“Nearby” — A Location-based App

Nearby is a location-based app that searches nearby point of interest locations and shows routes from the user’s current location to the selected POI location.

To start with, first download the starter project.

Step 1: Present a Search Bar

We first add a search bar on top of the Map view to let the user enter the search keywords.

```
class NearbyViewController: UIViewController, UISearchBarDelegate {
  ...
   @IBAction func showSearchBar() {
         let searchController = UISearchController(searchResultsController: nil)
         searchController.searchBar.delegate = self
         present(searchController, animated: true, completion: nil)
   }

   // MARK: - UISearchBarDelegate

   func searchBarSearchButtonClicked(_ searchBar: UISearchBar){

      searchBar.resignFirstResponder()
      dismiss(animated: true, completion: nil)

      if let searchText = searchBar.text {
         print("search text: " + searchText)
      }
   }
}
```

Step 2: Initiate a Local Search with the Keywords

We then initiate a *MKLocalSearch* request using the keywords entered in the search bar by the user.

```
func searchBarSearchButtonClicked(_ searchBar: UISearchBar){

      searchBar.resignFirstResponder()
      dismiss(animated: true, completion: nil)

      if let searchText = searchBar.text {
         let searchRequest = MKLocalSearchRequest()
         searchRequest.naturalLanguageQuery = searchText
         searchRequest.region = mapView.region

         let search = MKLocalSearch(request: searchRequest)
         search.start { response, error in
            guard let response = response, error == nil else {
               print("There was an error searching for: \(searchRequest.naturalLanguageQuery) error: \
(error)")
               return
            }

            for mapItem in response.mapItems {
               print("item: " + mapItem.name!)
            }
```

```
        }
    }
}
```

Step 3: Annotate the Searched Locations on the Map

First, define a new class conforming to the MKAnnotation protocol.

```
import Foundation
import MapKit

class POIAnnotation: NSObject, MKAnnotation {

    var placemark: MKPlacemark?

    var coordinate: CLLocationCoordinate2D
    var title: String?
    var subtitle: String?

    init(coordinate: CLLocationCoordinate2D,
         title: String?,
         subtitle: String?,
         placemark: MKPlacemark?) {

        self.coordinate = coordinate
        self.title = title
        self.subtitle = subtitle
        self.placemark = placemark
    }
}
```

Secondly, in NearbyViewController.swift, make the following changes: 1) add a new instance variable to record all annotations on the map.

```
var annotations = [MKAnnotation]()
```

2) Modify the searchBarSearchButtonClicked method to create a map annotation for each returned search item.

```
func searchBarSearchButtonClicked(_ searchBar: UISearchBar){

        searchBar.resignFirstResponder()
        dismiss(animated: true, completion: nil)

        self.mapView.removeAnnotations(annotations)
        annotations.removeAll()

        if let searchText = searchBar.text {
            let searchRequest = MKLocalSearchRequest()
            searchRequest.naturalLanguageQuery = searchText
            searchRequest.region = mapView.region

            let search = MKLocalSearch(request: searchRequest)
            search.start { response, error in
                guard let response = response, error == nil else {
                    print("There was an error searching for: \(searchRequest.naturalLanguageQuery) error: \
(error)")
                    return
                }

                for mapItem in response.mapItems {
```

```
                if let coordinate = mapItem.placemark.location?.coordinate {
                    let annotation = POIAnnotation(coordinate: coordinate, title: mapItem.name, subtitle:
mapItem.placemark.thoroughfare, placemark: mapItem.placemark)

                    self.annotations.append(annotation)
                }
            }
            self.mapView.addAnnotations(self.annotations)
        }
    }
}
```

3) Make the class *NearbyViewController* conform to the MKMapViewDelegate protocol

```
class NearbyViewController: UIViewController, UISearchBarDelegate, MKMapViewDelegate
```

4) Provide a custom pin view for every newly added annotation.

```
func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {
        if annotation.isKind(of: MKUserLocation.self) {
            return nil
        }

        var pinView = mapView.dequeueReusableAnnotationView(withIdentifier: "NearbyPin") as?
MKPinAnnotationView
        if pinView == nil {
            pinView = MKPinAnnotationView(annotation: annotation, reuseIdentifier: "NearbyPin")
        }

        pinView!.canShowCallout = true
        pinView!.pinTintColor = UIColor.green
        pinView!.rightCalloutAccessoryView = UIButton(type: .detailDisclosure)
        pinView!.animatesDrop = true

        pinView!.annotation = annotation

        return pinView
    }
```

Step 4: Request Routing Info

1) Add the following three properties for routing

```
    var destPlacemark: MKPlacemark?
    var sourcePlacemark: MKPlacemark?
    var currentRoute: MKRoute?
```

2) When the user taps the *Routes* button, initiate a routing request from the current location to the user selected point of interest.

```
@IBAction func showDirection() {
        guard let destPlacemark = destPlacemark else {
            return
        }

        let directionRequest = MKDirectionsRequest()
        // Set the source and destination of the route
        directionRequest.source = getDirectionSource()
        directionRequest.destination = MKMapItem(placemark: destPlacemark)
        directionRequest.transportType = MKDirectionsTransportType.automobile
        // Calculate the direction
        let directions = MKDirections(request: directionRequest)
        directions.calculate { response, error in
            guard let response = response else {
                if let error = error {
                    print("Error: \(error)")
                }
                return
            }
            self.currentRoute = response.routes[0]

            if let steps = self.currentRoute?.steps {
                for step in steps {
                    print("\(step.instructions)")
                }
            }
        }
    }
```

3) Define a fake current location for testing purpose

```
// MARK: - Helper Methods

    func getDirectionSource() -> MKMapItem {
        if let sourcePlacemark = sourcePlacemark {
            return MKMapItem(placemark: sourcePlacemark)
        } else {
            return MKMapItem.forCurrentLocation()
        }
    }

    func setCurLocation() {
        let geoCoder = CLGeocoder()
        geoCoder.geocodeAddressString("95 1st Ave New York, NY 10003", completionHandler:
{ placemarks, error in
            if error != nil {
                print("\(error!)")
                return
            }

            if let placemarks = placemarks {
                // Get the first placemark
                let placemark = placemarks[0]
                self.sourcePlacemark = MKPlacemark(placemark: placemark)

                // Add annotation
                let annotation = MKPointAnnotation()
                annotation.title = "UpState"
                annotation.subtitle = "Fusion"

                if let location = placemark.location {
                    annotation.coordinate = location.coordinate

                    // Display the annotation
                    self.mapView.showAnnotations([annotation], animated: true)

                    let region = MKCoordinateRegionMakeWithDistance(location.coordinate, 1000, 1000)
                    self.mapView.setRegion(self.mapView.regionThatFits(region), animated: true)
                }
            }
        })
    }
```

Step 5: Overlaying the Route

After receiving the routing information, we draw the routes as an overlay on top of the map view.

1) In `@IBAction func showDirection()`, replace

```
          if let steps = self.currentRoute?.steps {
              for step in steps {
                  print("\(step.instructions)")
              }
          }
```

with

```
          if let route = self.currentRoute {
              self.mapView.removeOverlays(self.mapView.overlays)
              self.mapView.add(route.polyline, level: MKOverlayLevel.aboveRoads)

              let rect = route.polyline.boundingMapRect
              self.mapView.setRegion(MKCoordinateRegionForMapRect(rect), animated: true)
          }
```

2) Draw the overlay in a MKMapViewDelegate protocol

```
func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) ->
        MKOverlayRenderer {
            let renderer = MKPolylineRenderer(overlay: overlay)
            renderer.strokeColor = UIColor.blue
            renderer.lineWidth = 3.0

            return renderer
    }
```

Step 6: Displaying the Routing Steps

In addition of showing the route using an overlay view, when the user taps the *info* button in the annotation callout, the app will also shown the routing steps in a pop-over table view.

1) Define *StepsTableViewController* as follows:

```
import UIKit
import MapKit

class StepsTableViewController: UITableViewController,UIPopoverPresentationControllerDelegate {

    var routeSteps = [MKRouteStep]()

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
    // MARK: - Table view data source

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete implementation, return the number of rows
        return routeSteps.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "StepCell", for: indexPath)

        // Configure the cell...
        cell.textLabel?.text = routeSteps[indexPath.row].instructions

        return cell
    }

    func presentationController(_ controller: UIPresentationController,
viewControllerForAdaptivePresentationStyle style: UIModalPresentationStyle) -> UIViewController? {

        let navigationController = UINavigationController(rootViewController:
controller.presentedViewController)
        let doneButton = UIBarButtonItem(barButtonSystemItem: .done, target: self, action: #selector(done))
        navigationController.topViewController?.navigationItem.rightBarButtonItem = doneButton
        return navigationController
    }

    func done(sender: Any?) {
        dismiss(animated: true, completion: nil)
    }
}
```

2) In *NearbyViewController.swift*, invoke the *ShowSteps* segue when the user taps the *info* button in the callout view

```
 func mapView(_ mapView: MKMapView, annotationView view: MKAnnotationView,
calloutAccessoryControlTapped control: UIControl) {

        performSegue(withIdentifier: "ShowSteps", sender: view)
    }
```

3) Pass the route steps to the *StepsTableViewController* class

Step 7: Getting the Current Location Using the Location Service

1) In the project's Info.plist, add a new key "Privacy - Location When In Use Usage Description", and set its value to "This app lets you keep track of interesting places. It needs access to the GPS coordinates for your location."

If this key is not specified, the app will crash with an error message "*This app has attempted to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSLocationWhenInUseUsageDescription key with a string value explaining to the user how the app uses this data*" when the user taps "*Get Location"* to get the current location.

2) Let the class CurLocViewController conform to CLLocationManagerDelegate protocol

```
class CurLocViewController: UIViewController, CLLocationManagerDelegate
```

3) Define a location manager instance variable

```
let locationManager = CLLocationManager()
```

4) Define two helper methods, one for starting the location service, and the other for stopping the location service

```
func startLocationService() {
        locationManager.delegate = self
        locationManager.distanceFilter = kCLDistanceFilterNone
        locationManager.desiredAccuracy = kCLLocationAccuracyThreeKilometers

        locationManager.startUpdatingLocation()
    }

    func stopLocationService() {
        locationManager.delegate = nil
        locationManager.stopUpdatingLocation()
    }
```

5) Implement the CLLocationManager Delegate methods

```
func locationManager(_ manager: CLLocationManager,
                     didFailWithError error: Error) {
        print("didFailWithError \(error)")

        if (error as NSError).code != CLError.locationUnknown.rawValue {
            stopLocationService()
        }
    }

    func locationManager(_ manager: CLLocationManager,
                     didUpdateLocations locations: [CLLocation]) {
        let newLocation = locations.last!

        if newLocation.timestamp.timeIntervalSinceNow < -5 {
            return
        }

        if newLocation.horizontalAccuracy < 0 {
            return
        }

        latitudeLabel.text =
            String(format: "%.8f", newLocation.coordinate.latitude)
        longitudeLabel.text =
            String(format: "%.8f", newLocation.coordinate.longitude)

        if newLocation.horizontalAccuracy <= locationManager.desiredAccuracy {
            print("We are Done!")

            stopLocationService()
        }
    }

    func locationManager(_ manager: CLLocationManager, didChangeAuthorization status:
CLAuthorizationStatus) {
        if status == .authorizedAlways || status == .authorizedWhenInUse {
            startLocationService()
        } else {
            stopLocationService()
        }
    }
```

6) Start the location service to get the user's current location when the user taps the *Get Location* button. Before the location service can be started, the app checks if the user has authorized or enabled the location service.

```
@IBAction func getCurLocation() {
        let status = CLLocationManager.authorizationStatus()
        if status == .restricted || status == .denied {
            print("locatio service is denied")
        } else {
            if status == .notDetermined {
                locationManager.delegate = self
                locationManager.requestWhenInUseAuthorization()
            } else {
                startLocationService()
            }
        }
    }
```

Step 8: Converting the Location Latitude/Longitude to Street Address

After retrieving the coordinate of the current location, we can request reverse-geocoding from Apple's server. In *LocViewController.swift,* make the following modifications:

1) Define an instance variable for geo-coder

```
let geocoder = CLGeocoder()
```

2) Define the method for requesting reverse geocoding and converting the geocoding result into a legible street address.

```
func findGeoInfo(_ location: CLLocation) {
        print("*** Going to geocode")

        geocoder.reverseGeocodeLocation(location, completionHandler: { [weak self]
            placemarks, error in
            if let error = error {
                print("fail with error: \(error)")
            } else if let placemark = placemarks?.last! {
                self?.addressLabel.text = self?.getAddress(from: placemark)
            } else {
                print("no address found")
            }
        })
    }

    func getAddress(from placemark: CLPlacemark?) -> String {
        var address = ""

        if let s = placemark?.subThoroughfare {
            address += s + " "
        }

        if let s = placemark?.thoroughfare {
            address += s }

        address += "\n"
        if let s = placemark?.locality {
            address += s + " "
        }
        if let s = placemark?.administrativeArea {
            address += s + " "
        }
        if let s = placemark?.postalCode {
            address += s
        }

        return address
    }
```

3) Invoke the geocoding method when the current location is found.

```
func locationManager(_ manager: CLLocationManager,
                         didUpdateLocations locations: [CLLocation]) {
        …
        if newLocation.horizontalAccuracy <= locationManager.desiredAccuracy {
            print("We are Done!")

            findGeoInfo(newLocation)

            stopLocationService()
        }
    }
```