Spring 2017

# Gymfo: a Gym Finding iOS App using Google Maps API

Clay A. Wyers
*The University of Akron*, caw138@zips.uakron.edu

# Gymfo: a Gym Finding iOS App using Google Maps API

Clay A. Wyers

The University of Akron

Abstract – This project explores iOS mobile app development. I developed a gym finding app in Swift, using Google Maps APIs,and called it Gymfo. Other technologies were studied as well, including the cloud computing service Microsoft Azure. Through this project I learned the basics of XCode and Swift, Google APIs, and sending and receiving HTTP requests. The final product is an iOS app that allows users to search for nearby gyms and displays the results on an interactive map.

## I. Introduction

The Apple iPhone was first released in 2007, and revolutionized smart phones. Today, there are over 100 million iPhone users in the United States, and 52.3% of people who have a smartphone in the United States and Canada have an iPhone (Smith). This has created a huge market for iOS mobile applications, and knowledge of the subject could give a software developer more job opportunities.

On the App Store, one can find many different kinds of apps: games, utilities, news, and more. I decided that I was going to create an iOS app that allowed users to search for nearby gyms. This would expose me to Swift and Google APIs, and would also be a useful app. Throughout the project I also learned about other technologies I thought I might have needed, such as Microsoft Azure services, Node.js Servers, and the Google Maps Javascript API. To complete my implementation I also needed to learn and apply Swift techniques and classes like

querying with URLSession, asynchronous programming with semaphores, and JSON parsing.

## II. Required Technologies/Languages

### A. Swift

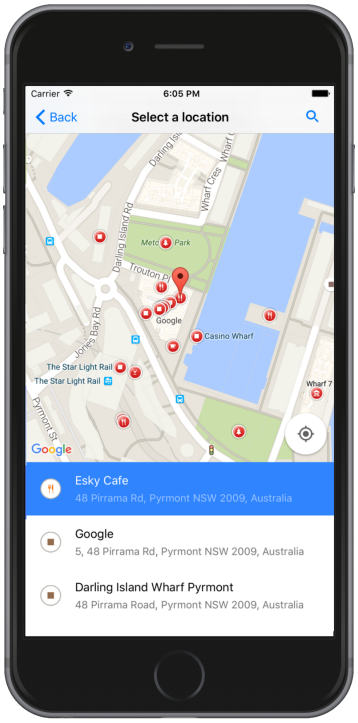
Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. for iOS, macOS, watchOS, tvOS, and Linux (Inc). An evolution of Objective-C, Swift was first released in 2015, and advertises itself as "fast, modern, safe, and interactive." If I had to compare it to another language I've been exposed to, syntax-wise it is similar to Ruby, requiring only very simple statements to accomplish tasks. No more semicolons at the end of a statement, and many of the API libraries have had their names simplified as much as possible. Even the traditional for loops have been removed and replaced.

I thought it might be difficult to code in a completely new language, using an IDE I've never seen before, but I was incorrect. XCode is has some very useful features, the most useful being its ability to check for errors as you code in real time and offer suggestions. I would estimate that around 30% of all my lines of code contained syntax errors, usually because I was calling APIs by their old names, or other simple errors like that. XCode makes correcting these errors very fast and easy, which saved me a lot of time and resources.

### B. Google API

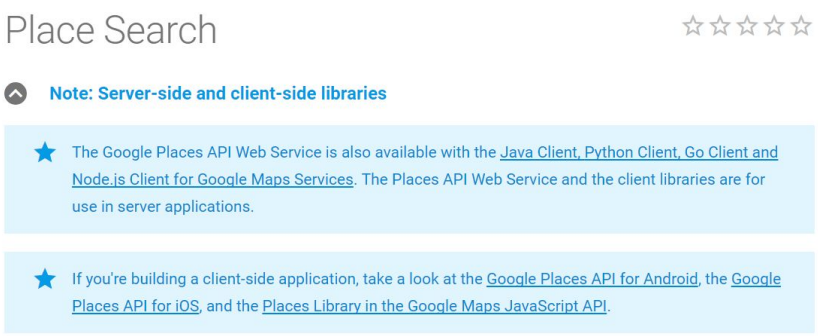
Google has a massive library of APIs available to developers to use in their applications. The Google Maps APIs give developers access to Google Maps Servers and use them to build their own map applications. Setting up this functionality is surprisingly simple, simply log into the API dashboard and request an API key. They offer a guide on getting started, as well as an

extensive reference guide. The APIs are offered as SaaS (Software as a Service), and it's completely free until an API key gets over 1000 requests a day.




For my project I originally thought I only needed to use the Places API for iOS, as it contains the GMSPlacePicker class, which provides an integrated map and UI to display nearby places to the user. The only issue with the class is that it offers no functionality for filtering the results of the nearby places it detects. My app only needed to display gyms, so this wasn't going to work. I tried several other Maps APIs in order to work towards a solution, and in the end I found a solution by using both the Maps API for iOS and the Web Service Places API.

III. Technologies I Thought I Needed, But Didn't




Once I realized that the Web Service Places API would meet the requirements of my project, I studied the documentation. It states that the Web Service API is designed for server-side applications, not client-side. It also suggested that it would work well as a proxy server for mobile applications to connect to Google's Servers through. Reading this, and not knowing anything about HTTP requests, I

assumed I needed to make these requests from a server, then send the results to the iOS device.

A. Node.js/Java Server

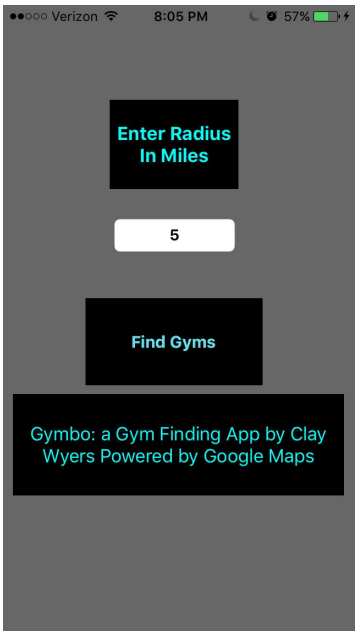
At first, I decided to implement a Node.js server, as it was another new technology I was interested in. Node.js is built on javascript, is asynchronous, event driven, and good for JSON API based applications. This seemed perfect for a server that would just wait for queries from an iOS device.

To host my server, I turned to Microsoft Azure. Azure is a cloud service that offers PaaS (Platforms as a Service) to clients. I set up an account and found a Node.js package that set up the server with minimal effort. After messing around with the server for a day or two, I was having difficulties understanding how all of the components fit together, and how I would change it to fit the scope of my project. During this time, I implemented a basic version of the app in HTML, using the Javascript api, as I figured it would be a good primer for writing the Node.js version. It did help me plan out my app in the end, but did not help much with the server. I tried a Java server also hosted by Azure, and understood that more due to previous experience. As I thought through the process I realized that I would be sending HTTP Requests to my Java server, which would send an HTTP request to Google's servers. This felt redundant, and if I can send an HTTP request to my server, why couldn't I send one straight to Google? As it turns out, I could, by using the Web Service API along with the URLSession library in Swift.


IV. Walking Through the Implementation

The actual implementation can be broken down into around 5 steps.

A. User Sets Radius



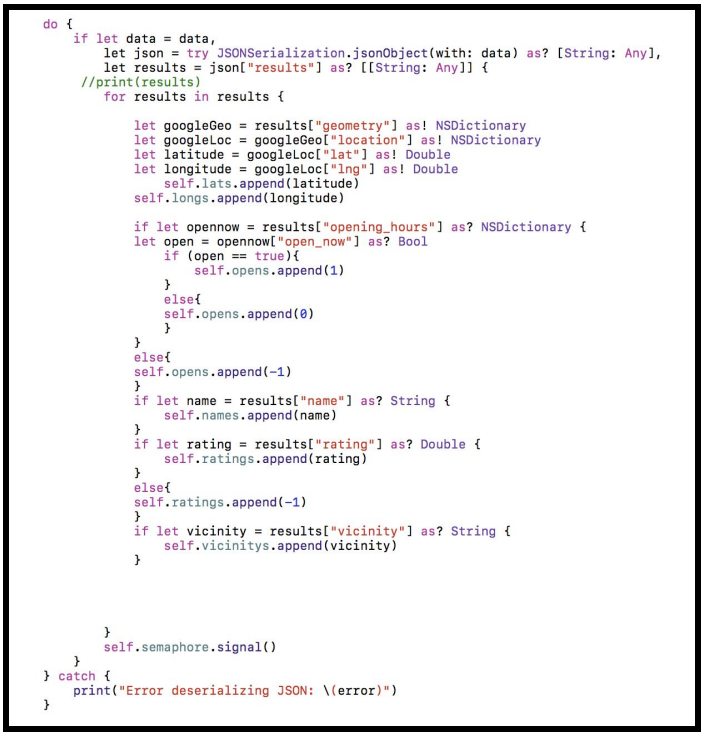

When the user opens the app for the first time, they are asked if it is okay for the app to track their location, as the phone will not provide it to the application unless it is authorized, for privacy reasons. The user is then is presented with the screen to the right, and prompted to enter a search radius. In version 1.0, the user must enter their desired search distance in miles, and it is converted to meters for the HTTP request. The maximum radius is 50,000 meters, according to Google's documentation. Once the user has set a distance, they tap on the Find Gyms button in order to start the search.

B. HTTP Request

To get the user's location, the app uses CLLocationManager, the central class for delivering location-based events to your app in Swift ("CLLocationManager - Core Location | Apple Developer Documentation"). The coordinates are converted to Doubles, and along with the search radius added to a String containing the URL needed for the query. URLSession is then used to send the HTTP request to Google's servers, which returns a response in JSON (JavaScript Object Notation).

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {

    let userLocation = locations.last!
    self.long = userLocation.coordinate.longitude
    self.lat = userLocation.coordinate.latitude

    let newCamera = GMSCameraPosition.camera(withLatitude: CLLocationDegrees(self.lat), longitude: CLLocationDegrees(self.long), zoom: 12.0)
    self.mapView?.camera = newCamera
    self.locationManager.stopUpdatingLocation()


    let url3 = URL(string: "https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=\(self.lat),\(self.long)&radius=\(self.rad)&type=gym&
    let session3 = URLSession.shared
    if let usableUrl3 = url3 {
        let task3 = session3.dataTask(with: usableUrl3) { (data, response, error) in
```

C. Parse JSON Response



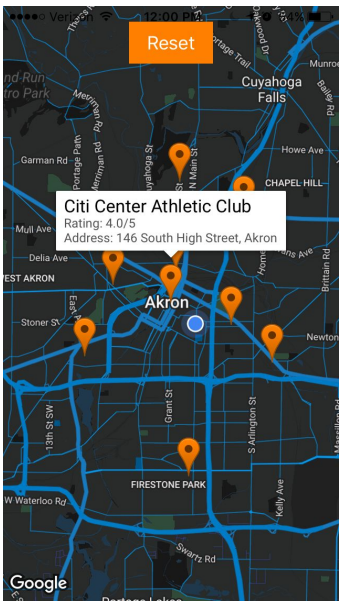
```
do {
    if let data = data,
        let json = try JSONSerialization.jsonObject(with: data) as? [String: Any],
        let results = json["results"] as? [[String: Any]] {
     //print(results)
        for results in results {

            let googleGeo = results["geometry"] as! NSDictionary
            let googleLoc = googleGeo["location"] as! NSDictionary
            let latitude = googleLoc["lat"] as! Double
            let longitude = googleLoc["lng"] as! Double
                self.lats.append(latitude)
            self.longs.append(longitude)

            if let opennow = results["opening_hours"] as? NSDictionary {
            let open = opennow["open_now"] as? Bool
                if (open == true){
                    self.opens.append(1)
                }
                else{
                self.opens.append(0)
                }
            }
            else{
            self.opens.append(-1)
            }
            if let name = results["name"] as? String {
                self.names.append(name)
            }
            if let rating = results["rating"] as? Double {
                self.ratings.append(rating)
            }
            else{
            self.ratings.append(-1)
            }
            if let vicinity = results["vicinity"] as? String {
                self.vicinitys.append(vicinity)
            }



        }
        self.semaphore.signal()
    }
} catch {
    print("Error deserializing JSON: \(error)")
}
```

The JSON contains useful information about nearby places such as names, coordinates, formal address, customer ratings, and whether or not the gym is currently open. Parsing this data and storing it into arrays is not very complicated, but if an entry doesn't have certain values such as a rating, then I add a dummy value to the array. This dummy value is usually something like -1, so that all the arrays stay the same size, and that I don't run into issues when I'm creating markers later. -1 is a good choice because it is a value that will not be present in the JSON query.

One very interesting issue that came up while developing forced me to learn about asynchronous programming in Swift. By default URLSession creates a new thread in order to send and receive the query. I process the data as soon as I get it, and because it was an asynchronous thread, the main thread of the application would load the map and create all the markers for local gyms, only the arrays would still be empty at that point, so no markers would be displayed. To get around this I used semaphores and dispatch_queue(). Semaphores avoids race conditions in my program restricting access of my marker arrays to one thread at a time, the thread filling the arrays with the parsed data. Dispatch_queue() was used to make sure that the markers were created after the arrays were filled and the map was created, assuring that they

appeared on the map. While it may seem inefficient to force my main thread to wait for another asynchronous thread to continue, the whole process was completed only a few seconds after hitting the Find Gyms button, so the user is not waiting for long, and there are no potential deadlocking issues, since only 1 thread modifies the arrays.

D. Create Map and Markers




Once the data about nearby places has been received and processed into arrays, the app creates a map and displays it on the screen using the Google Maps API for iOS. it then adds markers by iterating through all the arrays together, creating markers and plotting them on the map in their respective locations, using info from the coordinates arrays. Other information is added to the info window so that the user can tap on a marker to access it. At the top of this screen is a reset button, that takes the user back to the first view, so that they can change the search radius. GMSMarker is a class that allows users to create any kind of

```
let camera = GMSCameraPosition.camera(withLatitude: CLLocationDegrees(lat), longitude: CLLocationDegrees(long), zoom: 15.0)
mapView = GMSMapView.map(withFrame: CGRect.zero, camera: camera)
mapView?.isMyLocationEnabled = true
view = mapView
let button = UIButton(frame: CGRect(x: 5,y:  15,width:  80,height:  40))
button.backgroundColor = .orange
button.setTitle("Reset", for: .normal)
button.addTarget(self, action: #selector(buttonAction), for: .touchUpInside)

self.view.addSubview(button)
self.view.removeConstraints(self.view.constraints)
button.center.x = self.view.center.x
DispatchQueue.main.async
    {
        print(self.names)
        print(self.lats)
        print(self.longs)
        print(self.ratings)
        print(self.rad)
        var i = 0
        for names in self.names{
            let position = CLLocationCoordinate2D(latitude: self.lats[i], longitude: self.longs[i])
            let marker = GMSMarker(position: position)
            marker.icon = GMSMarker.markerImage(with: .orange)
            marker.title = names
            var snippet:String = ""
            if (self.ratings[i] != -1){
            snippet += "Rating: \(self.ratings[i])/5"
            }
            if (self.vicinitys[i] != nil){
               snippet +=  "\nAddress: \(self.vicinitys[i])"
            }
            if(self.opens[i] == 1){
                snippet +=   "\nOpen Now: Yes"
            }
            else if(self.opens[i] == 0){
                snippet += "\nOpen Now: No"
            }

            marker.snippet = snippet
            marker.map = self.mapView
            i = i+1
            print("Success! marker \(i)")
        }
```

marker they want on a Google API Map. I iterate through the arrays to create the markers, create the marker at the parsed coordinates, and fill it with relevant information.

## V. Recommendations

The app has its core functionality, but there are a few improvements that could be made in order to improve the user experience:

### A. Set the Map Zoom Based on Search Radius

Currently, the map is always at a default zoom level. most of the time it is at a good distance, as the user can see all points, and they are mostly distributed across the screen. However, for very large and very small radii, it isn't very effective. Dynamically setting the map zoom level based on the provided radius would be a nice feature.

### B. List Search Results for an Alternative View

Just like the real Google Maps, it would be convenient for users to see all search results in a scrollable list if they tap a button. This makes it easy to see more data at one time, and provides an alternative to tapping on each marker to get that location's name and other info.

## VI. Conclusion

Overall, this has been a very educational project. When I proposed this project, I knew nothing about Swift or Google APIs. Throughout the course of the semester I've learned about them through reading guides and reference materials, and I've also learned about other technologies such as Microsoft Azure Cloud Computing  and Node.js servers. Swift is a really cool language to develop in, and working with Google APIs was interesting as well.

# References

"CLLocationManager - Core Location | Apple Developer Documentation." N.p., n.d. Web. 12 Apr. 2017.

Inc, Apple. "Swift - Apple Developer." N.p., n.d. Web. 12 Apr. 2017.

Smith, Craig. "50 Amazing iPhone Statistics." *DMR*. N.p., 25 Sept. 2015. Web. 12 Apr. 2017.