BeaconDetector.Swift

@Published var lastDistance = CLProximity.unknown
@Published var lastBeacon = BeaconRef(
uuid: "7777772E-6B6B-6D63-6E2E-636F6D000001",
major: 0,
minor: 0)

These two variables are constantly updated by the following functions upon change of beacon distance

locationManager(beacon)

If beacon is found, make sure the **beacon**'s proximity is not the same as the **lastDistance**
– This is done to not update the beacon too often, and to only update it on a change

update the distance and beacon accordingly, using **updateDistance** and **updateBeacon**

updateDistance(distance)

This updates the lastDistance var which will be passed onto the Home.swift file

updateBeacon(currentMajor: Int, currentMinor: Int)

This updates the lastBeacon variable, which will be passed onto Home.swift's View

Home.swift

currentBeaconDistances: Array<>

An array of 5 of the last beacon distances. Constantly removes first item, appends last item upon .onReceive

.onReceive(detector.$lastDistance) {

When detector notices a change in distance, updates this value and following functions fire :

validate(lastDistance: CLProximity)

validate(lastDistance: CLProximity)
Functionality based on beacon proximity

if("near" or "immediate" count >= 3) {

This means that the phone has detected .near or .immediate more than 3 out of the last 5 reads. This way, it can account for any *wild* reads

loadBeacon()
beginSession()

if("far" or "unknown" count > 3)

This means that the phone has detected more than 3 .near or .unknown reads in the last 5 reads

unloadBeacon()
endSession()

CurrentBeacon.swift

@Published var beacon = Beacon(UUID: "", major: "", minor: "", name: "", sizes: [])

This serves as the master current beacon for the app. Functions alter this beacon to reflect the current beacon that the user is located next to. It is initialized with empty values

func loadBeacon(major: Int, minor: Int, uid: String)

First, checks to see if the input values are not the same as the published beacon

Makes API call to FireStore to get the details of the beacon

UPDATES the published (current) beacon for the entire app with the information received from the FireStore API call

func unloadBeacon()

UPDATES the published (current) beacon for the entire app to empty values

Api.swift

beginSession()

FIrst, checks to see if the user's currentBeacon is already in use. It needs to be terminated first in order to continue
Next, sends the user object with corresponding data (email, first name, styles) to an endpoint. The endpoint will take care of websocket functionality (to iPad)

endSession()

Sends a reques to an endpoint that will handle destruction of websocket connection (to iPad)