Overview

Create an App that will drop PushPins onto a map based on addresses that the user inputs.

- Part 1: Introduction to MKMapKit
- Part 2: Introduction to PushPins
- Part 3: Use Google's API to lookup an Address

Submission

Lab evaluation form
Lab feedback form (Online)

Note: If you are unable to complete the lab during this week's lab period, please be ready to demonstrate before the start beginning of the next lab.

Part 1. Introduction to MKMapView

You will create a simple application that displays Google maps and begin playing with the MapKit Framework.

Step 1) Create a new project called **FindMe** using the View-based application template.
Step 2) Add the **MapKit Framework** to your project.
Step 3) Import the MapKit/MapKit.h header in your View Controller header. `#import <MapKit/MapKit.h>`
Step 5) Add an instance of MKMapView to your view using **one of the following methods**:

To add a MKMapView using the Interface Builder:

1. Add an IBOutlet in your header file for an instance of the MKMapView class.
2. Open FindMeViewController.xib in Interface Builder and drag a MKMapView onto the view.
3. Connect a new referencing outlet from the new instance of the MKMapView to your IBOutlet that you created.

Or, to add a MKMapView programmatically:

1. In the viewDidLoad method, instantiate the instance variable and add it to the view:

```
- (void) viewDidLoad {
      [super viewDidLoad];
      MKMapView *mapView = [[MKMapView alloc] initWithFrame:self.view.bounds];
      [self.view insertSubview:mapView atIndex:0];
}
```

Note: When the mapView is added to the view hierarchy, there is no need to have a strong pointer to the object.

Step 6) Test your application in the simulator; you should see a map.
Step 7) Add the following to your viewDidLoad method (where mapView refers to your MKMapView instance):

```
mapView.showsUserLocation=TRUE;
mapView.mapType=MKMapTypeHybrid;
```

Step 8) You can reposition your map using the setRegion function of the MKMapView class.

```
MKCoordinateRegion ames;
ames.center.latitude = 42.02832;
ames.center.longitude = -93.651;
ames.span.latitudeDelta = 0.005;
ames.span.longitudeDelta = 0.005;

[mapView setRegion:ames animated:YES];
```

Step 9) Run your project in the simulator; you should see a map of Ames.
Step 10) (DEMO) The final part of this lab can be used to demo all parts of the project at once, or you may demo each part individually to incrementally earn the demo points.

Part 2: Introduction to PushPins

Reusing your FindMe project from part 1, you will now explore the MKAnnotation protocol, MKPinAnnotationView class and MKMapViewDelegate. You will be able to add Push Pin images to your map at certain latitudes and longitudes. If you want to use your own custom images instead of pushpins, then read the documentation on MKAnnotationView; the implementation is very similar.

Create a PushPin Class

Step 1) Create a **new class** called **PushPin** that subclasses **NSObject**.
Step 2) Import the MKAnnotation header. **#import <MapKit/MKAnnotation.h>**
Step 3) Make your PushPin class implement the MKAnnotation protocol. **<MKAnnotation>**
Hint: At a minimum you should implement the ***- (id)initWithCoordinate:(CLLocationCoordinate2D)coordinate*** *initializer and must have a property named* ***coordinate*** *of type CLLocationCoordinate2D. Right-click on MKAnnotation and Jump to the Definition to see other optional properties to implement.*

Step 4) Build your project to make sure it is free of errors.

Use the PushPin Class in your View Controller

Step 5) Import PushPin.h in FindMeViewController.h
Step 6) The following code is used to animate adding a push pin to an instance of MKMapView called mapView.

```
CLLocationCoordinate2D location;
location.latitude = 42.02832;
location.longitude = -93.65066;
PushPin *myPlace = [[PushPin alloc] initWithCoordinate:location];
[mapView addAnnotation:myPlace];
```

Add this code in an IBAction that is triggered when a button is pressed.

Step 7) Test your code in the simulator. A red push pin should appear on the map when you press the button.
Step 8) (DEMO) The final part of this lab can be used to demo all parts of the project at once, or you may demo each part individually to incrementally earn the demo points.

Change the Color of the PushPin: The MKMapViewDelegate Protocol

Step 9) You can change the color of a PushPin by extending the behavior of the MKMapView class through its delegate.
Step 10) Update FindMeViewController.h so it implements the MKMapViewDelegate protocol.
Step 11) Set the delegate of you MKMapView instance to the FindMeViewController. You can do this in the Interface Builder by drawing a new connection, or by setting `mapView.delegate = self;` in the viewDidLoad method of FindMeViewController.m.
Step 12) We must now implement the methods of MKMapViewDelegate. Delegate definitions can have a mix of optional and required methods, so it's good to read the documentation about the delegate interface you're implementing.

Documentation:
http://developer.apple.com/library/ios/#documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html

Step 13) Implement the **mapView:viewForAnnotation:** method of the protocol.

Note: You'll notice that this method is very similar to the implementation of tableView:cellForRowAtIndexPath: from the UITableViewDelegate used in the last lab.

```
- (MKAnnotationView *) mapView:(MKMapView *)localMapView viewForAnnotation:(id<MKAnnotation>)annotation {
    if ([annotation class] != [PushPin class]) return nil;

    MKPinAnnotationView *pin = (MKPinAnnotationView *) [localMapView dequeueReusableAnnotationViewWithIdentifier:@"i"];
    if (pin == nil) {
        pin = [[MKPinAnnotationView alloc] initWithAnnotation:annotation reuseIdentifier:@"i"];
    } else {
        pin.annotation = annotation;
    }
    pin.pinColor = MKPinAnnotationColorGreen;
    pin.animatesDrop = YES;
    return pin;
}
```

Step 14) (DEMO) The final part of this lab can be used to demo all parts of the project at once, or you may demo each part individually to incrementally earn the demo points.

Part 3: Use Google's API to lookup an Address

In this section, you'll create a UITextField that will allow the user to add a PushPin to the map by typing an address.

Step 1) Type the following URL into a web browser:

http://maps.google.com/maps/geo?q=ames,%20ia&output=csv

You'll notice that it returns four arguments in a comma separated list, the last two being the latitude and longitude for the address in the query.

Step 2) Reusing your project from the previous part, alter it so that it queries Google. You may find the following methods very useful:

Methods of NSString Class

```
+ (NSString *) stringWithFormat:(NSString *)format, …
- (NSString *) stringByAddingPercentEscapesUsingEncoding:(NSStringEncoding)enc    // use NSUTF8StringEncoding
+ (NSURL *) URLWithString:(NSString *)url;
+ (NSString *)stringWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)enc error:(NSError **) &err
- (NSArray *) componentsSeparatedByString:(NSString *)separator
```

Methods of NSURL Class

```
+ (NSURL *) URLWithString:(NSString *)url;
```

Methods of NSArray Class

```
- (id) objectAtIndex:(NSInteger)index
```

Methods of NSNumber

```
- (double) doubleValue
```

Using these methods, you can:

1. Build an NSString of the URL you want to submit to Google.
2. Create a NSURL from that string.
3. Get the results using contentsOfURL.
4. Parse the results into an array.
5. Pull out the 3rd and 4th objects in the array.

Step 3) Demo your project to the TA by typing an address into the UITextField and having a PushPin (not a red PushPin) drop on the map at that given location.