Lecture 10

Data Persistence, MapKit

Today

- Swift Lesson: Generics
- Data Persistence
- Firebase
- MapKit

Semester Plan

- 1 more app (assigned tonight)
- 1 more tutorial
- Final Project
  - Starting with initial brainstorming this week

Final Project Schedule

- April 9th — Final project kickoff / workshop
    - *Try to come to class with a few different ideas on April 9th. The more you come with, the more helpful I can be! We will discuss people's project proposals, feasibility, and recommendations for technologies etc.*
- April 16th — **Halfway Point**
    - *By this point you should have a clear runway ahead.*
- April 25th — **Demo Days**

Generics

The problem

```
func swapTwoStrings(_ a: String, _ b: String) {
    let temporaryA = a
    a = b
    b = temporaryA
}

func swapTwoDoubles(_ a: Double, _ b: Double) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

The solution: Generics!

```
func swapTwoValues<T>(_ a: T, _ b: T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

The solution: Generics!

```
func swapTwoDoubles(_ a: Double, _ b: Double) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

```
func swapTwoValues<T>(_ a: T, _ b: T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

Example: Stack

```
struct IntStack {
    private var items = [Int]()
    mutating func push(_ item: Int) {
        items.append(item)
    }
    mutating func pop() -> Int {
        return items.removeLast()
    }
}
```

Non-generic

```
struct Stack<Element> {
    private var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
}
```

Generic

Stack

```
struct Stack<Element> {
    private var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
}
```

Type Constraints

```
func someFunction<T: SomeClass, U: SomeProtocol>(someT: T, someU: U) {
      // function body goes here
}
```

More info

https://docs.swift.org/swift-book/LanguageGuide/Generics.html

Data Persistence

Data Persistence: lots of options!

- Built-in system frameworks
  - UserDefaults, FileSystem, CoreData, Keychain
- Third party db frameworks
  - Firebase
  - Realm
  - .... lots of others!

UserDefaults

PROS

- Extremely easy to use
- No setup required
- Fast and always available

CONS

- VERY LOW storage size
- Intended to store.... User Defaults!
  - Like simple Bool settings, or the id of the tab they last visited
- Has to be loaded in memory before your app opens

FileSystem

PROS

- Built in, not too difficult to use
- Very dependable
- Somewhat secure

CONS

- On-device only!
- Intended to store whatever you need. File size is less restricted here
- However, iOS is still weird about file stuff (mark your files as the right type)

CoreData

PROS

- Built in
- Very dependable
- Somewhat secure
- Uses Apple’s servers and the User’s iCloud storage tier

CONS

- Can be difficult to use, scale, and set up

Realm

PROS

- Easy to use, massive functionality
- Great for reactive programming
- Setup isn’t too difficult

CONS

- Free tier, paid after that
- Third party (but a really GOOD third party)

Firebase

PROS

- Massive functionality
- Firebase does a ton of extra stuff, including Analytics and Deployment
    - Firebase aquired Crashlytics, so this is also the best crash reporting in the biz

CONS

- Free tier, paid after that
- Third party (but its Google)
- Pretty heavy solution
- Requires auth
- Setup is not the easiest (not bad..)
- Not very *“swifty”*

Secure Enclave (Keychain)

PROS

- Super, super secure
- Synced with iCloud

CONS

- Only used for keys, id, passwords, credit cards, etc
    - So, limited use case (can’t put a photo in keychain)

UserDefaults Snippet

```
// Implementations available for String, Int, Date, etc.
// To save Codable structs, convert them to JSON first
// UserDefaults is just a persistent Dictionary, loaded on app
launch

// Only save SMALL amounts of data (< 40kb)
func save(_ userEmail: String) {
    UserDefaults.standard.set(userEmail, forKey: "user-email-key")
}

func loadUserEmail() -> String? {
    return UserDefaults.standard.string(forKey: "user-email-key")
}
```

Other Guides

- FileSystem
- CoreData
- Firebase
- Realm
- Keychain

Firebase

What is Firebase?

Firebase is Google's mobile platform to help you quickly develop high-quality apps.

What is Firebase really?

- A grotesque (but useful!) collection of tools:
  - Cloud Functions — serverless backend code
  - Web hosting
  - Authentication (through google, facebook, phone numbers, 2FA, etc!)
  - Cloud Storage
  - Realtime DBs

What is Firebase really?

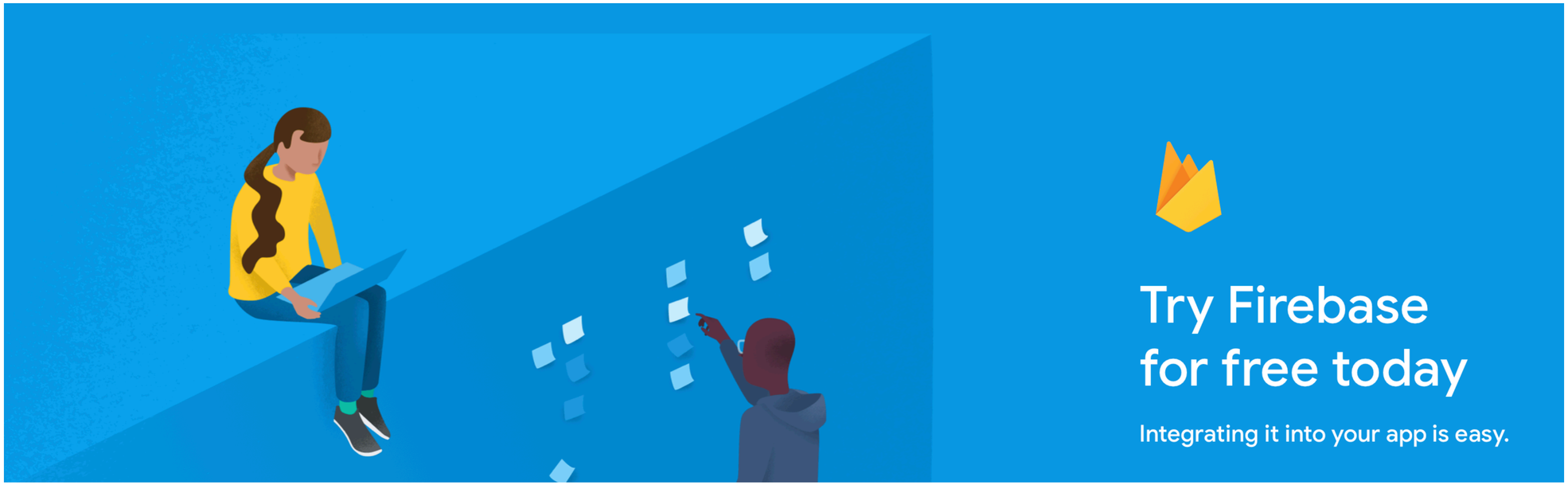
- A grotesque (but useful!) collection of tools:
  - Crashlytics :(
  - Performance monitoring
  - Cloud testing
  - App distro
    - Beta / pre-release workflows

What is Firebase really?

- A grotesque (but useful!) collection of tools:
  - In-app messaging (behavior-based triggers)
  - User demographic predictions
  - AB Testing
  - Remote config (ew)

What is Firebase really?

- Something you should take advantage of when making a project, if you don't mind the sharp edges
- Something you should avoid getting tied to, if you're starting a company



In this class

- We'll use the Realtime Database to build a small demo app
  - This is a NoSQL database with built-in data synchronization (more on that next slide..)
  - Everything is always up to date on both the client and server, as long as we have internet! No fetching required

What's a NoSQL database?

- This means instead of building our own APIs, endpoints, and servers held up by databases (web development).... We just use Firebase!

- We can think of Firebase's **Realtime Database** as a giant JSON file (or "Dictionary")

  - Each value has a key. Each value can contain a type like String, or a series of nested types (in either a dictionary or an array)

  - **Big feature 1:** we can ***set those key/values directly from code***, and they get automatically pushed to the server! No URLSession required.

  - **Big feature 2:** we can setup our code so that ***when those values change on the server, a closure is run in our code***

Getting started

- The docs for Firebase are huuuuuuuge
  - Don’t get lost!
  - We care about **iOS** & **Realtime Database** ~~**Cloud Firestore**~~
- **Cocoapods** is required to install Firebase. It’ll be on Swift Package Manager eventually — but Google moves slowly and Firebase is a complicated project to migrate big.

cocoapods.org

https://guides.cocoapods.org/using/getting-started.html

CoreLocation

What is CoreLocation?

Core Location provides services that determine a device's ***geographic location, altitude, and orientation****…*

The framework gathers data using ***all available components*** *on the device, including the Wi-Fi, GPS, Bluetooth, magnetometer, barometer, and cellular hardware.*

What is CoreLocation?

- We don’t have a lot of choice in **how** the location is determined; iOS itself figures out the best way to determine location.
- Sensors
  - GPS and Cellular
  - WIFI and Bluetooth
  - Magnetometer (Compass) and Barometer
  - iBeacons
  - iPhone 11 — UWB sensor

CoreLocation Types

- **CLLocation** - struct that contains information about a device's geographic location, altitude, orientation, heading, and speed. Also contains information about the **accuracy** of the measurement.

- **CLLocationManager** - used to start and stop the delivery of location-related events.

- **CLLocationManagerDelegate** - methods you use to receive events from an associated location manager object.

    - For example, we may receive the latest update of the users position from the CLLocationManagerDelegate; we use this to update our map.

Privacy

- **Location is among the most private of information.**
    - In general, users are not happy to learn of apps tracking their location when they aren’t using the app, unless for good reason.
- Just like the photo lectures — you must request authorization before receiving location events.
    - Users can deny your location request, or they can accept it.
        - They can accept it once, for while the app is being used, or for all of the time. **You as the developer have no control over this.**

Efficiency

- **Determining location is hard work that quickly uses battery life.**
  - For this reason, we want to receive location updates with the **least** accuracy we need for our use case. This allows the phone to save power.
- There are 3 “services” we use to receive location updates:
  - Visits location service (most efficient)
  - Significant change location service (efficient, big changes only)
  - Standard location service (accurate, uses lots more power)

CLLocationManager

```
let locationManager = CLLocationManager()
locationManager.delegate = self
locationManager.desiredAccuracy =
kCLLocationAccuracyNearestTenMeters

locationManager.startUpdatingLocation()
```

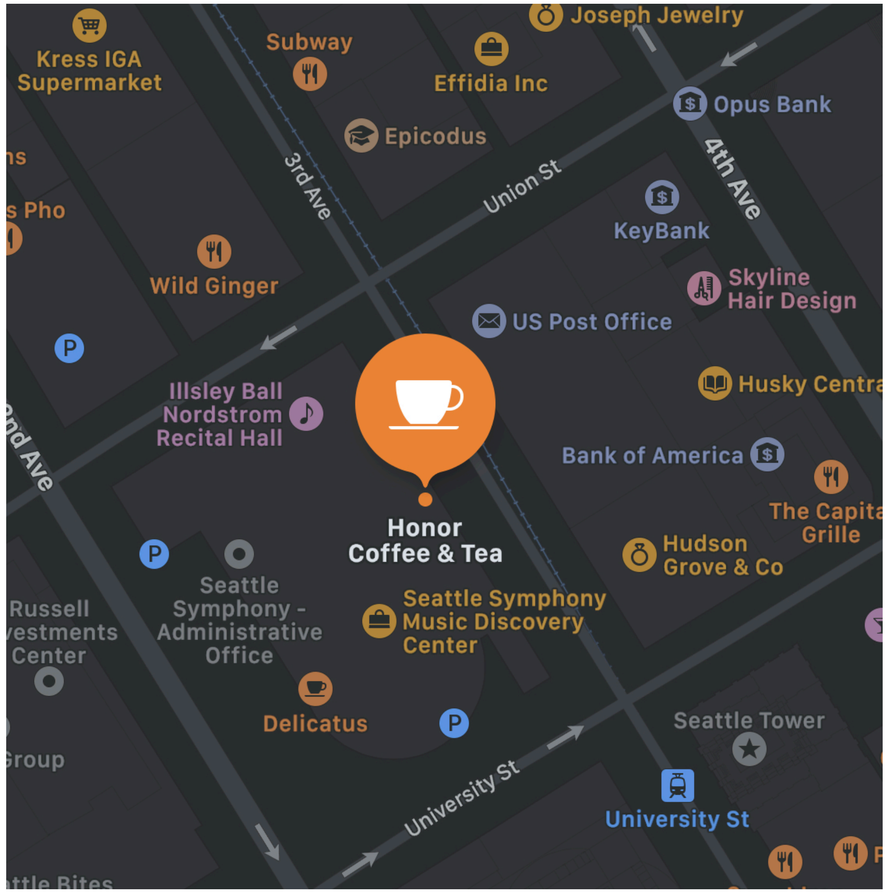
MapKit

Setup

- Use a MKMapView in Storyboard
- Request permission if using user's location
  - *mapView.showsUserLocation = true* is a quick way to display this
  - Doesn't work on simulator!
- Set up initial region
  - Coordinate centering and zooming

Annotations

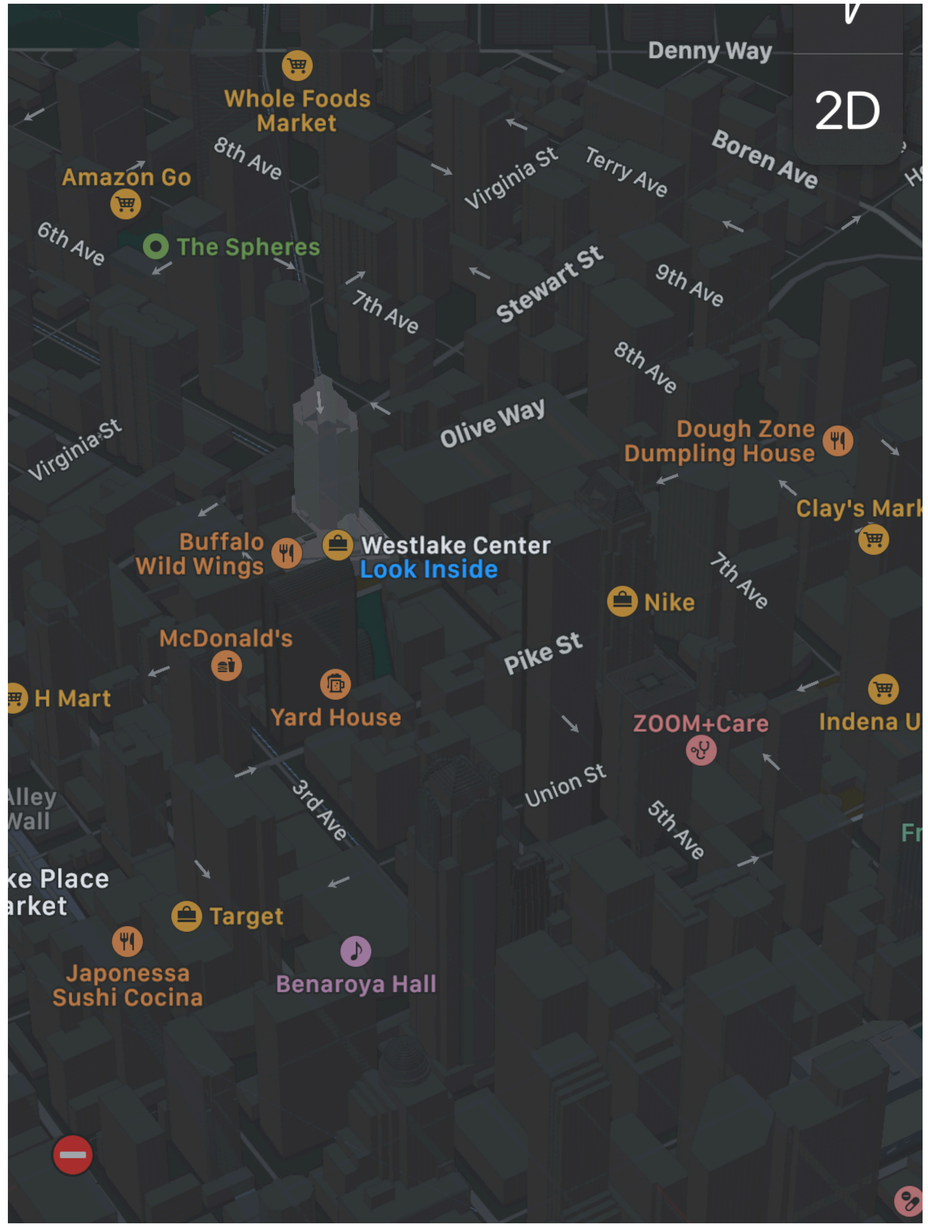
- Adopt the **MKAnnotation** protocol. This is an interface for associating your content with a specific map location.
  - Conceptually, the model that conforms to this will contain information needed to display a “pin” on a map, as well as the “callout” view displayed when the pin is tapped
  - **MKMarkerAnnotationView** - a balloon-shaped marker
  - **MKPinAnnotationView** - a pin-shaped marker
  - …. or your own!



MKMapDelegate

```
mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView?
```

- Gets called by MapKit whenever a View is needed at a Pin location (annotation). So it takes in something conforming to MKAnnotation, and outputs something conforming to MKAnnotationView. Each “Icon” in the map is an MKAnnotationView!
- **Dequeue annotation views with reuse identifiers!**
  - What does this remind you of?



MKMapDelegate

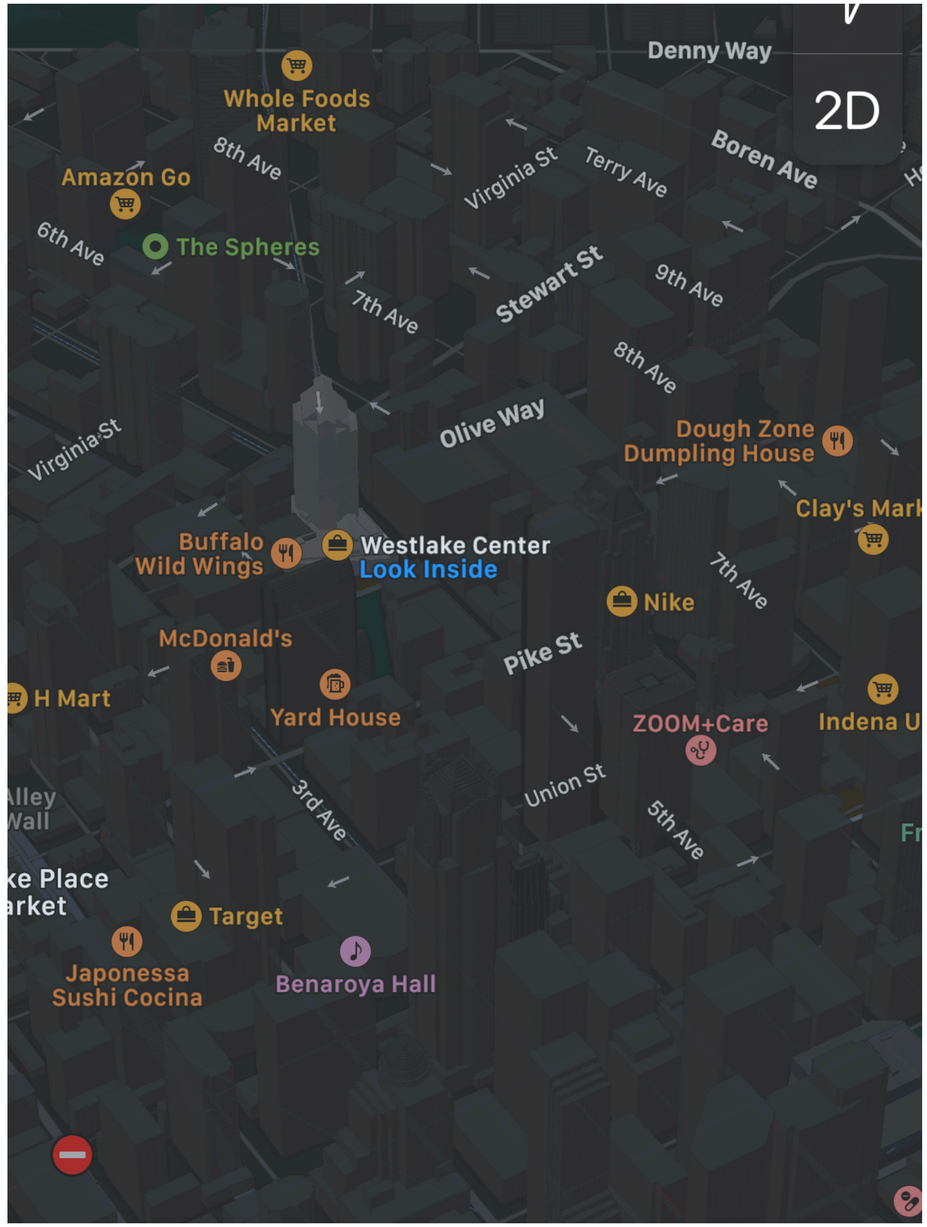
```
mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView?
```

- Gets called by MapKit whenever a View is needed at a Pin location (annotation). So it takes in something conforming to MKAnnotation, and outputs something conforming to MKAnnotationView. Each “Icon” in the map is an MKAnnotationView!

- **Dequeue annotation views with reuse identifiers!**

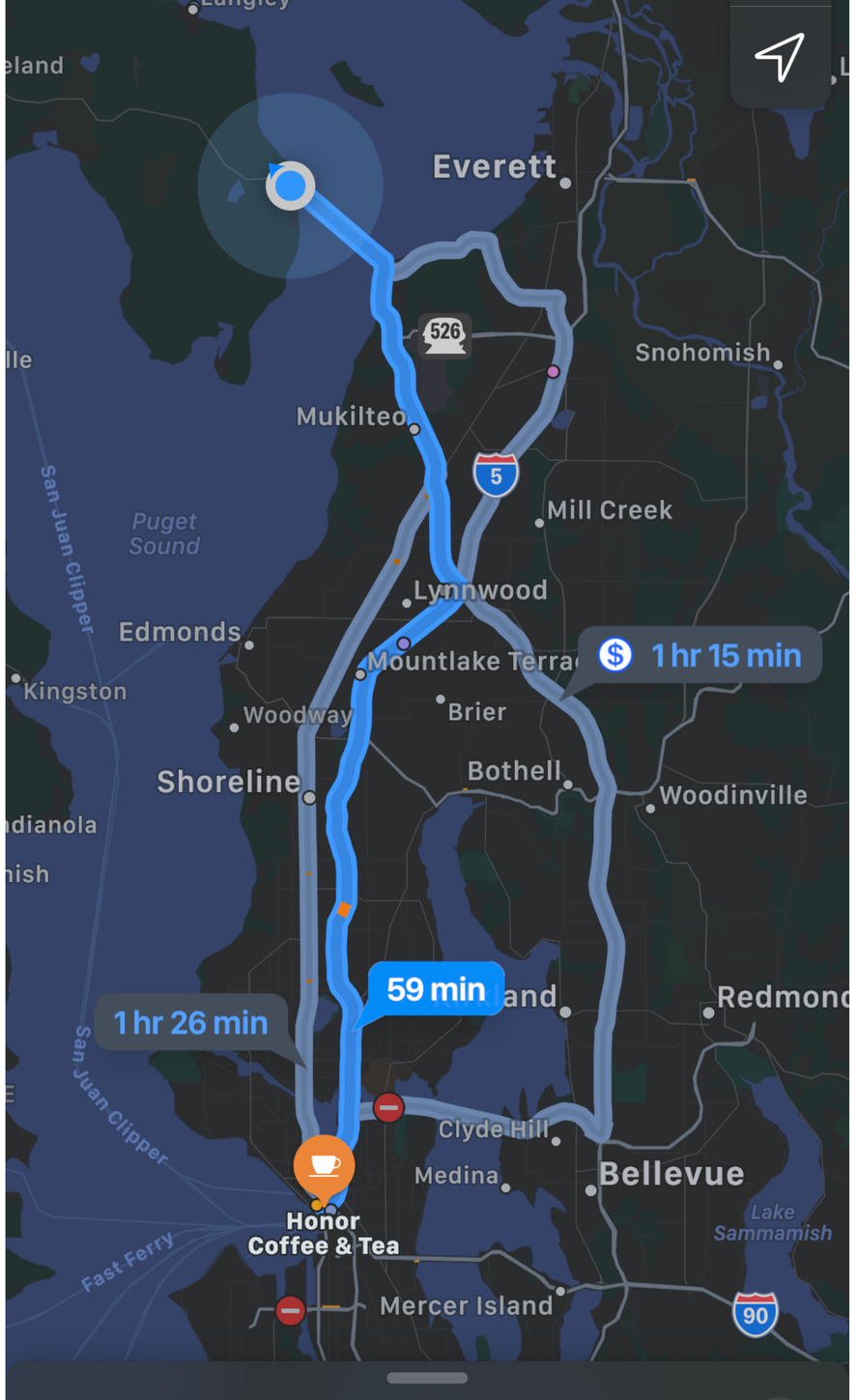
  - What does this remind you of?

  - **tableView(cellForRowAt:)**



Overlays

- **Layering** - Overlays give us a way to layer content over an arbitrary region of the map. We can create shapes, sets of lines, and even polylines and polygons.
    - Layer traffic information, draw the path a user has walked, show park boundaries, etc
- **MKOverlay** - this is a protocol which manages the data points for the overlay
- **MKOverlayRenderer** - used to draw visual representations of the overlay on the map
    - Again, we use a **delegate** method to take in Overlays and return OverlayRenderers



Other Resources

- Slightly outdated, but a great resource:
  - https://www.raywenderlich.com/548-mapkit-tutorial-getting-started#toc-anchor-006
- Apple docs for MapKit/CL are pretty good!

Live Demo: Firebase Setup, MapKit

https://firebase.google.com/docs/ios/setup

Due Before Next Class

- **App 7: Firebase + MapKit**
- **Initial Final Proj. Ideas (completion grade)**

Links

- Piazza: **tiny.cc/cis195-piazza**