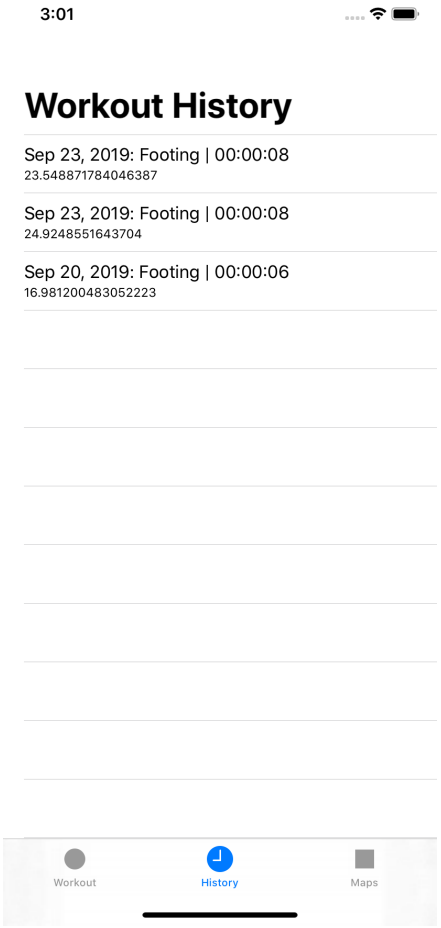
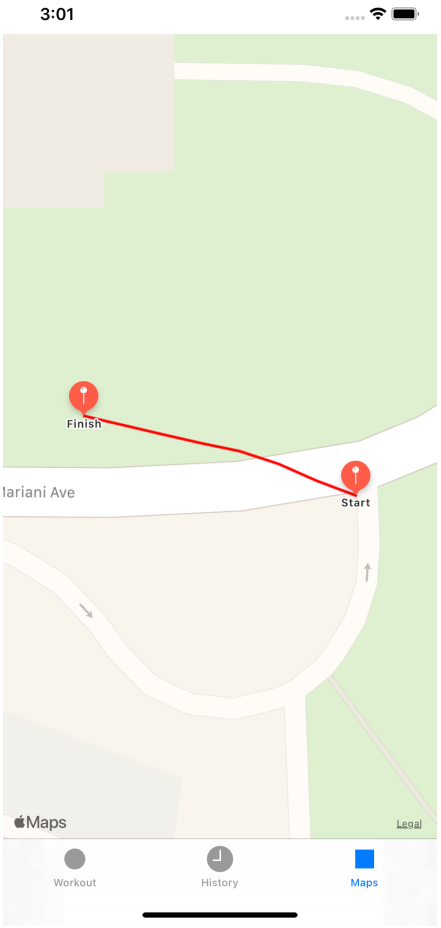
FITIot - Part 1

The aim of this first part is, in a first time, to design the UI part of the app. This UI can be similar to the following one:





When a user hits the start button, the session (or workout) view will be the following:

To manage a timer, please use the Timer API. For the distance covered, you will use the CoreLocation API. These two APIs are well documented and numerous tutorials exists on the web.

When the user hits Resume the session must… resume.

When the user hits Save, the session must be saved (on disk and Healthkit) but this will be the purpose of the next part of the project.

FITIot - Part 2

In this part we will focus on saving the session on disk and display it on a map (the loading process will also be implemented).

Saving a session on disk

To save a session on disk (and later in HealthKit) it would be interesting to model it. For this purpose you could use a `struct` with 5 fields: `startTime (Date)`, `endTime (Date)`, `duration (TimeInterval)`, `locations ([Coordinate])` and `distance (Double)`. The `Coordinate` type can also be a struct with two fields: `latitude` and `longitude` both of `Double` type.

The saving part will consists in saving the `session` object in the form of a `plist` file. To do this you can use the `PropertyListEncoder` (`PropertyListDecoder` for loading) class. This one will allow to encode a swift object into a `plist` file which will be saved in the `documentDirectory` of the `FileManager`. To use the decoder/encoder classes, your object must implement the `Codable` protocol: this is the equivalent of the Serialization concept in Java.

Be careful to not overload the main `ViewController.` Feel free to create a new class for the data management.

Displaying the last session on a map

Since the session is saved on disk and contains an array of the different locations of the run, you can display its path on map.
To do this, you can use the `MKAnnotation` class (allowing to dispaly a pin on a map).
To display a path, you can use the `MKPolyline` and `MKOverlay` classes.

FITIot - Part 3

In this part we will focus on saving the session on using the HealthKit repository to save and load health data. For a first tutorial about HealthKit please see: https://agostini.tech/2019/01/07/using-healthkit/

HealthKit Authorization

First, check if HealthKit is available (for example it is not available on iPad): HKHealthStore.isHealthDataAvailable()

Then, prepare the data type that will interact with HealthKit. For example, for the walking distance:
let distanceWalkingRunning =
HKSampleType.quantityType(forIdentifier: .distanceWalkingRunning)

Prepare a list of types you want HealthKit to read and write:

```
let healthKitTypesToWrite: Set<HKSampleType> = [activeEnergy,
                                                distanceWalkingRunning,
                                                HKObjectType.workoutType()]

let healthKitTypesToRead: Set<HKObjectType> = [activeEnergy,
                                               distanceWalkingRunning,
                                               HKObjectType.workoutType()]
```

Finally, Request Authorization

```
HKHealthStore().requestAuthorization(toShare: healthKitTypesToWrite,
                                     read: healthKitTypesToRead) { (success, error) in
  ...
}}
```

Saving a session on disk

The functions to save data on HealthKit can be the followings:

```
func saveWorkoutToHealthKit(_ workout: Workout) {

        let distanceQuantity = HKQuantity(unit: HKUnit.meter(),
doubleValue: workout.distance)

        let workoutObject = HKWorkout(activityType:
HKWorkoutActivityType.running, start: workout.startTime, end:
workout.endTime, duration: workout.duration, totalEnergyBurned: nil,
totalDistance: distanceQuantity, metadata: nil)

        healthStore?.save(workoutObject, withCompletion: { (completed,
error) in
            if let error = error {
                print("Error creating workout")
            } else {
                self.addSamples(hkWorkout: workoutObject, workoutData:
workout)
            }
        })
    }
```

```
    func addSamples(hkWorkout: HKWorkout, workoutData: Workout) {

        var samples = [HKSample]()

        guard let runningDistanceType =
HKQuantityType.quantityType(forIdentifier:
HKQuantityTypeIdentifier.distanceWalkingRunning) else { return }
        let distanceUnit = HKUnit.meter()
        let distanceQuantity = HKQuantity(unit: distanceUnit,
doubleValue: workoutData.distance)
        let distanceSample = HKQuantitySample(type: runningDistanceType,
quantity: distanceQuantity, start: workoutData.startTime, end:
workoutData.endTime)

        samples.append(distanceSample)

        healthStore?.add(samples, to: hkWorkout, completion:
{ (completed, error) in
            if let error = error {
                print("Error adding workout samples")
            } else {
                print("Workout samples added successfully")

            }
        })

    }
```

Loadind data from HealthKit

It’s up to you