TabBarControllers

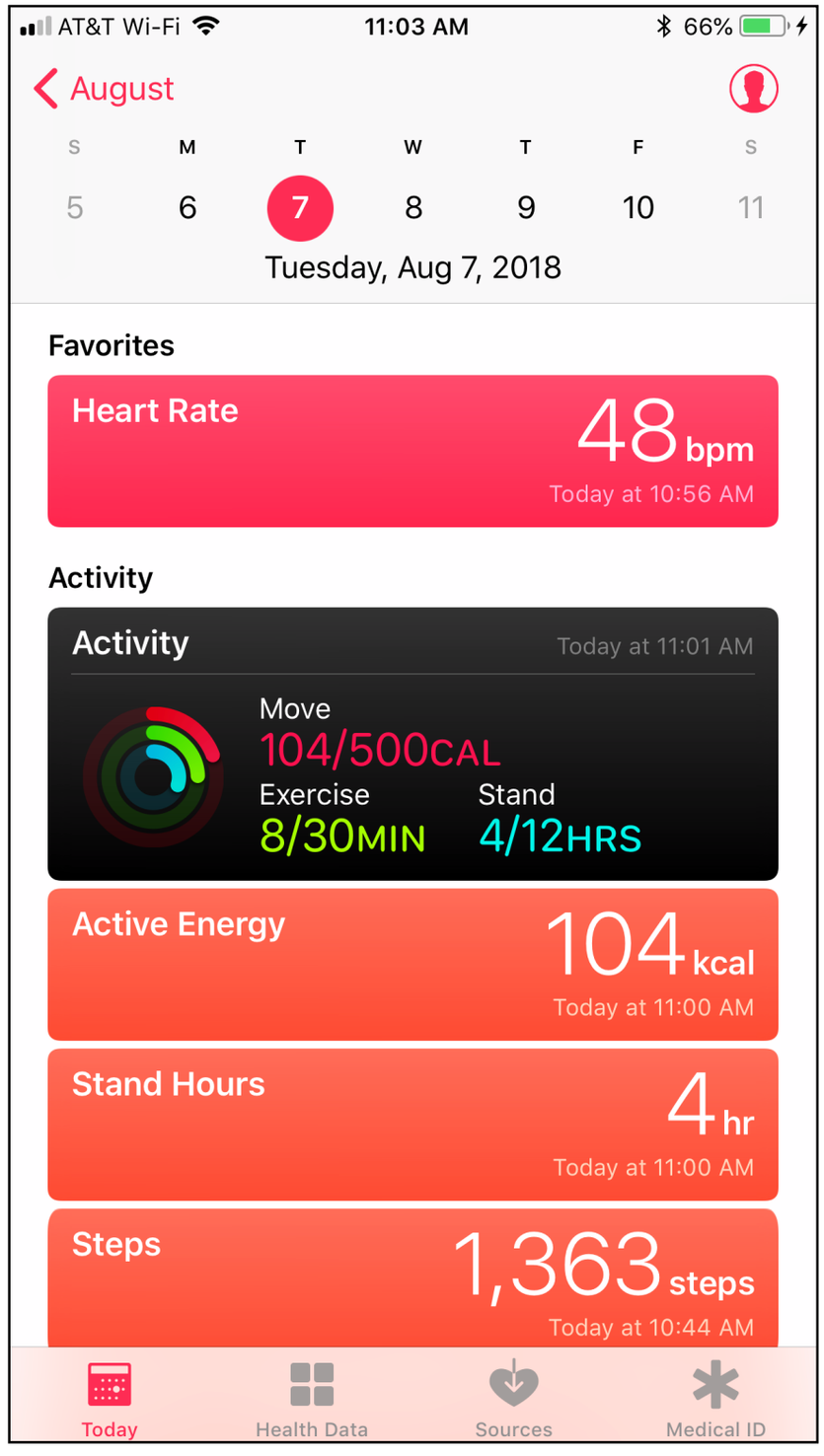
Mobile Computing - iOS

Objectives

- Students will be able to:
  - explain the purpose of tab bar controllers
  - create apps that use tab bar controllers
  - describe techniques for sharing information among view controllers in a tabbed application

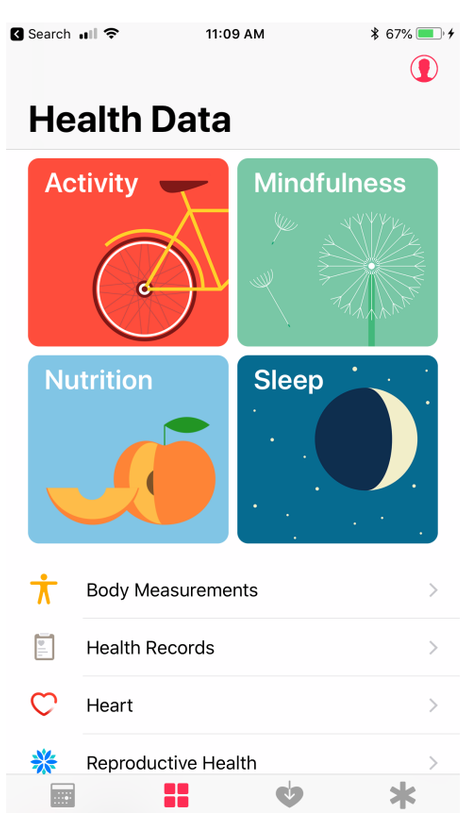
Tabbed Applications

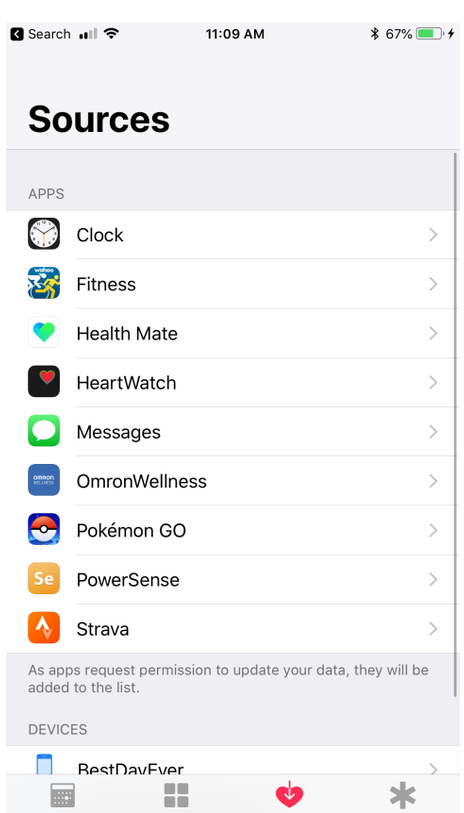
- A tabbed application provides an interface in which a user switches from view to view by tapping on a tab
- At times tabs may share content (e.g., in the Health app, Health Data needs to know Sources)
- Tabs may also be independent, just placed in 1 app because they are functionally related (e.g., Medical ID doesn't share/use data from any other tab, it's just a useful place to store things like organ donor preferences)

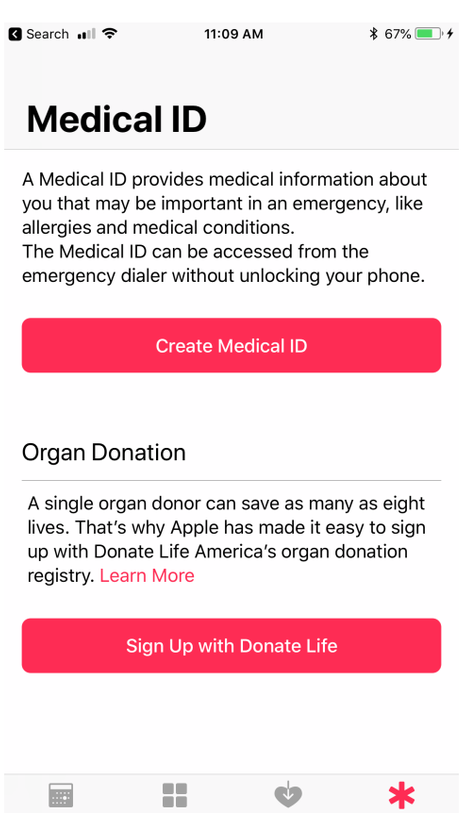


Tab Bar App Structure

- Each view is supplied by a different view controller (e.g., TodayViewController, HealthDataViewController, etc.).
- All the view controllers are contained in a tab bar controller, a **container** view controller. A tab bar controller's view consists of two parts, one for the tab bar, one for the content (the view of whichever view controller is currently selected)







Tab Bar Controller Properties

- A tab bar controller, a instance of the UITabBarController class, has several properties worth knowing about:

  - var delegate: UITabBarControllerDelegate?
    The tab bar controller's delegate object.

  - var tabBar: UITabBar
    The tab bar view associated with this controller.

  - var viewControllers: [UIViewController]?
    An array of the root view controllers displayed by the tab bar interface.

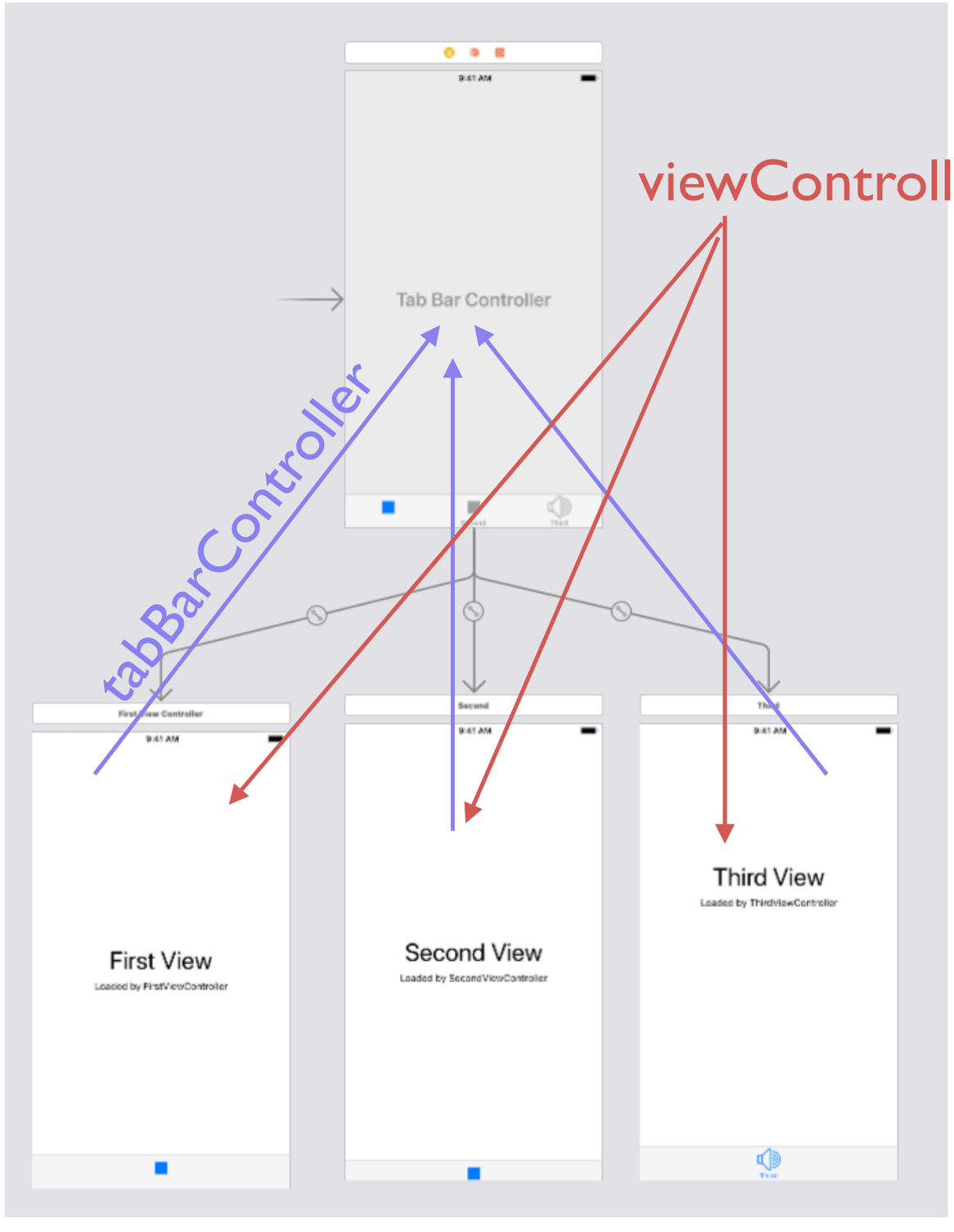
  - var selectedViewController: UIViewController?
    The view controller associated with the currently selected tab item.

  - var selectedIndex: Int
    The index of the view controller associated with the currently selected tab item.

Tab Bar Details

- Tabs are stored in a **UITabBar**
- Selecting a tab selects a view controller, whose view then becomes visible
- Each view controller has a **tabBarItem** property, consisting of **title**, **image** & **badge** properties. They are used, when a view controller is contained in a tab bar controller, to populate the tab bar.
- Fun fact! A view controller also has a **tabBarController** property. When housed in a tab bar controller, that property references it.
- Tab bar icons need to be monochrome with transparent backgrounds and sized appropriately.
- A good source of pre-made images is icons8.com or glyphish.com

Properties Galore!



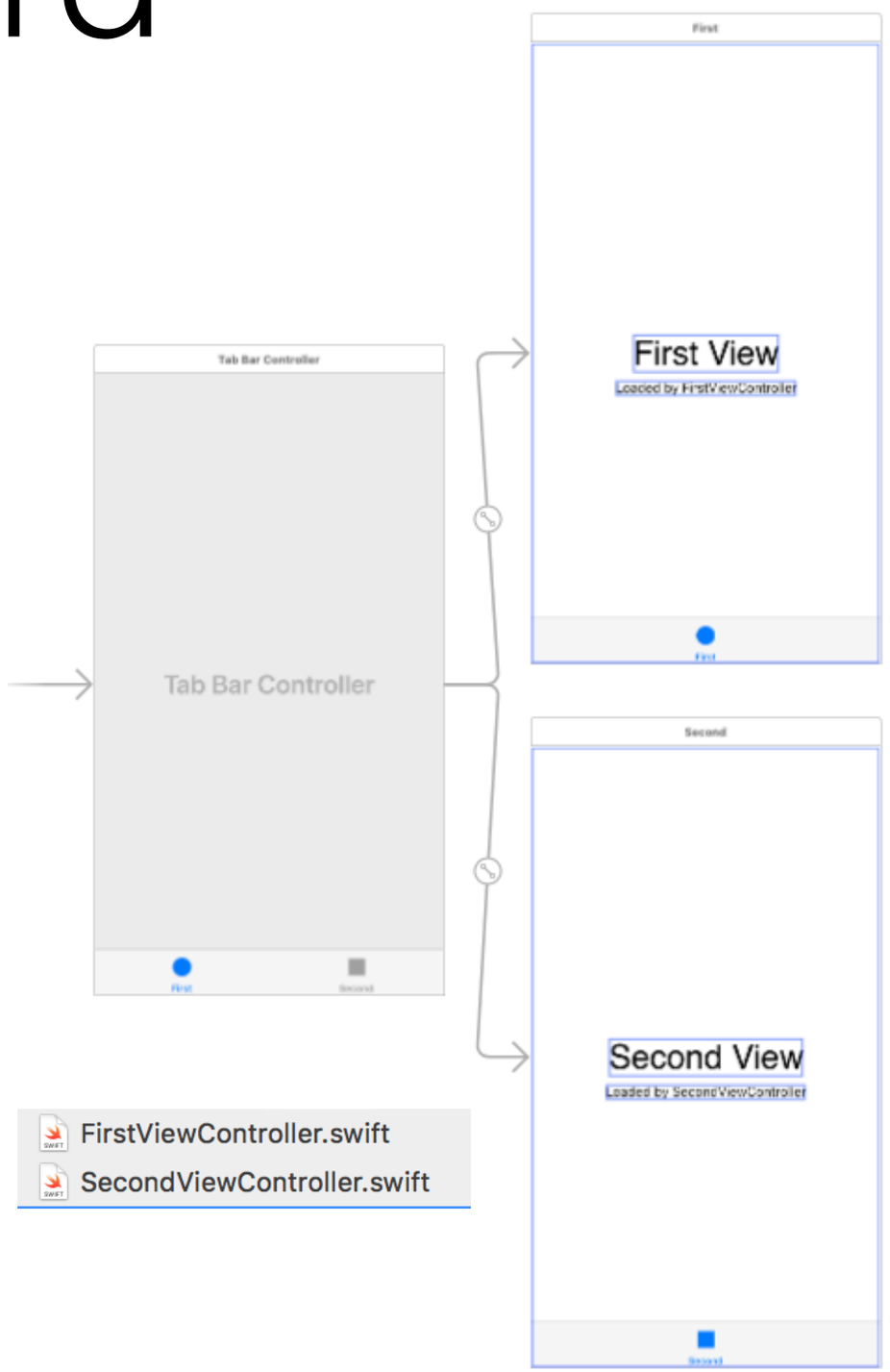
Construction

- UITabBarControllers can be specified in a storyboard or in code
- In the former cases, interface builder makes it easy to visualize connections

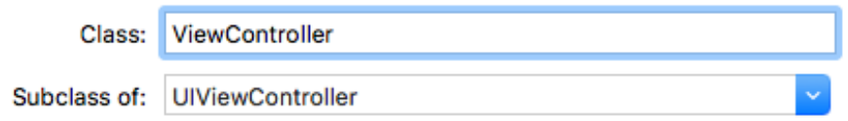
Setting up TabBars in Storyboard

1. Create a Tabbed Application

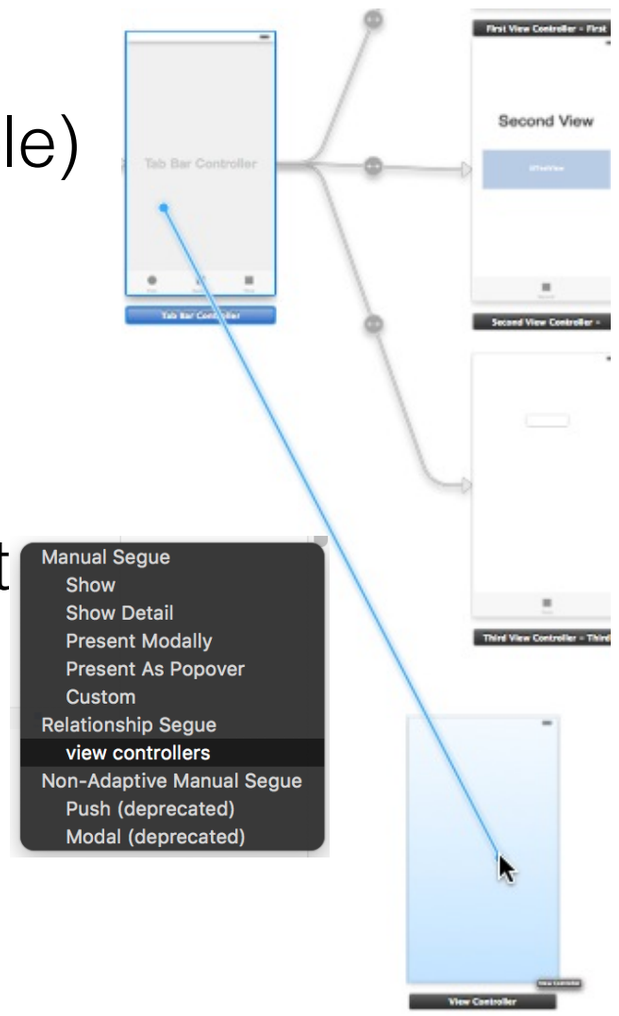
2. Template comes with 2 UIViewControllers, ready to configure



Adding Other UIViewControllers



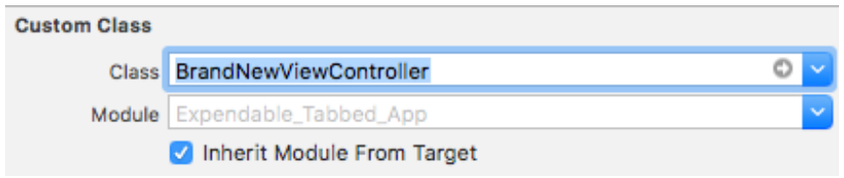
1. File > New > File (or ⌘-N)
    1. Select Cocoa Touch Class as the template
    2. Create a subclass of UIViewController
    3. Call it **BrandNewViewController** (for example)
2. Drag a view controller from the Object Library over the Storyboard window
3. Control-drag from the tab bar controller to the view controller (use relationship segue: view cont
    - This adds the view controller to the tab bar controller's viewControllers array



Configuring the New ViewController

1. Click on the newly-added ViewController in storyboard, called Item by default (on the yellow dot above the scene or in the document view)

2. In the Identity Inspector, change the class to BrandNewViewController



3. Click on the tab bar item (at the bottom of the scene)

4.In the Attributes Inspector, change the title and image

- Use Tab Bar Item for system items, or Bar Item for custom items

Strategies for Storing Model Data

a static instance in the Model class. This uses the Global visibility of static members of a class. You will need a static function to get the instance of the Model which has the responsibility of creating the instance on first access. (Singleton)

1. Keep the data in the view controller classes. Data is close to the tab that needs it, but difficult to share. Not recommended.
2. Create a separate model class and follow one of these approaches:
    1. **Declare a reference** to an instance of the model (struct or class) **in the AppDelegate.** The AppDelegate will create the instance in application(didFinishLaunching). To access the delegate, use `UIApplication.shared.delegate as! AppDelegate`
    2. **Define a static instance** of the model (struct or class) **in the AppDelegate.** The Model is created on start of application. Static members of AppDelegate are visible from any place in our application.
    3. **Define a static instance** of the model (struct or class) **in its own file.**
    4. **Use a singleton**, a design pattern that allows only one instance to be created.

A Model Struct

- Every tournament has a name, date, and collection of players

```
struct Tournament {
    var tournamentName:String
    var date:Date
    var players:[Player]

    init(tournamentName:String){
        self.tournamentName = tournamentName
        date = Date()
        players = []
    }
}
```

Strategy # 1: Declare a Reference in the AppDelegate

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

    var window: UIWindow?
    var tourney:Tournament

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

        tourney = Tournament(tournamentName: "Bearcat 2019") // a reference in the app delegate
        return true
    }
```

```
class ScheduleViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        print((UIApplication.shared.delegate as! AppDelegate).tourney.date)
    }
```

Use this in any other view controller, contained in the tab bar controller, that needs to access App Delegate's tourney

To make the code shorter/more legible, we could *contemplate* storing it in a property that we define in the other view controllers:

```
// in all other view controllers that the tab bar controller contains
var tourney:Tournament = (UIApplication.shared.delegate as! AppDelegate).tourney //
```

Then, we could just write tourney.*whatever* in each view controller -- much shorter! Alas, **this won't work**. One feature of structs is that they use copy-by-value. Each tourney, in each other view controller, would be a distinct copy.

Strategy # 2: Define a Static Instance in the AppDelegate

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    static var tourney:Tournament = Tournament(tournamentName: "Bearcat 2019")

    func application(_ application: UIApplication,
          didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }
```

```
class ScheduleViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        print(AppDelegate.tourney.date)
    }
```

Now you can write AppDelegate.tourney in any view controller in the code where you need to access the tourney

Strategy # 3: Declare a Static Instance In Its Own File

Tournament.swift

```
struct Tournament {

    static var tourney = Tournament()

    var tournamentName:String
    var date:Date
    var players:[Player]

    init(){
        self.init(tournamentName: "")
    }
    init(tournamentName:String){
        self.tournamentName = tournamentName
        date = Date()
        players = []
    }
}

struct Player {


}
```

Just write Tournament.tourney any where in your code: all tabs can access it

Encapsulation

- Good coding practice dictates we make stored properties private, and only allow access through methods such as getPlayer(), addPlayer(), etc.

```
struct Tournament {

    static var tourney = Tournament()

    var tournamentName:String
    var date:Date
    private var players:[Player]

    init(){
        self.init(tournamentName: "")
    }
    init(tournamentName:String){
        self.tournamentName = tournamentName
        date = Date()
        players = []
    }

    mutating func add(player:Player) -> Void {
        players.append(player)
    }

    func getPlayer(_ playerNum:Int) -> Player? {
        if playerNum >= 0 && playerNum < players.count {
            return players[playerNum]
        } else {
            return nil
        }
    }

    mutating func deletePlayer(_ playerNum:Int) -> Bool {
        if playerNum >= 0 && playerNum < players.count {
            players.remove(at: playerNum)
            return true
        } else {
            return false
        }
    }
}
```

Overachievers! Read up on subscripts to see how you could improve this further.

Strategy # 4: Singletons

- Suppose we want to guarantee that a struct has exactly one instance with global visibility: it might be confusing and awkward if we have two model structs (or classes)
- This is a job for ... the **singleton** design pattern. It guarantees that there will only be one instance of a model

Singletons in Apple's Frameworks

- Apple often names its singletons "shared", "default"

```swift
// Shared URL Session
let sharedURLSession = URLSession.shared
// Default File Manager
let defaultFileManager = FileManager.default
// Standard User Defaults
let standardUserDefaults = UserDefaults.standard
// Default Payment Queue
let defaultPaymentQueue = SKPaymentQueue.default()
```

Making a Singleton

- Starting with your basic class, there are a few things we need to add to our class… and one thing to modify.
- Specifically we need to

1. make a static private stored property that stores the actual instance,
2. a public, read-only computed property that controls access to that instance
3. privatize all initializers to stop anyone else from making an instance

```swift
struct Tournament {

    static var tourney = Tournament()

    var tournamentName:String
    var date:Date
    private var players:[Player]

    init(){
        self.init(tournamentName: "")
    }
    init(tournamentName:String){
        self.tournamentName = tournamentName
        date = Date()
        players = []

    }

    mutating func add(player:Player) -> Void {
        players.append(player)
    }

    subscript (index:Int) -> Player {
        return players[index]
    }

    // etc.
}
```

Anywhere in code, just write Tournament.shared to get to the one and only Tournament instance.

```swift
struct Tournament {

    private static var _shared:Tournament!

    static var shared:Tournament {
        if _shared == nil {
            _shared = Tournament()
        }
        return _shared
    }

    var tournamentName:String
    var date:Date
    private var players:[Player]

    private init(){
        self.init(tournamentName: "")
    }
    private init(tournamentName:String){
        self.tournamentName = tournamentNam
        date = Date()
        players = []

    }

    mutating func add(player:Player) -> Voi
        players.append(player)
    }

    subscript (index:Int) -> Player {
        return players[index]
    }

    // etc.
}
```

A Template for Singleton Code

```
struct Model {
    private static var _shared:Model!
    static var shared: Model {    // Everyone can see it
        if _shared == nil {       // only created once
            _shared = Model()     // only we can make it
        }
        return _shared            // return it
    }

    // Singleton taken care of, now make any initializers private
    private init(){
        attribute = 0         // initialize the models
    }

    // And everything else is as before.
    var attribute:Int = 20
    func method() {
        print("My attribute is \(attribute)")
    }
}
```

Using the Singleton

```
print(Model.shared.attribute)       // attribute was initialized to 0
Model.attribute = 10                // we set attribute to 10
```

TabBarControllers in Code

- How does storyboard work its magic? Let's peer behind the curtain ...
- The sample code for this may be found in TabBarControllersWithoutStoryboard

UITabBarController in Code

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

    self.window = UIWindow(frame: UIScreen.main.bounds) // all life begins in a window
    let left = LeftViewController()                     // let's make our view controllers
    let right = RightViewController()
    let tabBarController = UITabBarController()         // and a tab bar controller to store them
    tabBarController.viewControllers = [left,right]     // now we create the relationship segue
    self.window?.rootViewController = tabBarController  // make the tab bar controller appear
    self.window?.makeKeyAndVisible()                    // make the window appear
    return true
}
```

Creating Views in Code

```
override func loadView() {
      // Create Views
      let view = UIView()
      view.autoresizingMask = [.flexibleHeight, .flexibleWidth]
      view.backgroundColor = UIColor(red: 1, green: 1, blue: 1, alpha: 1)

      let label = UILabel()
      label.textAlignment = .natural
      label.lineBreakMode = .byTruncatingTail
      label.baselineAdjustment = .alignBaselines
      label.text = "Big Sur"
      label.contentMode = .left
      label.isOpaque = false
      label.setContentHuggingPriority(UILayoutPriority(rawValue: 251), for: .horizontal)
      label.setContentHuggingPriority(UILayoutPriority(rawValue: 251), for: .vertical)
      label.font = .systemFont(ofSize: 17)
      label.textColor = nil

      let button = UIButton(type: .roundedRect)
      button.titleLabel?.lineBreakMode = .byTruncatingMiddle
      button.isOpaque = false
      button.setTitle("Energize", for: .normal)

      let myBigSurTBL = UITableView(frame: CGRect(x: 20, y: 60, width: 300, height: 422), style: .grouped)
      myBigSurTBL.separatorStyle = .singleLine
      myBigSurTBL.rowHeight = -1
      myBigSurTBL.sectionHeaderHeight = 18
      myBigSurTBL.sectionFooterHeight = 18
      myBigSurTBL.alwaysBounceVertical = true
      myBigSurTBL.clipsToBounds = true
      myBigSurTBL.backgroundColor = UIColor.groupTableViewBackground

      // Assemble View Hierarchy
      view.addSubview(label)
      view.addSubview(button)
      view.addSubview(myBigSurTBL)
```

Creating Views in Code, Cont'd

```
    // Configure Constraints

label.topAnchor.constraint(equalTo: view.topAnchor, constant: 45)// = view.topAnchor + 45
label.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor, constant: 166)// = view.leadingAnchor + 166

    myBigSurTBL.heightAnchor.constraint(equalToConstant: 422)
    myBigSurTBL.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor, constant: 16) //= view.leadingAnchor + 16
    myBigSurTBL.topAnchor.constraint(equalTo: label.bottomAnchor, constant: 39)// = label.bottomAnchor + 39

    button.topAnchor.constraint(equalTo: myBigSurTBL.bottomAnchor, constant: 21.0) //= myBigSurTBL.bottomAnchor + 21
    button.centerXAnchor.constraint(equalTo: view.safeAreaLayoutGuide.centerXAnchor)      // = view.centerXAnchor

    view.trailingAnchor.constraint(equalTo: myBigSurTBL.trailingAnchor, constant: 16) //= myBigSurTBL.trailingAnchor + 16
    view.trailingAnchor.constraint(equalTo: label.trailingAnchor, constant: 153)// = label.trailingAnchor + 153

       // Remaining Configuration
    self.myBigSurTBL = myBigSurTBL
    button.addTarget(self, action: #selector(LeftViewController.energize(_:)), for: .touchUpInside)
    myBigSurTBL.dataSource = self
    myBigSurTBL.delegate = self

    self.view = view
}
```

The Right Place to make a TabBarItem in Code

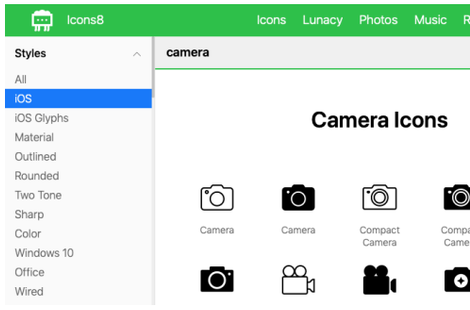
```
// We need this, because viewDidLoad() is *too* late in the
// process to set the tabBarItem properties

init(){
        super.init(nibName: nil, bundle: nil)
        tabBarItem.title = "Left"
    }

// Called when a ViewController is read in from a Storyboard
required init(coder aDecoder: NSCoder) {
  super.init(coder: aDecoder)
  self.tabBarItem.title = "Right"
  self.tabBarItem.image = UIImage(named:"snow.jpg")
}
```

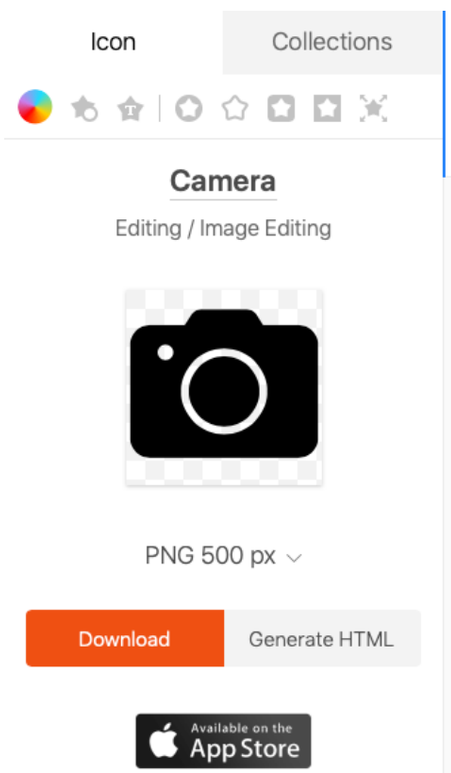
Tab Bar Item Images

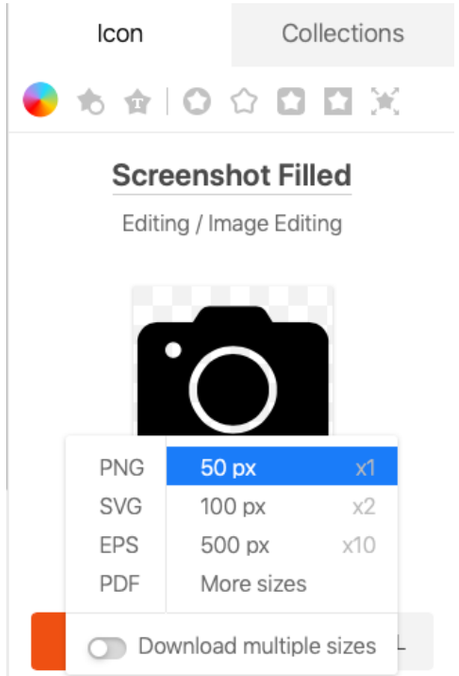
- Tab bar item images must be of a specific size, which depends on the size of your device (compact or regular), and whether it is circular or square. Try 36x36 px for an iPhone.
- Read Apple's Human Interface Guidelines for the full details
- You can make icons at glyphish.com or icons8.com
- To make your own from scratch, start with a suitably sized .png file with a transparent background
- Draw whatever you want as an image.

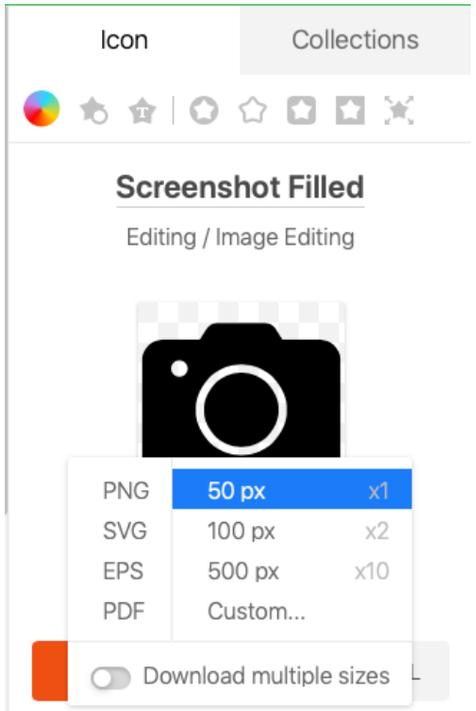
Making Icons

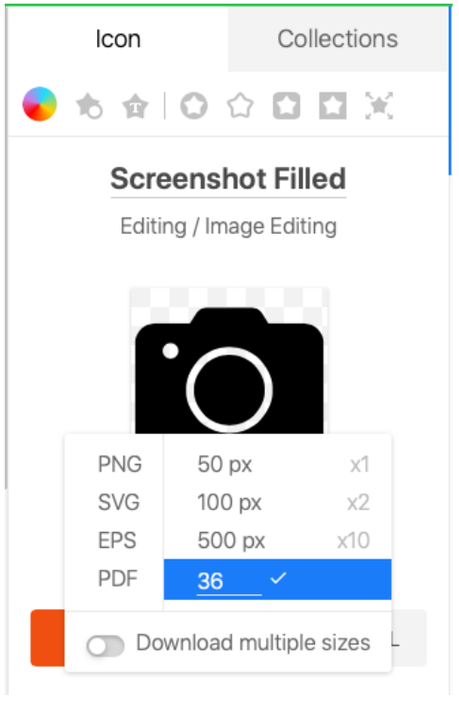
- Search for an icon

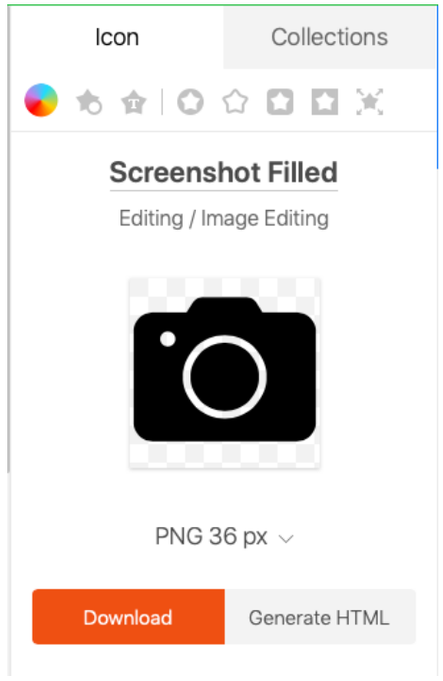






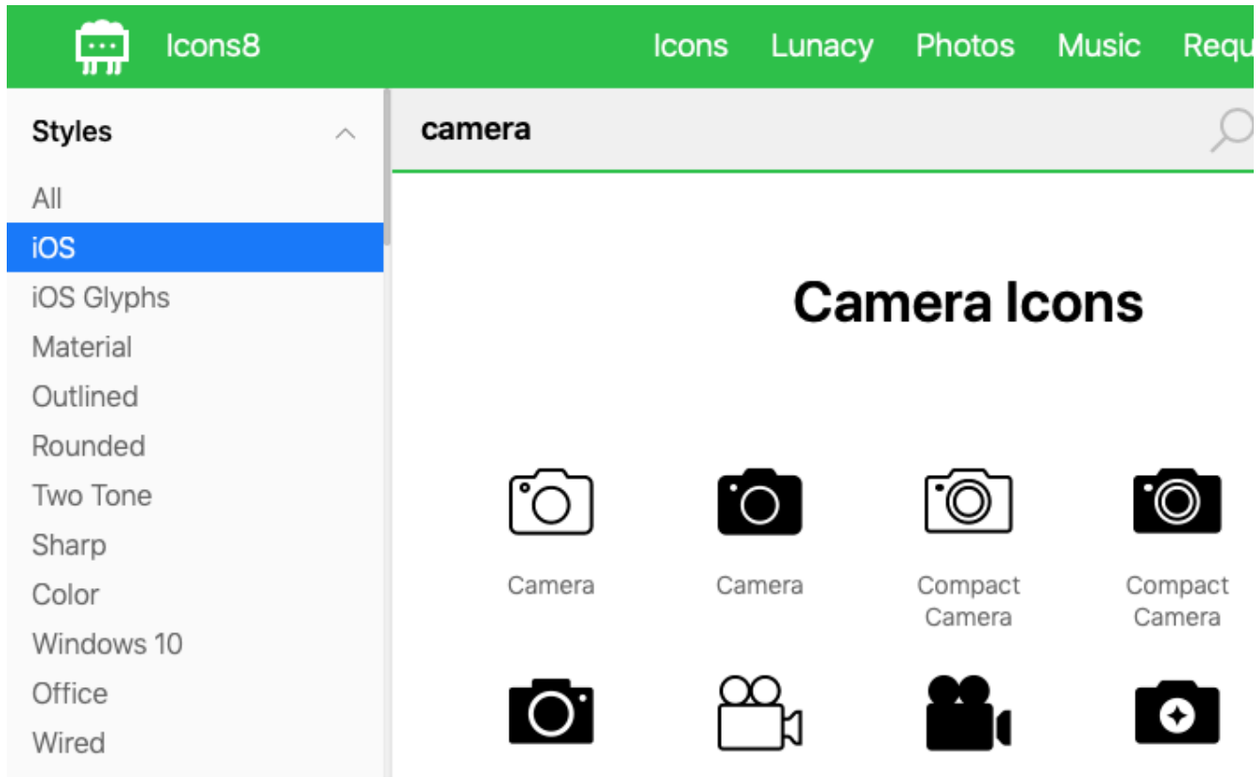






Using Icons 8

1.Search for an icon and click on it



Using Icons 8

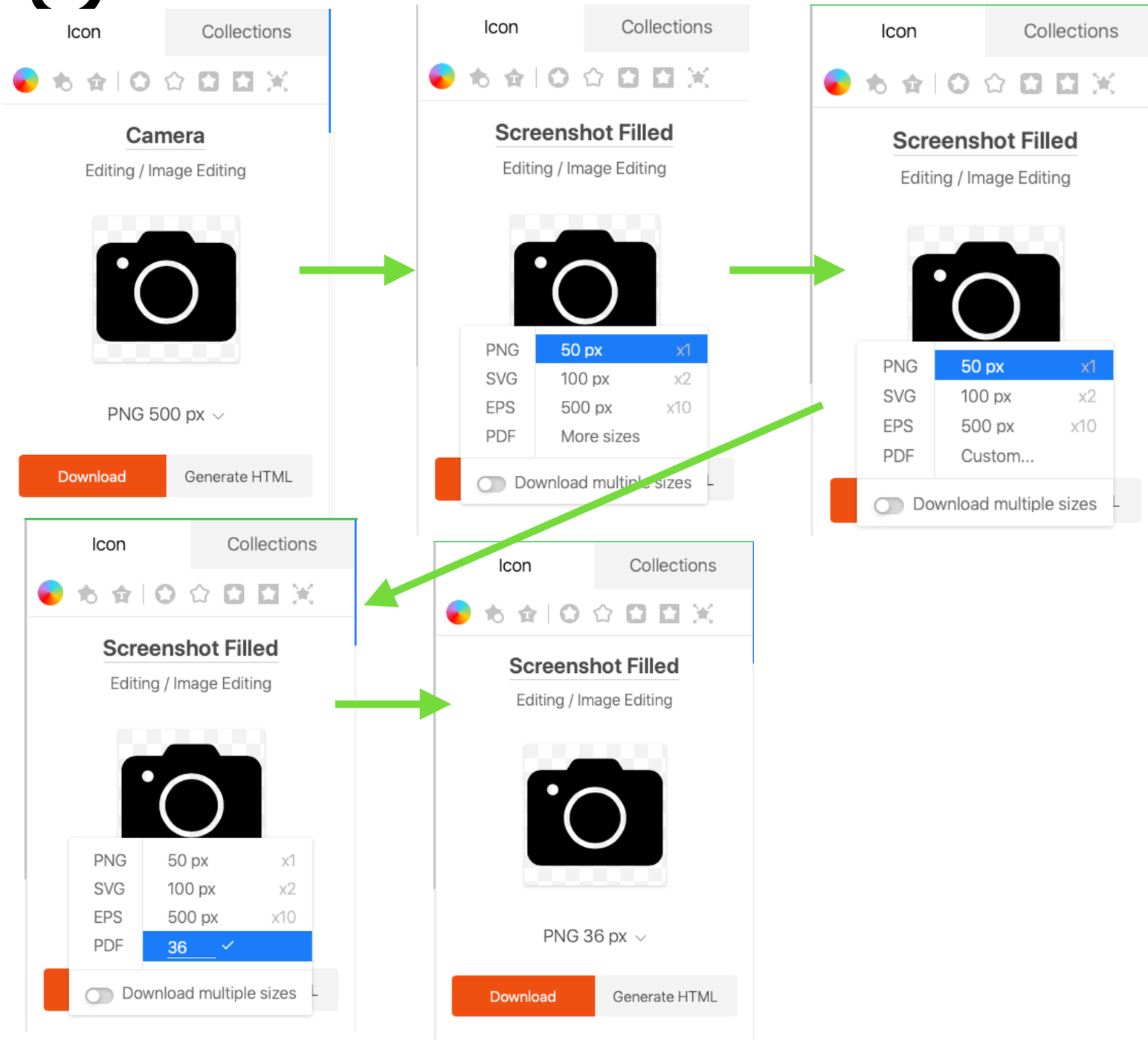
2.Click on the pull down

3.Click on More Sizes

4.Click on Custom...

5.Enter a Size

6.Click Download



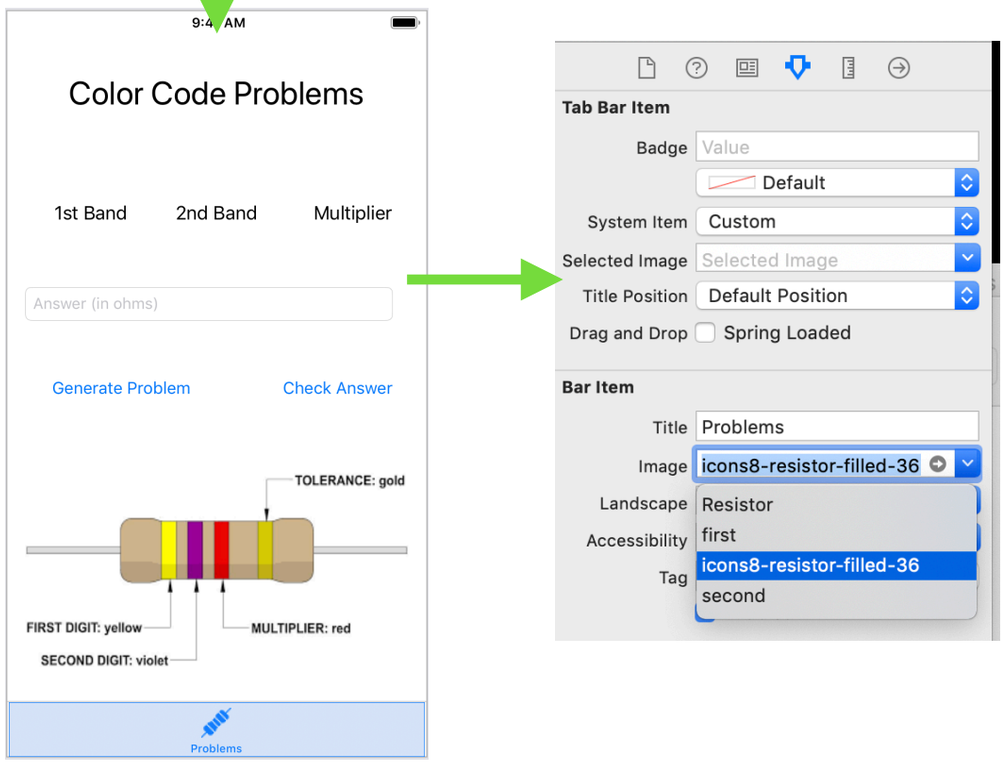
Using Icons 8

7. In Xcode, select Assets.xcassets

8. Drag the icon in

9. In SB, select the tabbar item

10. In its attribute inspector, choose the icon under Items



References

- https://developer.apple.com/ios/human-interface-guidelines/graphics/custom-icons/
- http://www.glyphish.com