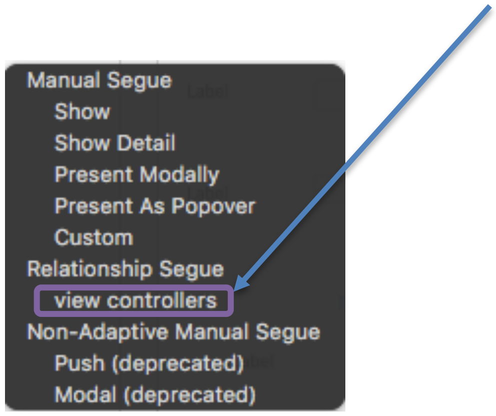
CIS 257 Programming Lab 7 – **Wellness app – part 2**

This lab will be a continuance of lab 6. Include any view controllers necessary to finish this 2-part series.

Step 1. Include a mapkit to show office locations of Dr. Feelgood.

Add a new **ViewController** to your exisiting storyboard from your Object library similarly how you did on the last lab.

In your storyboard, right click on your **Tab Bar Controller** and while keeping your mouse clicked, drag over your mouse into your new controller and at the pop up, under **Relationship Seque**, click on the **view controllers** choice.

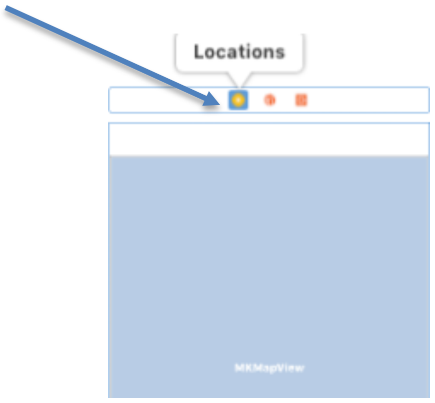


Next drag and drop a **Map Kit View** into a your new controller. Stretch out the map view to cover the majority of the viewing area.

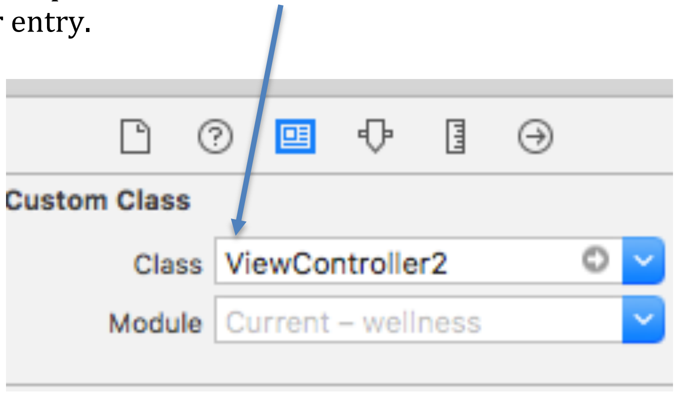
Title your new tab bar item as '**Locations**'. Add also an appropriate tab bar item image similarly how you did it last lab, that resembles a map.

Next add in a new ViewController class, aka "swift" file (File > New > File … Cocoa Touch Class (under iOS templates) and name it something appropriate. Keep all the default settings.

Next "glue" your new swift file to your new controller. From your storyboard, click once on your new controller at the top (as shown below), to grab *focus* of your controller.



Now go to your Identity Inspector in your utilities view and add in your new ViewController swift **<u>file name</u>** as a Custom Class (example shows ViewController2 as a class name. Yours of course may vary). Press Enter to commit your entry.



Open your *new* **ViewControllerxxx.swift** file and include the following import:

```
import MapKit
```

<u>Add</u> <u>in</u> an **<u>IBOutlet</u>** from your mapKit View onto your new ViewController swift file and call it **mapView** (*note*- Go to Editor > Assistant to always help split views in your work area).

Next include these delegates to your class heading:

```
MKMapViewDelegate, CLLocationManagerDelegate
```

Outcome should be something like the following opening:

```
class ViewController2: UIViewController, MKMapViewDelegate, CLLocationManagerDelegate {
```

Add the following lines at the <u>beginning</u> area <u>within</u> your class

```
//create a class to inherit from MKPointAnnotation class to hold points
class MyAnnotation: MKPointAnnotation { }
```

```
Note- Nothing else will be included in the class declaration above!
```

Include <u>also</u> these lines as part of your ViewController2 class members:

```
var locationManager = CLLocationManager()

var annotationArray = [MyAnnotation]()

var allObjectsTupleArray: [(objLat: CLLocationDegrees, objLong: CLLocationDegrees,
objName: String, objDesc: String)] =
    [(objLat: 41.878114, objLong: -87.629798, objName: "Nuveen Bld.", objDesc: "333 W
Wacker, Chicago"),
    (objLat: 41.888969, objLong: -87.633924, objName: "Merch. Mart Bld.", objDesc: "222
W Mechandise Mart Plaza, Chicago"),
    (objLat: 41.885295, objLong: -87.621490, objName: "Aon Building", objDesc: "200 E
Randolph St, Chicago")]
```

Next, include the following lines within your viewDidLoad to render your map with marker annotations! NOTE- THE ERROR ON THE CALL TO DISTANCE FUNC, WILL BE RESOLVED NEXT.

```
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationManager.requestAlwaysAuthorization()

        //set current head office location of Dr. Feelgood (state of IL building)
        let currentLatitude = 41.885598
        let currentLongitude = -87.625001

        //set map span
        let latDelta = 0.10
        let longDelta = 0.10

        let currentLocationSpan: MKCoordinateSpan =
            MKCoordinateSpan(latitudeDelta: latDelta,longitudeDelta: longDelta)
        let currentLocation: CLLocationCoordinate2D =
            CLLocationCoordinate2DMake(currentLatitude, currentLongitude)
        let currentRegion: MKCoordinateRegion =
            MKCoordinateRegion(center: currentLocation,span: currentLocationSpan)
        self.mapView.setRegion(currentRegion, animated: true)

        var indx = 0;

        for oneObject in allObjectsTupleArray {

         let oneAnnotation = MyAnnotation()
         let oneObjLoc: CLLocationCoordinate2D =
              CLLocationCoordinate2DMake(oneObject.objLat, oneObject.objLong)
         print("Latitude: \(oneObject.objLat)  Longitude: \(oneObject.objLong)")
         oneAnnotation.coordinate = oneObjLoc

         oneAnnotation.title = oneObject.objName
         print("ObjectName: \(oneObject.objName)")

         oneAnnotation.subtitle = oneObject.objDesc
         print("ObjectDescription: \(oneObject.objDesc)")

         if (indx < allObjectsTupleArray.count-1) {
         let vlat1:Double = allObjectsTupleArray[indx].0
         let vlong1:Double = allObjectsTupleArray[indx].1
         let vlat2:Double = allObjectsTupleArray[indx+1].0
         let vlong2:Double = allObjectsTupleArray[indx+1].1

         let d:Double = distance(lat1: vlat1, lon1:vlong1, lat2:vlat2, lon2:vlong2)

         print(d);
         }
         annotationArray.append(oneAnnotation);
         indx = indx + 1;
         print()
         }
        self.mapView.addAnnotations(annotationArray)
```

Within your class and outside any other function/method, add this haversine function which helps calculate distances between 2 points as well to the class:

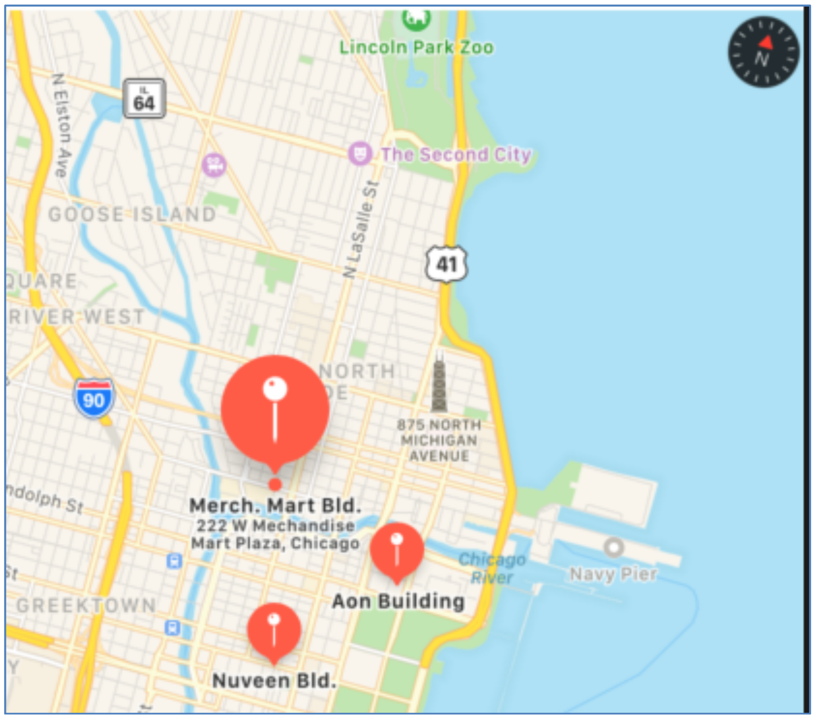
```
func distance(lat1: Double, lon1: Double, lat2: Double, lon2: Double) -> Double {

    let radlat1 = .pi * lat1/180
    let radlat2 = .pi * lat2/180
    let theta = lon1-lon2
    let radtheta = .pi * theta/180
    var dist = sin(radlat1) * sin(radlat2) +  cos(radlat1) *  cos(radlat2) *
        cos(radtheta);
    dist =  acos(dist)
    dist = dist * 180/Double.pi
    dist = dist * 60 * 1.1515
    return dist
}
```

Step 2. Run your app and test your map view.

If all is successful, you should see 3 pins or markers added in with their Name/address locations! Press and hold your option key to move pin objects closer or farther apart with your mouse so you can see addresses marked clearly on the map when you click on one of the annotation markers.

Snapshot of map. Merch. Mart marker chosen.



Step 3. Modify your map view logic.

Add in a few more locations to the array of tuples or the `allObjectsTupleArray` the following:

1. Add in Dr. Feelgood's home location so it shows as a pin. Include the necessary long and lat (located in the viewDidLoad code), address and a description for the State of Illinois building. You can find appropriate long/lat information located here - > http://www.latlong.net

2. Add in yet another downtown building office location *close* to the build locations supplied above along with an address and description.

Test your app again to see two more pins added in!

Step 4. Mod your coding one more time for the road!

Check the distance returned from the distance function and make sure no marker is shown when the distance (variable **d**) is greater than .7 miles from one location to the next. No need to check distance of each location with another location as that code has been already provided for you. :)

Test your app again to verify what pins should only be showing up given the above distance constraint.

Check your console display area as it shows data from the ***for each*** loop which may tell you quite a bit of intel as far as each marker by position, description, etc., and it shows the distance between each location! Cool!!!

Example of output in debug area

```
Latitude: 41.878114  Longitude: -87.629798
ObjectName: Nuveen Bld.
ObjectDescription: 333 W Wacker, Chicago
0.779423112682092

Latitude: 41.888969  Longitude: -87.633924
ObjectName: Merch. Mart Bld.
ObjectDescription: 222 W Mechandise Mart Plaza, Chicago
0.688073820264509

Latitude: 41.885295  Longitude: -87.62149
ObjectName: Aon Building
ObjectDescription: 200 E Randolph St, Chicago
```

Step 5. Make snapshots so far of things.

Snapshot into a Word doc, results for maps with all pins and then only pins satisfying the distance constraints (they're probably is at least one I see so far, you??). Also highlight your output in the debug area of all your location information and paste that into word as well.

Step 6. Add yet another view controller and view controller file for a Wellness video!

Add in yet another View Controller to your storyboard and append it to the Tab Bar Controller again. Add in a relationship from your TabController to your new ViewController. Adjust your tab bar item once again and label it 'Wellness Today'. Include an appropriate symbol reflecting some media pic.

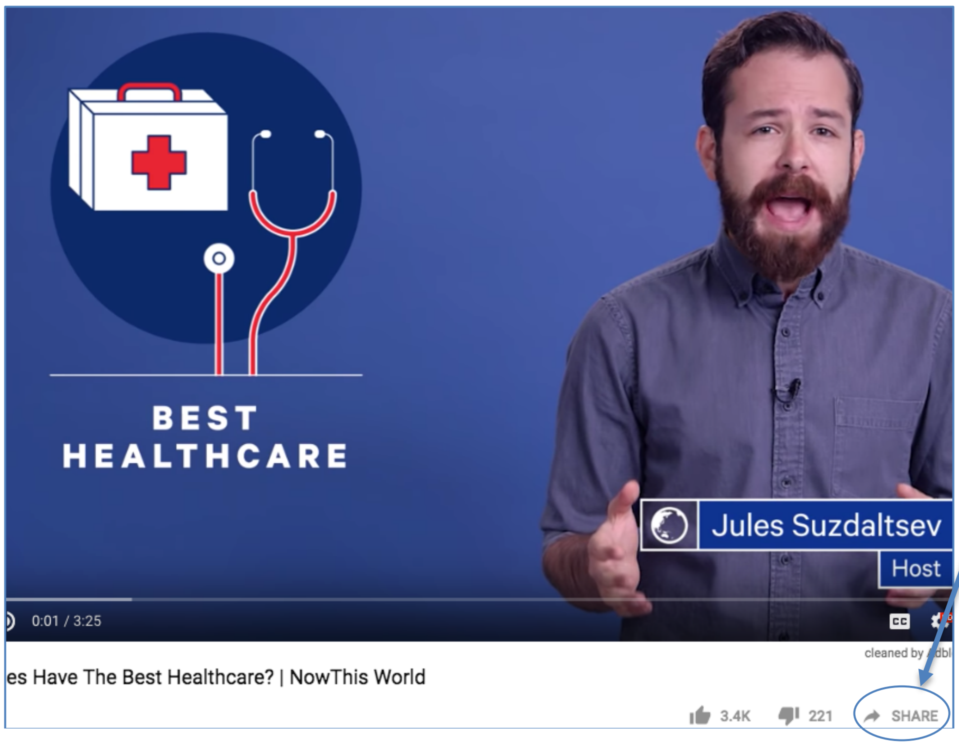
Drag in a **WebKitView** control to your new controller. Cover the web view of your entire controller. Include also a new **ViewControllerxxx.swift** file and again make sure to add in the **Custom Class** name of your new controller file for your View Controller control as was done in step 1 above.

Create an IBOutlet for the control for your new ViewController and call it **webView**.

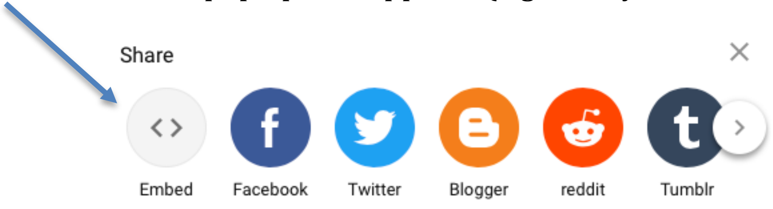
Add in the following import into your new ViewController file

```
import WebKit
```

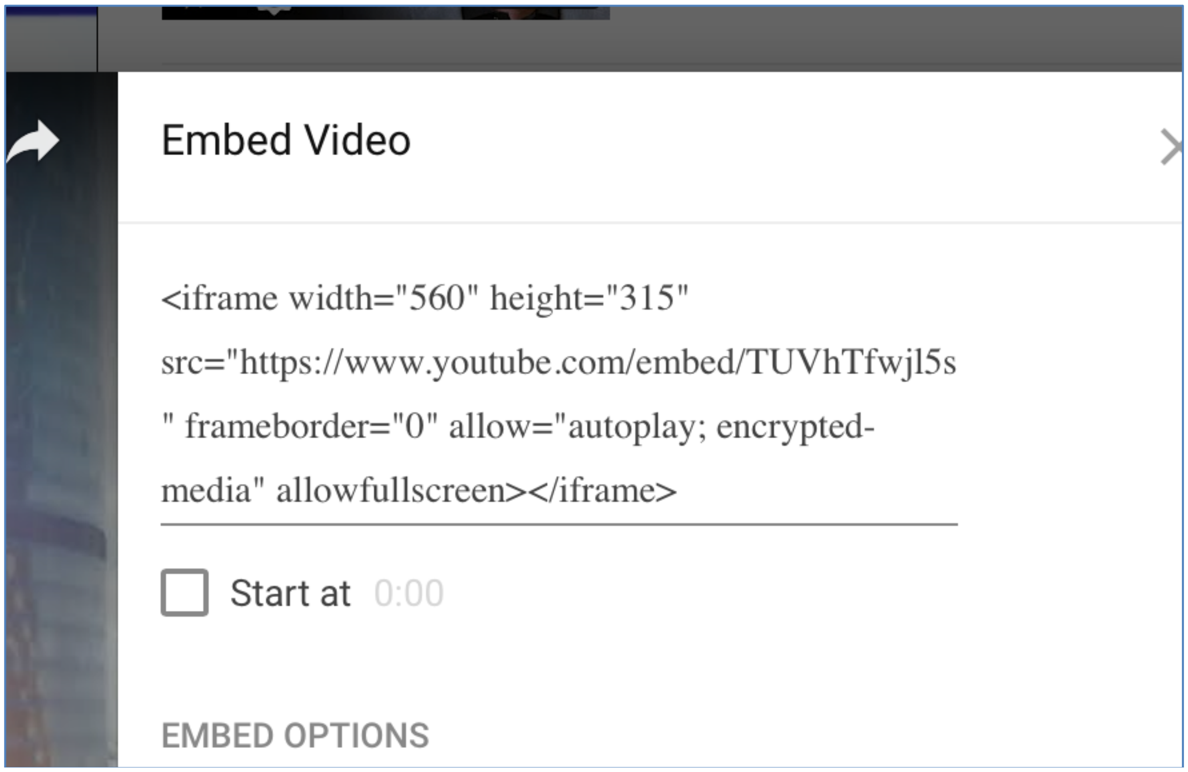
To add in a video link for your web view, go to youtube and follow the steps that are given next. Search for some awesome Health oriented video. Once your video appears, click on SHARE.



Next click the Embed link from the pop up that appears (sign into youtube if it doesn't open).



Highlight the iframe code that now appears and copy the full iframe link that appears. Example iframe code follows…



```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/TUVhTfwjl5s
" frameborder="0" allow="autoplay; encrypted-
media" allowfullscreen></iframe>
```

On your viewDidLoad – add in your copied link and paste it into the **htmlUrl** link assignment as shown next (I will leave the assignments below blank for you to add your own information).

```
let htmlUrl = "  "

webView.loadHTMLString(htmlUrl, baseURL: nil)
```

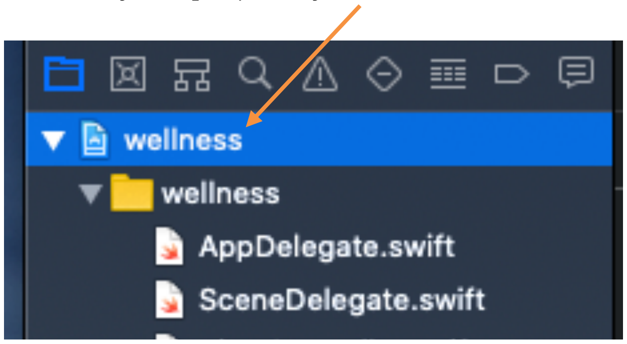
Just fyi, it may be a good idea to "escape" any double quotes from *within* your **embed tag,** changing each set to single quotes –(keep surrounding double quotes though for your entire string):

ex.

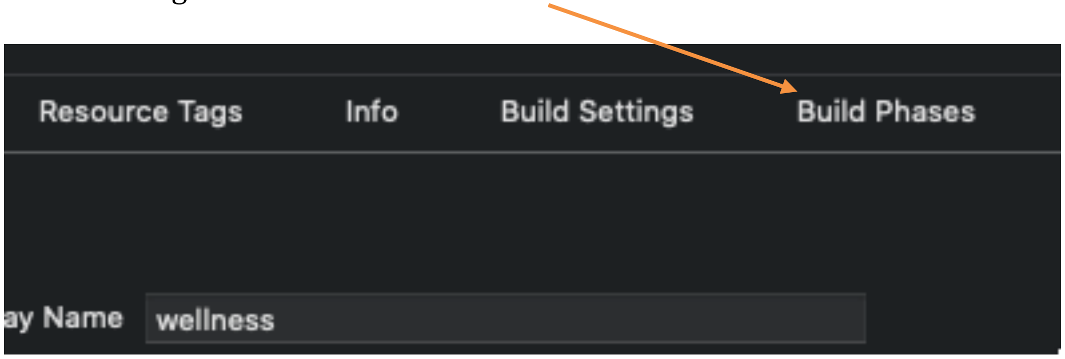
```
"<iframe width='560' height='315' src='https://www.youtube.com/embed/TeLc'...></iframe>"
```

Oh, one more final and important thing to add. Since webkit is a framework we have to include it in our build phase. To do so follow the steps as follows.

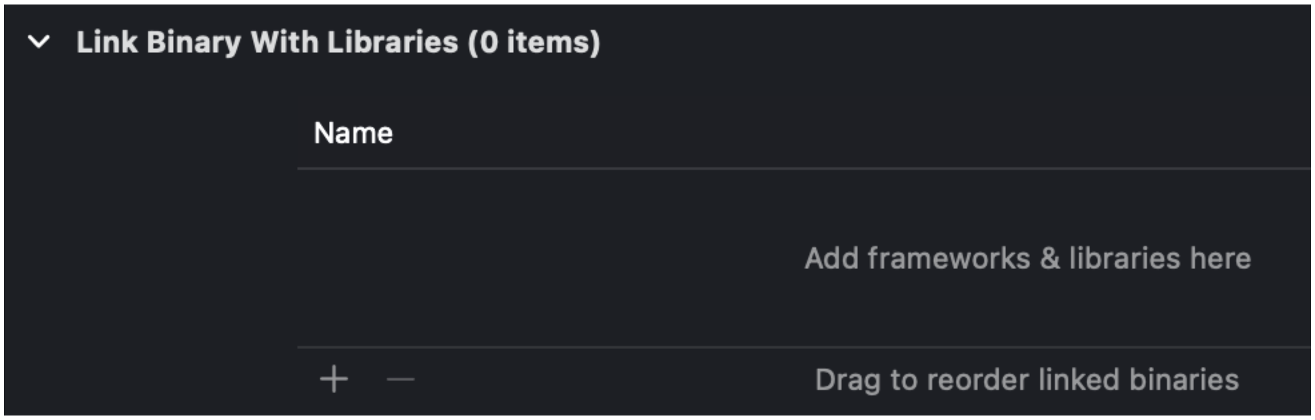
1. In the project navigator, select your project by name.



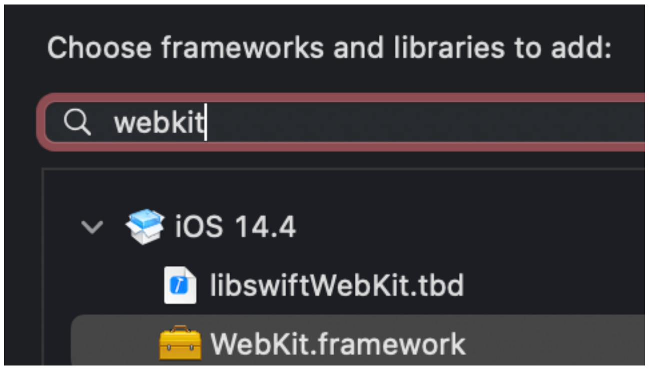
2. Next, select the target and choose the Build Phases tab.



Then much like lab 5, choose to add in a WebKit framework, by clicking on **Link Binary with Libraries** to open it up. Click the + sign to search for the WebKit.framework so you can add it to your project.


Link Binary With Libraries (0 items)
Name
Add frameworks & libraries here
+ —
Drag to reorder linked binaries

At the pop up, key in **webkit** in your search area and select the WebKit.framework as shoown below, then click the Add button to finish.



Take yet another trial run at this point.

And that's it!!! Test it and snapshot your video in Action for credit!

Step 7. Add yet another view controller and view controller file for a final run through.

Here your on your own. Create an added controller/file to be added to your head tab controller. Add in necessary features to the tab item to make it understandable to the user. Perform something original in addition to all the items you have now added. Ideas could be a simulated contact form, RSS news feed using a TableView, a Contact view etc...

Snapshot your working new view in action as well!

Step 8. Include the following for full credit

Include a snapshot of your initial view showing all your tab items. Show additional tab items in action you snapshot for each view controller you added in for this lab exercise. Include code for each view controller you worked with for this final lab part as well.