Create App and Add Map

Getting Started

Open xcode

Select create a new project

Select Single View Application (Picture)

Name application (“MapApp”) – language = ‘Swift’, device family = ‘Universal’

Add the Map

Select Main.storyboard from the Project navigator menu on the left

In the object library in the bottom right corner of the screen search for ‘Map Kit View’

Drag the ‘Map Kit View’ object to the View Controller screen

Size and Constraints

Expand the Map View to fill the screen

From the menu on the bottom of the screen click the ‘Resolve Auto Layout Issues’ and select ‘Reset to Suggested Constraints’ to set constraints for the map view.

Code

In the top right corner click the ‘Show Assistant Editor’ button to open the ViewController file alongside the storyboard

In the ViewController.swift file, add ‘import MapKit’ under import UIKit

To create an outlet for the map in the code, click the map while holding control and drag to the code inside the ViewController class and above the functions

When the outlet window pops up set the Connection to ‘Outlet’ and the name to ‘mapView’

Run

Press the play button to build and run the app in the simulator

The app should display the map

Expanding the Code

The next step is to get the users location. To do this, first, you need to create an instance of the CLLocationManager class and store a reference to it.

Picture

Place this code inside the ViewController class.

Declare the class as implementing the MKMapViewDelegate protocol, which allows you to use certain set of map related methods, and the CLLocationManagerDelegate, which allows you to manage the delivery of location related events to the application.

Add ‘MKMapViewDelegate’ and ‘CLLocationManagerDelegate’ to the class declaration

Picture

We then need to indicate that the ViewController class is the delegate for both the mapView object and the locationManager. To do that add this code inside the viewDidLoad() function.

User Location Authorization

The next step is to request permission form the user to access information about their current location. Put the following line inside the viewDidLoad() function:

Additionally, to get the request to appear to the user you need to update the Info.plist file. Open the file. To add to the add to the apps ‘Information Property List’, hover over the ‘Information Property List’ and a + sign will appear. Click the + sign and a new line will be added to Information Property List. Set the name to ‘Privacy – Location When In Use Usage Description’, the Type to String, and the Value to the message you want to display to the user when the request is made.

Now run the app and the request to access the users location should pop up

Adding Current Location

Now that you have requested permission from the user to access their location you can display that location on the map.

Return to the Main.storyboard/ViewController.swift view to add to the code.

To display the current location add the following code inside the viewDidLoad() function under the delegate expressions:

Picture

Explanation of this code:

if CLLocationManager.locationServicesEnabled() – anything inside the this block will only be executed if the user allows the application to access their current location

mapView.showUserLocation = true – shows the users current location on the map view

Run the app and the users current location will be marked

MKMapItems

To navigate to a destination, you can ask the user to input an address, but for simplicity we will just create a variable and hard code an address to use for the destination

Next, create MKMapItem variables inside the ViewController class.

Picture

MKMapItems hold information about a specific point on a map. So, these variables will hold information about the user’s current location and the destination.

Set MKMapItem Variables

This line will set the currentMapItem variable to hold the information for the current location

Picture

Setting the destinationMapItem variable is more complicated. You need to use the CLGeocoder class to convert the destination address into coordinates that can be used to locate the destination on the map. You should put the CLGeocoder object inside the if statement that checks if the user granted access to the app to access the current location.

Picture

Explanation: The CLGeocoder takes the destination address a returns a CLPlacemark object. You can set the placemark variable to the first placemark returned in the placemarks list. This CLPlacemark object has a location property which in turn has a coordinate property. Additionally, the placemarker has and addressDictionary property. Use these properties to set the coordinates of the destinationMapItem.

MKDirectionsRequest

Now that both MKMapItems hold information about the two navigation points, we can now get directions to navigate between the two points.

Because the CLGeocoder runs asynchronously, the rest of the code to get directions needs to be placed inside the CLGeocoder completion block so that it does not run before the CLGeocoder finishes running.

Create an instance of the MKDirectionsRequest class. This instance can then be configured with the source and destination points using the MKMapItems, you can specify the transportation mode, and you can indicate that you do not want to receive alternative routes. Your code should look like the following:

Picture

Calculate Directions

You can now create an instance of the MKDirections class from the request. This object will give you route-based directions from the user’s current location to the destination address. The direction object passes the request to the Apple servers and returns the requested directions.

After the directions are requested you can use the calculate function to calculate the requested route information.

Displaying Directions to User

The calculate function also runs asynchronously, so the following code to display the directions must be placed inside the calculate return block.

Picture

Code Explanation:

if error != nil – if an error is returned, will print ERROR to the console

self.Route = response?.routes.first – sets the Route variable to the first route returned from the calculate function

self.mapView.add(self.Route!.polyline) – adds a lines to the map for the route

self.mapView.setVissibleMapRect(…) – bounds the size of the map on loading to fit the directions

Before the route can be displayed on the map the following function needs to be added to the ViewController class to render the polyline

The app can be run now and the directions will be displayed

Turn by turn directions