TP 5 - MapKit

Adrien Humilière 30/03/2017

Part 1

- Using Interface Builder, add a `Button` to the view labeled *Open Maps App with URL*.
- Add constraints to center the button horizontally, and to set its top vertical space relative to the main view.
- Create a connection from the button to a controller action called `openMapsAppWithURL:`.

```
1 @IBAction func openMapsAppWithURL(sender: UIButton) {
2     if let url = URL(string: "http://maps.apple.com/?q=↩
    Yosemite") {
3         UIApplication.shared.open(url, options: [:], ↩
    completionHandler: nil)
4     }
5 }
```

- Explain the creation of an `URL` from a string, representing a URL or "the location of a resource", a map that one wishes to open, which iOS handles by opening the Maps app.
- Explain how the `q=` query string parameter represents a "query", and is parsed by the Maps app to search for a location on the map.
- Using the documentation, explore the `UIApplication` class reference and the `openURL:` method.
- Explain how the call to `UIApplication.shared` returns a reference to the app instance itself.
- Run the app, tap the button, and observe how the Maps app enters the foreground.

Part 2

- Search for Map Kit Framework in the documentation and explore the resulting documentation.
- Map Kit provides an easy way to embed maps in your own application.
- Using Interface Builder, delete the button from the interface.

- Add a Map View to interface. Adjust its top edge to leave room for the iOS status bar, and place the constraints to completely fill the screen, for all device sizes.
- In the `ViewController` class, comment the `openMapsAppWithURL:` method.
- Run the app and observe the crash: *Could not instantiate class named MKMapView.* We need to link the Map Kit framework in the project.
- Using the Project Navigator, select the project and add **MapKit.framework** to the project *Linked Frameworks and Libraries*, and observe the framework appear in the Project Navigator.
- Run the app and observe the map appear. Tap and drag the map to scroll, and alt-click with the mouse to zoom in and out of the map.

Part 3

- Using Interface builder, select the Map View and open the Attributes Inspector.
- Change the **Type** to **Hybrid** and ensure that **Shows User Location** is checked.
- Run the app and observe how the map adds satellite imagery to the application.
- Observe the console warning *Trying to start MapKit location updates... must call requestWhenInUseAuthorization ... first.* iOS apps must request user authorization to use location information with the Core Location framework.
- Import the Core Location framework above the `AppDelegate` class definition.
- Explore Core Location Framework documentation.
- In the `AppDelegate` class, declare a `CLLocationManager` property with a default value.

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

- Explore the `CLLocationManager` class documentation.
- In the `AppDelegate` class, modify `application:didFinishLaunchingWithOptions:` to request permission to use iOS location services.
- `application:didFinishLaunchingWithOptions:` will prompt the user for permission to use location services, but will require additional app configuration. Using the Project Navigator, open `Info.plist` and add the Key `NSLocationWhenInUseUsageDescription` and Value *Required for displaying your location on the map.*
- Run the app and tap the Allow button. Observe the position of the location beacon, scrolling and zooming the map by alt-clicking with the mouse if necessary.
- The iOS Simulator chooses Cupertino, CA as the default location.

- Using the Simulator menu item *Debug* > *Location* > *Custom Location...*, enter a latitude and longitude, and observe the change in the position of the location beacon.

Part 4

- By default, a map view does not automatically zoom the map to the current location.
- Explore the `MKMapView` class documentation, the `setUserTrackingMode:animated:` method, and the `MKUserTrackingMode` enumeration.
- Import Map Kit in `ViewController`.
- Using Interface Builder, drag a connection from the map view to create an outlet in the `ViewController` class.
- Add a call to `setUserTrackingMode:animated:` in `viewDidLoad`.
- Run the app, and observe the map "zoomed in" on the location.
- We might wish to zoom the map at a scale that is different from the default zoom level provided by the map view.
- Explore the `MKMapViewDelegate` protocol documentation, and the `mapView:didUpdateUserLocation:` method.
- Using Interface Builder, control-drag a connection from the Map View to `ViewController` in the Document Outline, specifying `ViewController` as the Map View delegate.
- Declare the adoption of the `MKMapViewDelegate` protocol for the `ViewController` class.
- Add an implementation of `mapView:didUpdateUserLocation:` to `ViewController`.

```
1 func mapView(mapView: MKMapView, didUpdateUserLocation ↩
    userLocation: MKUserLocation) {
2     let center = CLLocationCoordinate2D(latitude: ↩
    userLocation.coordinate.latitude, longitude: userLocation.↩
    coordinate.longitude)
3     let width = 1000.0 // meters
4     let height = 1000.0
5     let region = MKCoordinateRegionMakeWithDistance(center, ↩
    width, height)
6     mapView.setRegion(region, animated: true)
7 }
```

- The map view calls the `mapView:didUpdateUserLocation:` method in its delegate when the user location is updated.

- Explore the domentation for `CLLocationCoordinate2D` structure, `MKCoordinateRegion` structure, and the `MKCoordinateRegionMakeWithDistance` function.
- `MKCoordinateRegion` consists of a center, width and height; and the `MKMapView setRegion:animated:` method receives a "square region" for configuring the map display.
- Run the app and observe the map "zoomed in" on the location.

Part 5

- Using Interface Builder, add a Toolbar to the bottom of the view. Add autolayout constraints for the toolbar leading, trailing, and bottom space. Adjust the bottom edge of the map view to match the top edge of the toolbar.
- Click on the default Item button and use the Attributes Inspector to change the **Identifier** attribute to **Add**. Observe how the button appearance changes.
- Run the app.
- Using Interface Builder, control-drag from the Add button to the `ViewController` class definition to create an action called `dropPin:`.
- Map Kit uses the concept of an `MKAnnotation` to represent markers on a map. Explore the documentation for `MKAnnotation` protocol reference. There is no `MKAnnotation` class, only a protocol. We will need to create a class for an annotation, adopting the `MKAnnotation` protocol.
- Add a new Pin class to the project.
- Import `MapKit` and update the class definition to extend `Object` and adopt the `MKAnnotation` protocol.
- Using the documentation, review the requirements of the `MKAnnotation` protocol. It requires having a `CLLocationCoordinate2D` property.
- Add a `CLLocationCoordinate2D` property and an initializer to the `Pin` class.
- Update the implementation of the dropPin: method in the controller class: create a new `Pin`, located at the center of the map.
- Run the app, observe the current location beacon, tap the Add button, and observe the annotation appear.

Part 6

- Helped by the documentation, use an image instead of the default pin view for annotations.
- Display the coordinates in a tooltip when tapping the pin.