Chapter 23: Use Location Data

Lecture 23
Jacob Bakarich

Chapter Goals

- Handle GPS errors
- Improve GPS results
- Reverse geocoding
- Testing on device
- Limiting Support Of Devices
- Supporting different screen sizes

Handle GPS Errors

- Getting GPS is error prone
- Modern devices use a mixture of GPS, cell tower triangulation, and wifi router information to attempt to pinpoint location data
- Due to the instability of these methods, any application that uses GPS needs to be able to handle a variety of errors.

Types Of Location Errors

- CLError.locationUnknown
  - Location is Unknown
  - Core Location will keep trying
- CLError.denied
  - User denied location access
- CLError.network
  - There was a network related error
- More errors related to compass and geocoding, but theses are the basics
- Can also be globally disabled in settings
  - Checked with !CLLocationManager.locationServicesEnabled()

Improving GPS Results

- Core location is constantly sending new location, regardless if the GPS Coordinates have changed.
- Would be useful if we needed to track the users location as they moved, however in this app, this is unnecessary.
- Costs a lot of battery.
- Need to stop scanning once we're certain that we have an accurate location.
    - As the GPS zeroes in on our location, the coordinates will change as we get closer and closer to our actual location.

Update UI

- Now that we're ensuring that we're getting an accurate location, we need to update the UI to reflect this fact.
- Currently, user can start saving their location immediately, despite the first locations not being very accurate.
- We will change what the "get my location" button will say to accurately reflect state when it is searching, and change it back to "get my location" when it has found the current location.
- Animated spinners are pretty and are universally recognized to mean that the app is doing something. By implementing one we can indicate to the user the app is still getting location.

Reverse Geocoding

- We are currently dealing with raw GPS coordinates, like (37.3324090, -122.0351218)
- Not human readable, ugly, and cumbersome to display
- Need to turn our GPS coordinates into an address
- Known as “Reverse Geocoding”
- Handled by the CLGeocoder object
- Uses a geocoding API hosted by apple to determine the actual address
- Need to be careful, exceeding rate or number of geocoding requests could cost money or will block the API requests from being returned.
- Happened to me using google maps geocoding, accidentally used up all of my daily allotted requests at 9am in the morning of a workday.

Closures

- A closure is a self contained block of functionality.
- Stores references, variables in the context which they are defined.
- Closures can be of three different types:
  - Global functions are closures that have a name and do not capture values
  - Nested Functions that have a name, can capture value from enclosing function
  - Unnamed expressions that can capture values from the surrounding context.
- Syntax:

```
{ ( parameters ) -> return type in
    statements
}
```

Closures

- Escaping Closures
  - A closure is said to “escape” a function when the closure is passed as an argument to the function, but is called after the function returns.
  - Denoted by “@escaping”

```
func getState(completion: @escaping (JSON) -> ()) {

    Alamofire.request("http://localhost:5000/state")
        .validate()
        .responseJSON { response in
            do {
            let jsonData = try JSON(data: response.data!)
                completion(jsonData)
            } catch{
```

Closures

- Escaping Closures
  - A closure is said to “escape” a function when the closure is passed as an argument to the function, but is called after the function returns.
  - Denoted by “@escaping”

```
func getState(completion: @escaping (JSON) -> ()) {

    Alamofire.request("http://localhost:5000/state")
        .validate()
        .responseJSON { response in
            do {
            let jsonData = try JSON(data: response.data!)
                completion(jsonData)
            } catch{
```

```
            getState() {response in
                self.setState(state: response)
            }
```

Handle Reverse Geocoding Errors

- We have implemented geocoding, but do not handle errors that could potentially be encountered during operation.
- Need to implement error checking much in the same way we handled errors encountered while searching for GPS

Importance Of Diverse And Comprehensive Testing

- The author notes that as an exercise, he tried this app on his iPod Touch, which does not actually have a GPS, so it must rely on wifi scanning to determine location
- The method of using WiFi scanning to determine a location is unlikely to reach the threshold of our desired accuracy of <10m
- This means that on a device without GPS capabilities, the app will be “Searching…” forever.
- We need to make sure our code terminates at some point if the location does not change much after a certain time, even if it does not reach the desired accuracy threshold.

Required Device Capabilities

- Since GPS is important to this location based app, it makes sense that we should only target iOS devices that possess a GPS sensor.
- This can be done by editing the info.plist file, where you can define device compatibility based on several different hardware/software aspects of iOS devices such as iOS version, CPU Generation, or the presence of a GPS.

Supporting Different Screen Sizes

- Our app is currently laid out for an iPhone SE sized screen, 4 inches.
- Testing our app on other screen sizes will reveal that our layout does not display correctly on other iOS devices.
    - “Get My Location” is missing on a 3.5 inch screen
    - Labels are not aligned on an iPhone 6/7/8+
- iOS has two options for supporting multiple size screens: Autolayout(newer) and Autoresizing
- Each view has a resizing setting that determines what happens to the size and positions of view elements.
- Basically defining where everything should be as a percentage of the screen.

Questions?