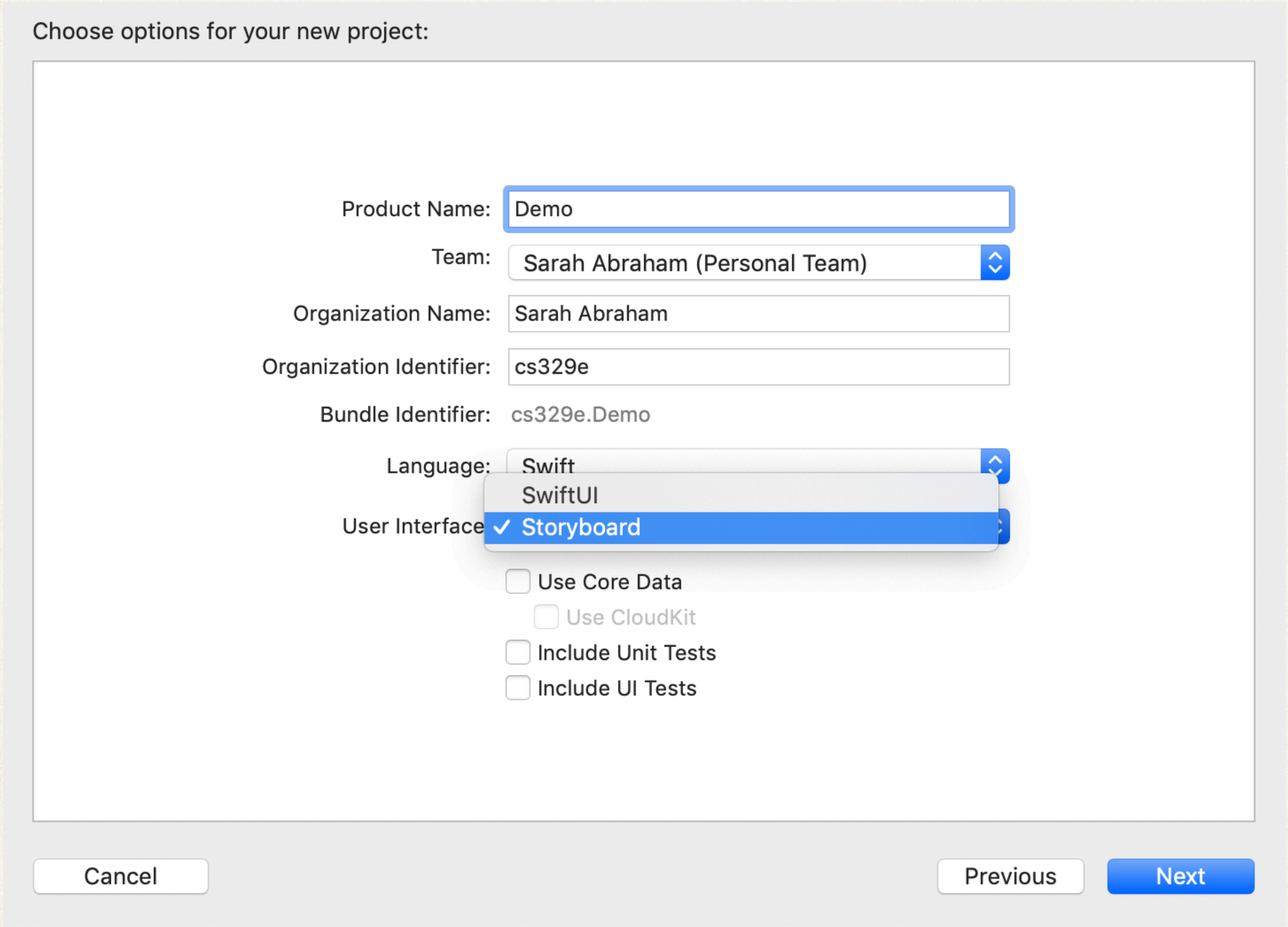
- iOS frameworks provide **2 of 3 MVC components**:
  - View Controllers
  - Views
- Model component custom-defined based on application purpose
- Views customized based on desired user-interface
- View Controllers customized based on required communication between models and views

Creating a View-based Application

- Select Single View iOS Application:

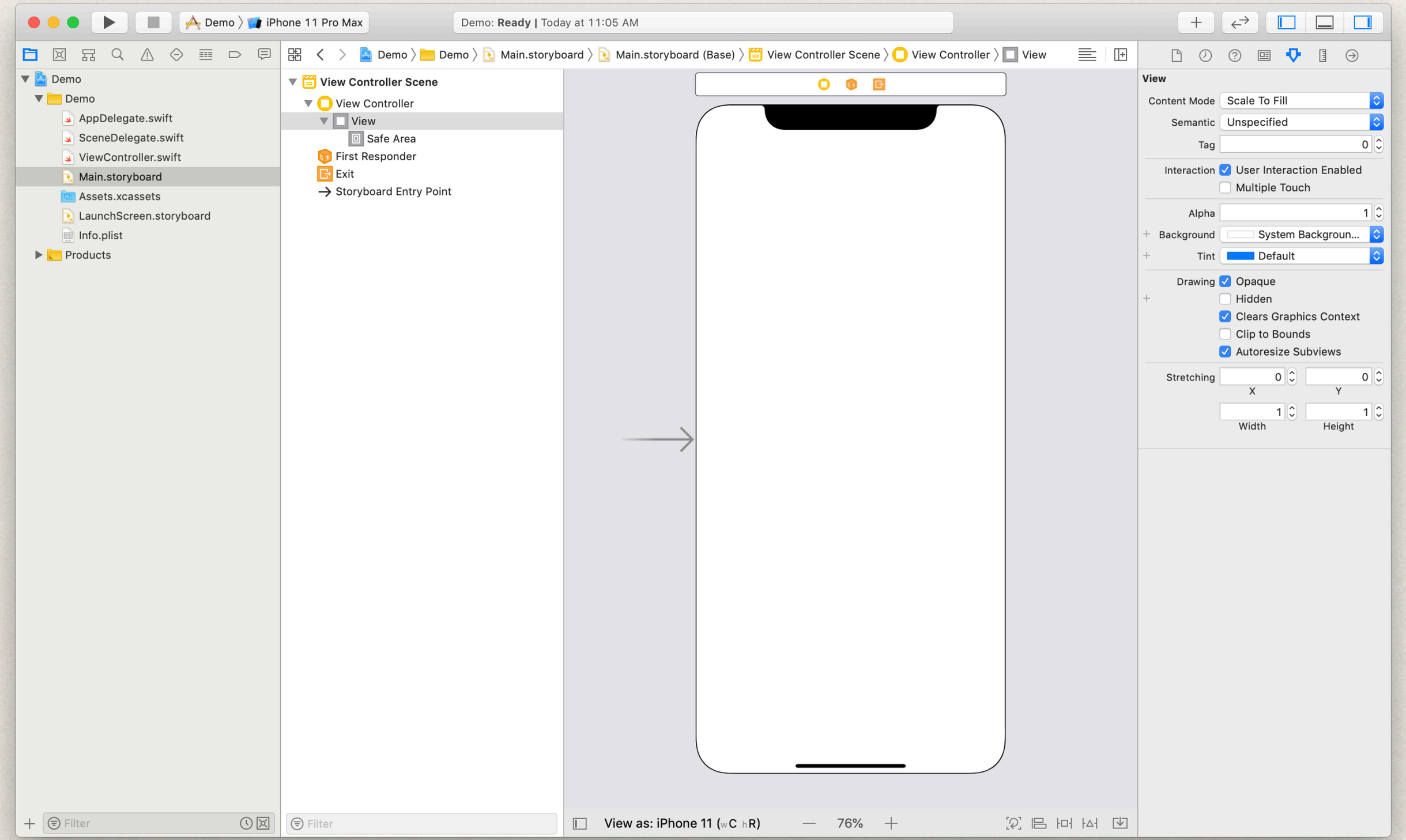


- Enter / select project options:



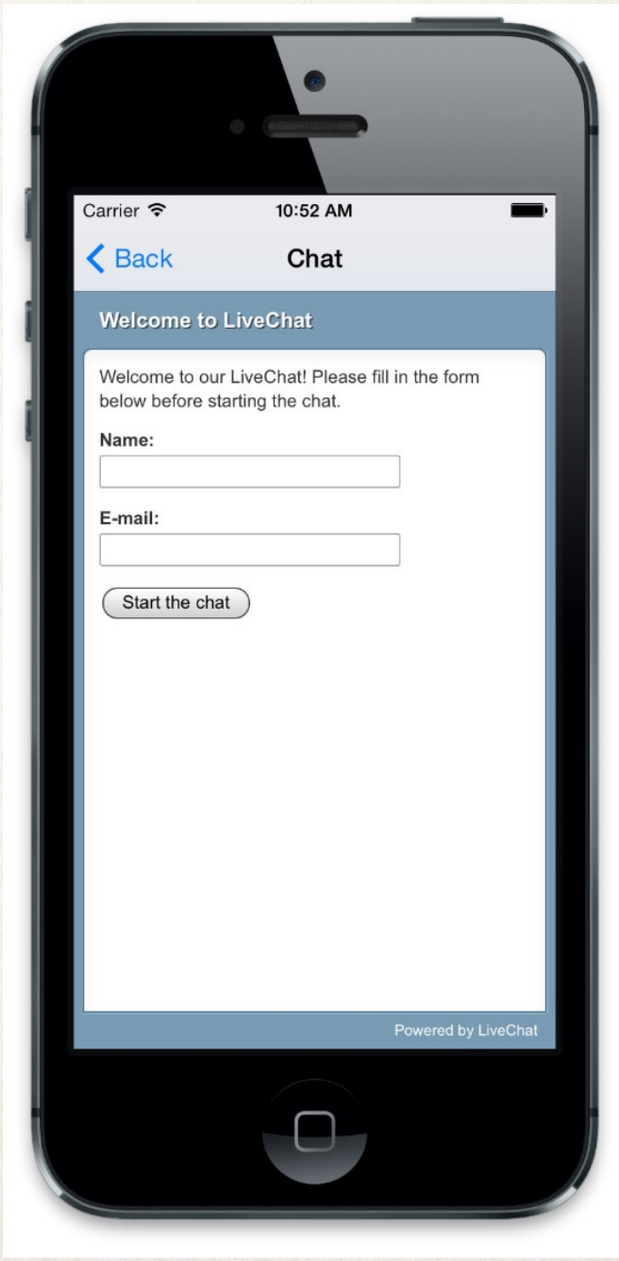
We will start with storyboards but eventually discuss SwiftUI as well

Project is ready for prototyping!



Constructing Views

- Display elements of user interface:
  - Buttons
  - Labels
  - Text fields
  - Sliders
  - Images
  - etc

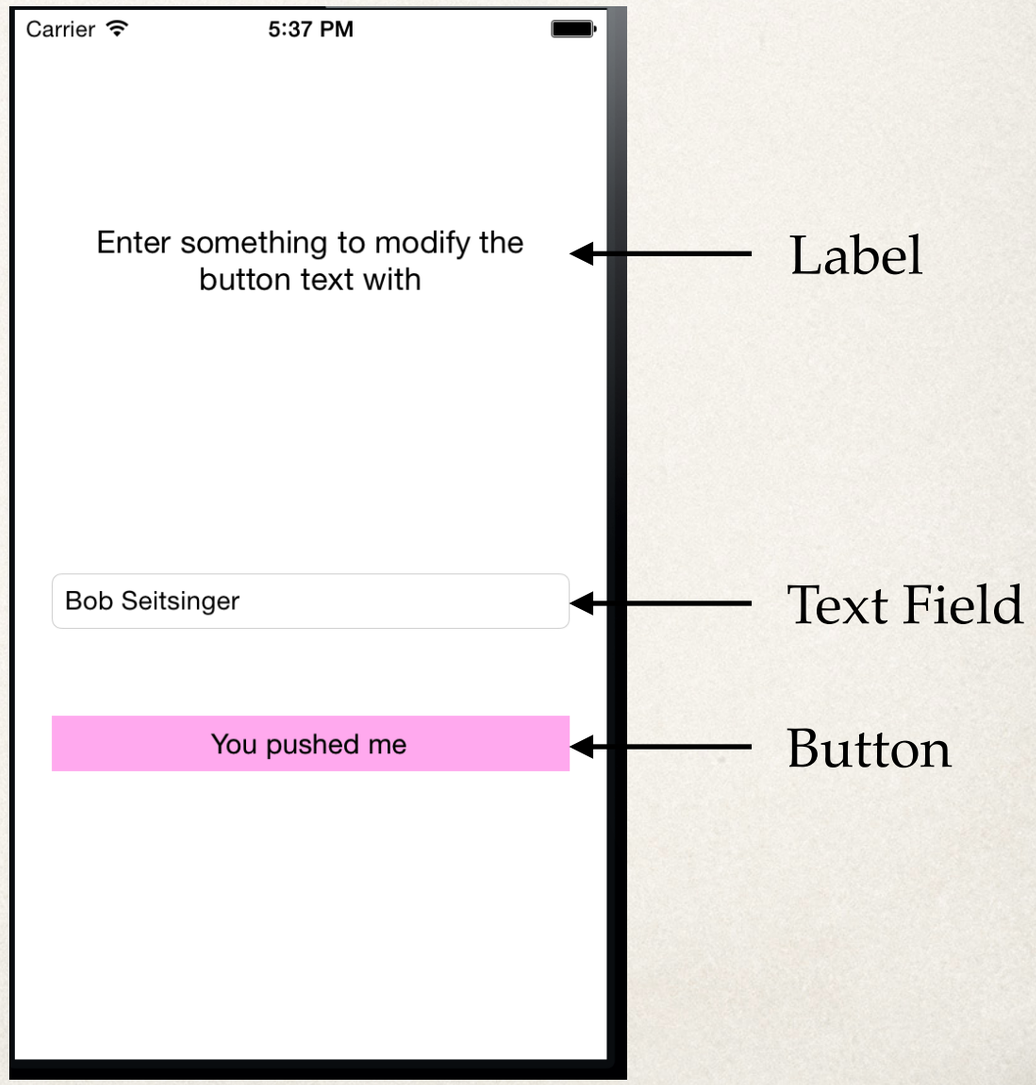


View Hierarchy

- Views can be composed of other views
- Base view (of view controller) has other views (buttons, labels, etc) added as child views
  - Establishes a view hierarchy
- Properties of views can inherit to subviews
  - e.g. if a view is hidden, its subviews are hidden

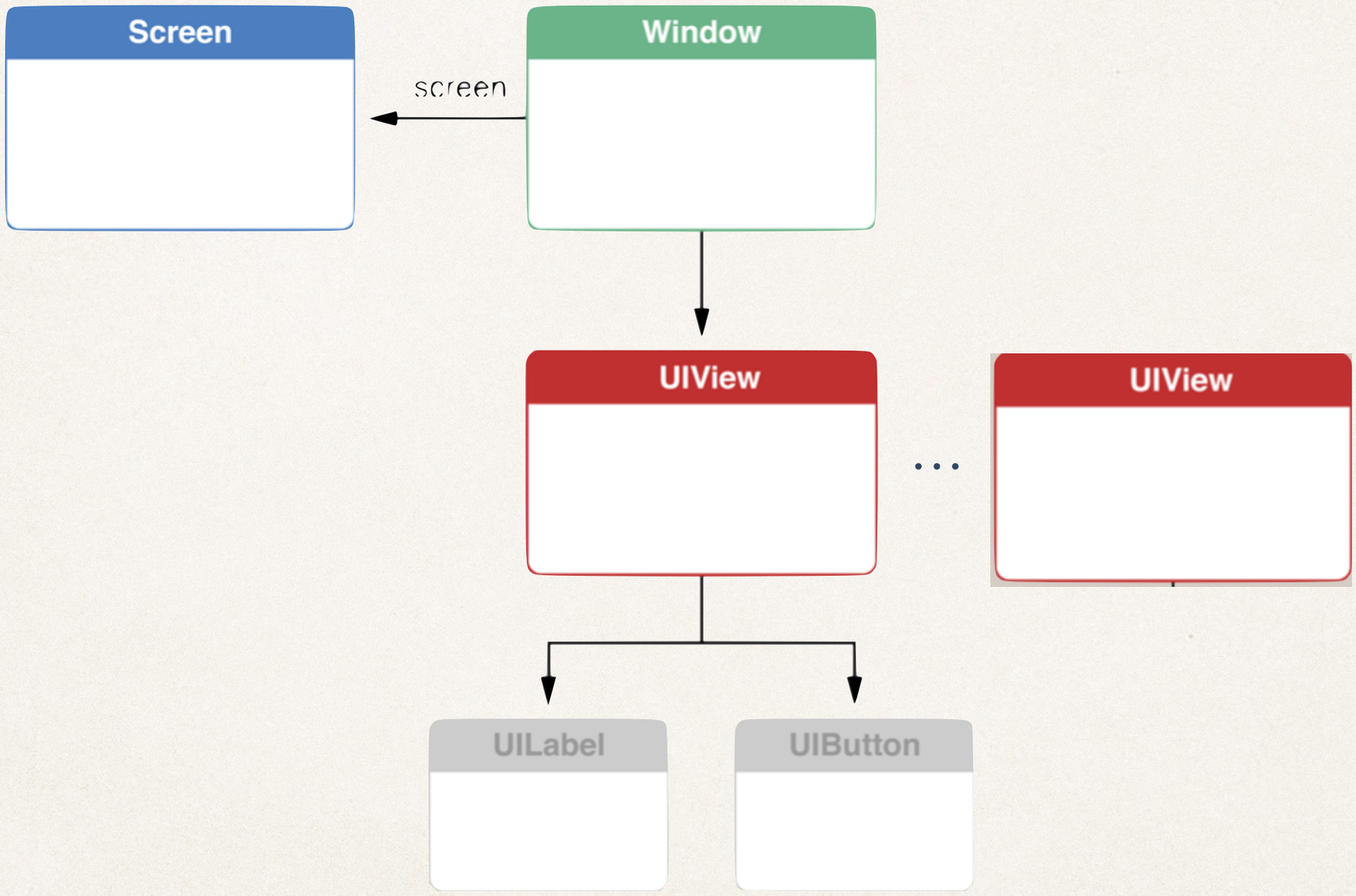
Widget Examples

- Simple application with 3 views within the main view:
  - Label (display text)
  - Text field (text input)
  - Button (initiates action)



WidgetExampleDemo

Window with target screen and content views



View Controllers

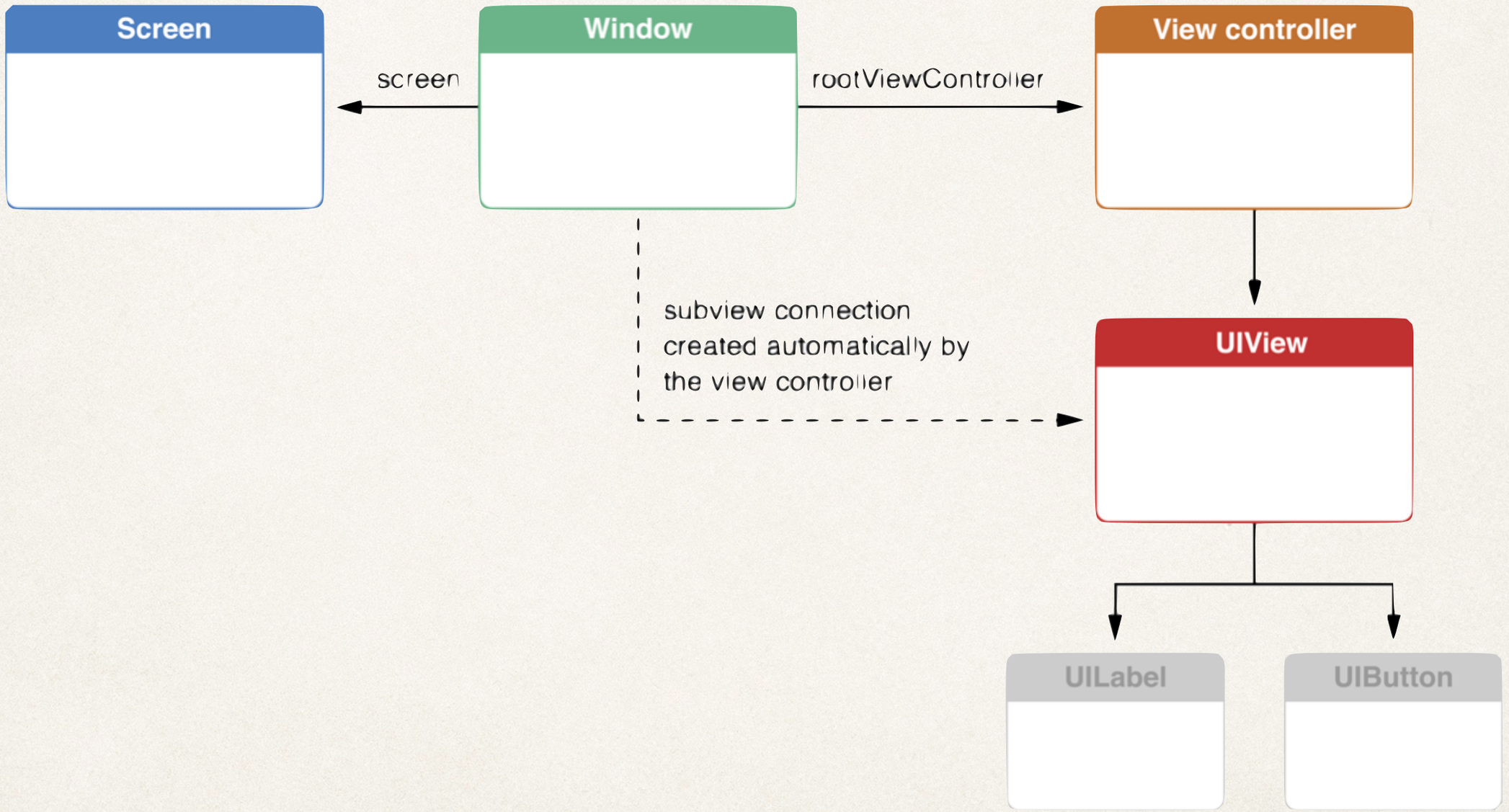
- Objects in iOS application that contain code for coordinating data and view components

- All view controllers derive from UIViewController class

- All iOS applications have at least one view controller

  - Typically one window per application

View controller attached to window automatically adds its views as window subviews

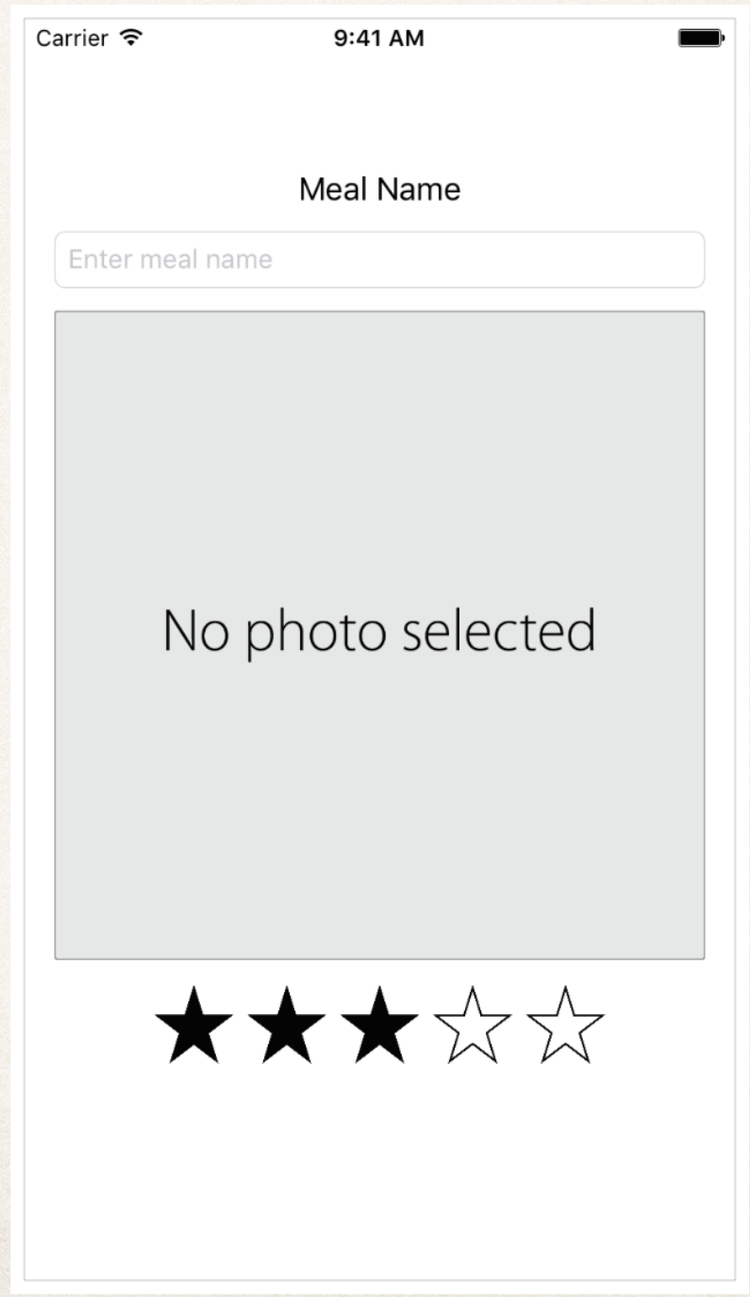


Why One Window?

- iPhone applications have limited screen real estate
- User interface broken into views that are managed by view controller
  - Only one chunk displayed at a time
- Less of an issue on tablets and larger phones
- iPad apps often make use of multiple windows

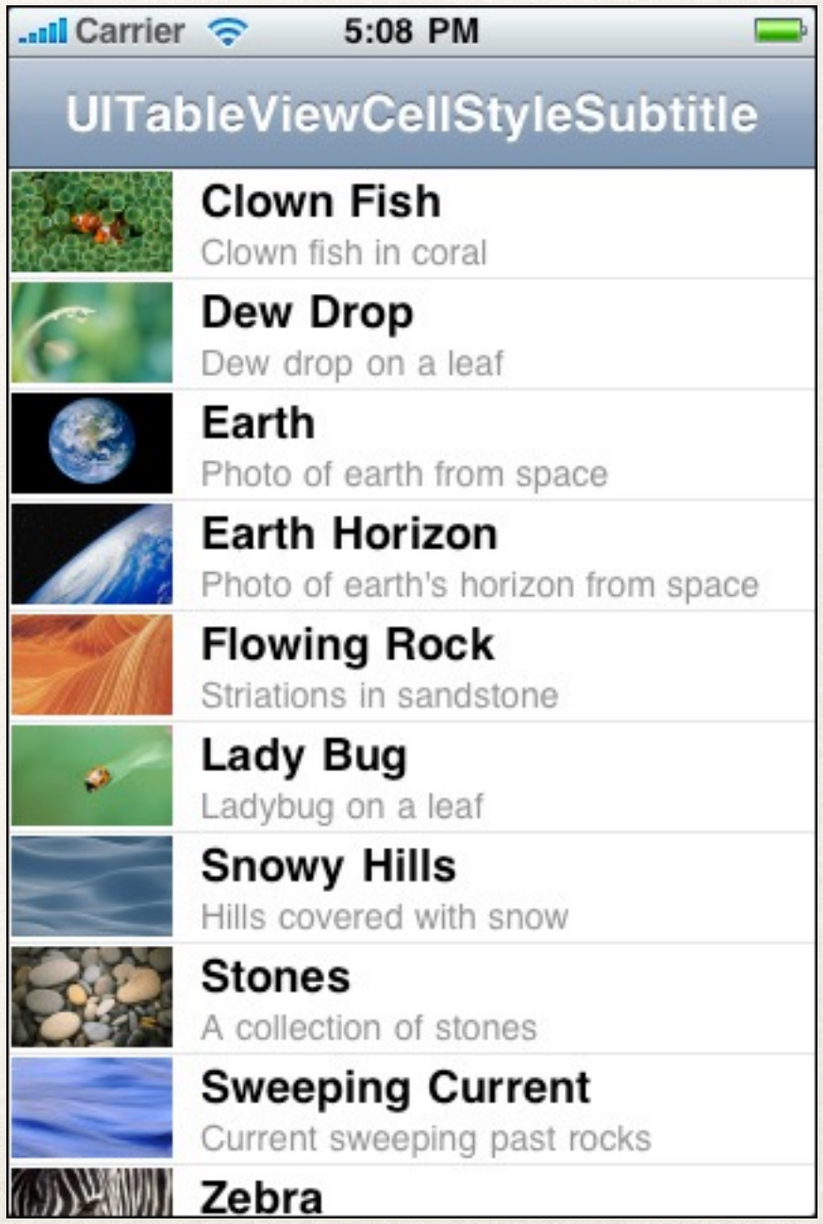
UIViewController

- Display a combination of views



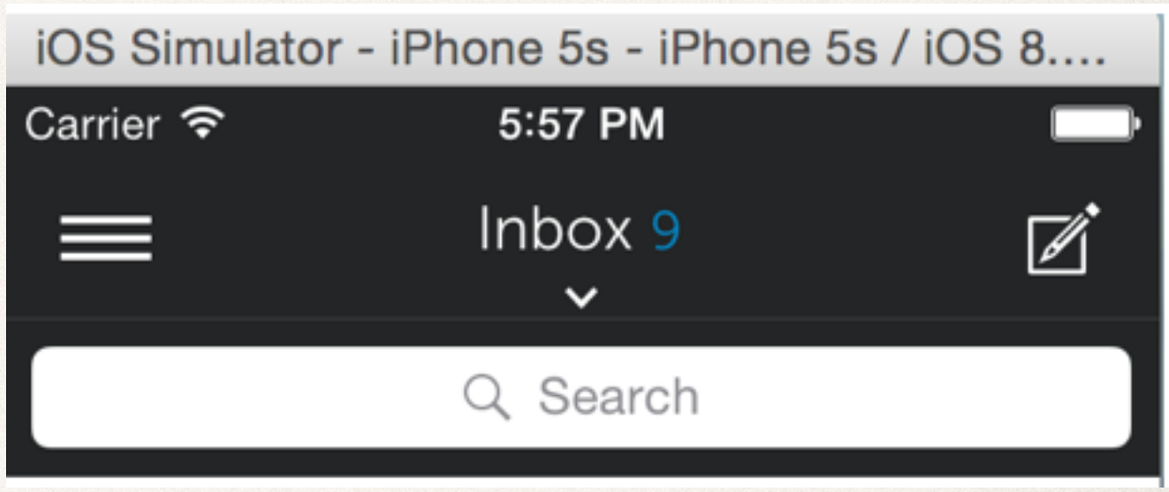
UITableViewController

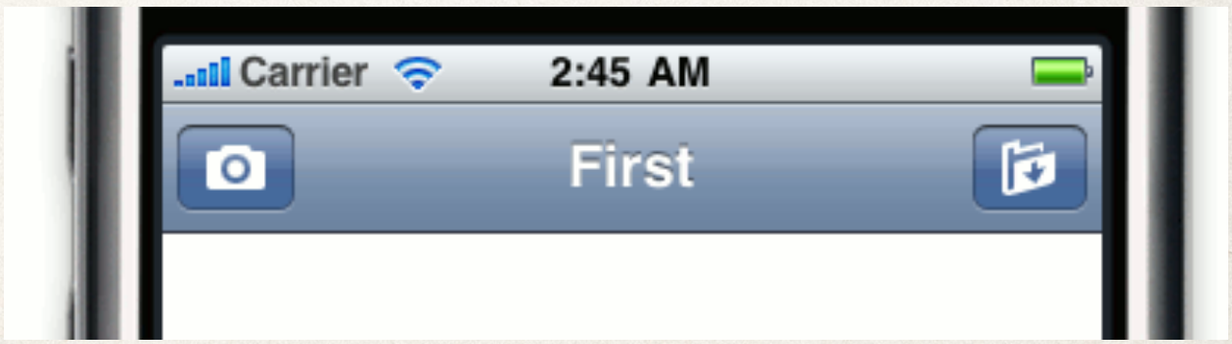
- Displays list of things in tabular form



UINavigationController

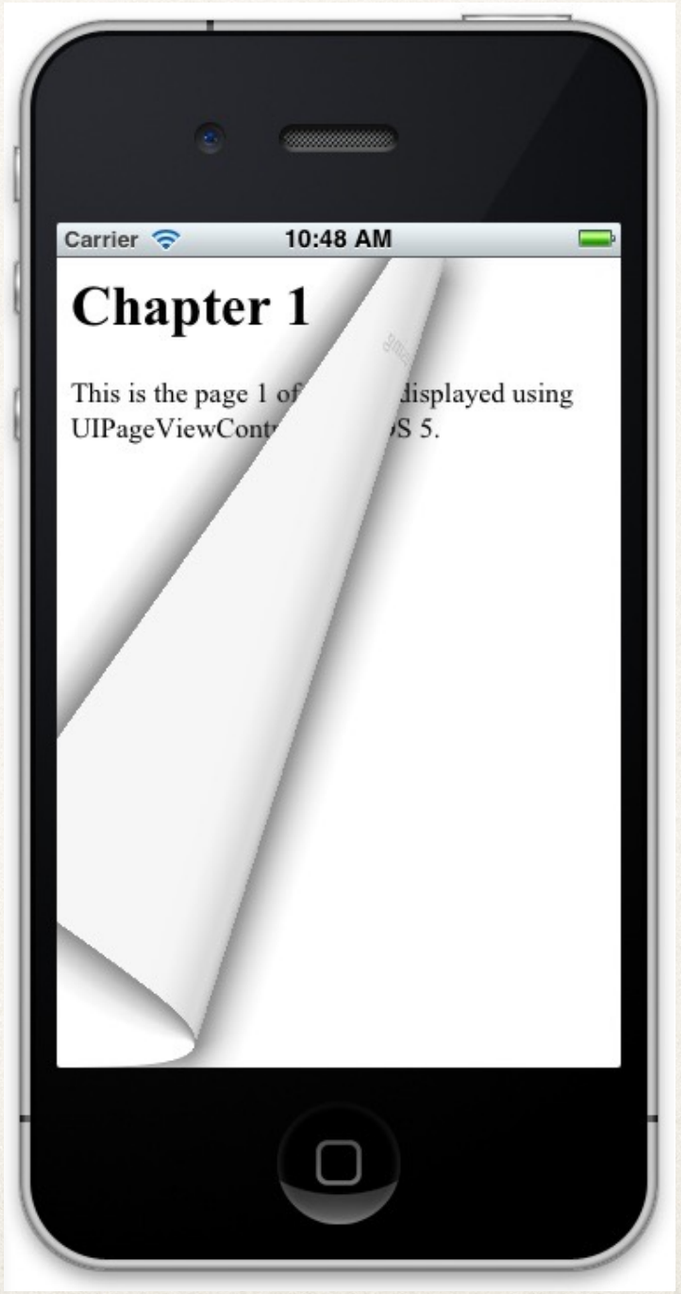
- Contains and coordinates *navigation* between view controllers





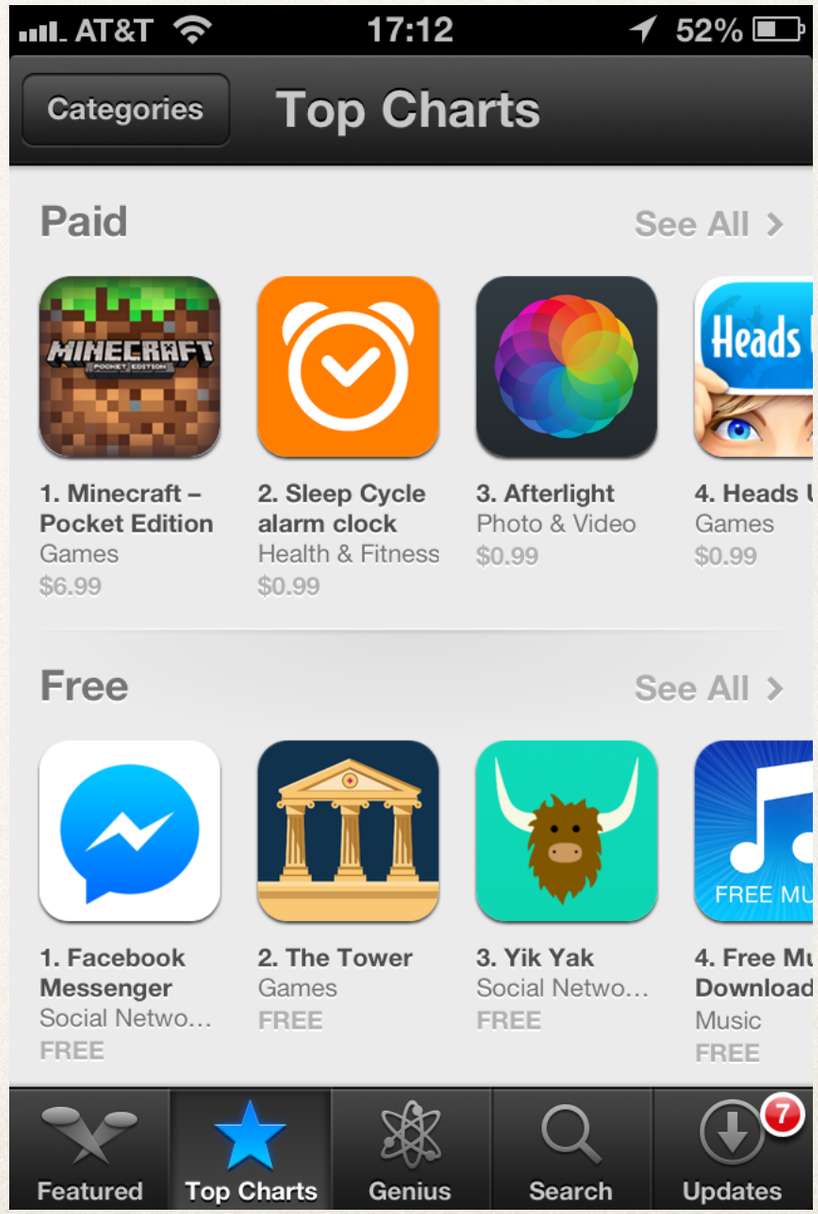
UIPageViewController

- Simulates the notion of flipping through pages



UITabBarController

- Provides tabs to *navigate* between view controllers



Quiz Question!

- True or false: each view requires its own, unique view controller to coordinate behavior with other views.