COSC 4355/6355 – Introduction to Ubiquitous Computing

Exercise - 7

October 26, 2023

Objective

Create an iWatch app that reads heart rate data and displays it real-time.

Motivation

You will learn how to use: HealthKit, watch app, .

Must Follow

You must follow the rules below. Otherwise, you will get **50% of your actual score.**

1. Start your XCode project "**Exercise7_LastName_FirstName**" (replace **LastName** with your last name and **FirstName** with your first name).
a. **DON'T DO** the following:
   i. Start with any other project name and change the zip file name later.
   ii. This will not be accepted at all.

2. You must have to do the exercise compatible with **XCode version 14.1**.

Tips

- **Read the question carefully, then start coding!**
- **Build, Build, and Build**
    - If you add anything on storyboard -> **Build**
    - If you make a reference from storyboard -> **Build**
    - Do not wait until finishing all parts and build.
    - It is easier to debug after each single feature added.

Details

Create a single view iWatch application using Swift as a programming language. Start your XCode project "**Exercise7_LastName_FirstName**" (replace **LastName** with your last name and **FirstName** with your first name).

[5 pts] Design your interface to look like the screenshots [Figures 1-3]
• Design your UI for the Apple Watch
• Single button with the heart emoji, system font, size 50
• Heart Rate value, system font, size 70, regular weight
• "BPM" text, headline font, size 28, bold weight

[2 pts] When user launches the app for the first time it creates permission requests [Figures 4-6].

[1 pt.] After permissions were granted, app is ready to collect HR data (works only for real watch). Simulator will show "0" [Figure 1]

[2 pts] When user taps heart button, emulated outdoor running heart rate data is generated. You might need to restart the app after granting permissions to receive emulated data [Figures 2-3]

• The application should not crash permissions were not granted. Add an appropriate alert for the user in case of any error.

• Make sure to add the appropriate properties to the target properties list:
Privacy – Health Share Usage Description
Privacy – Health Update Usage Description
Privacy – Health Records Usage Description

Bonus

[1 pt.] Create a paired phone app (with a single text label) and transfer heart-rate data to the phone in real time, at least once every 2 sec. (no particular design requirements here, you may add heart to make it similar looking)
[1 pt.] Add data simulation control from the watch (start/stop)
[1 pt.] Add data simulation control from the phone (start/stop)

Submission

Zip XCode project and submit to the blackboard. The name of your zip file will be automatically "**Exercise7_LastName_FirstName.zip**" (**Last Name** is your last name and **FirstName** is your first name). One submission per person.

UNIVERSITY of HOUSTON

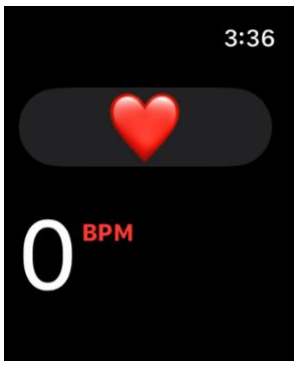

Figure 1

Figure 2

Figure 3

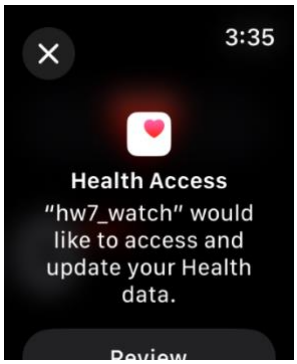

Figure 4

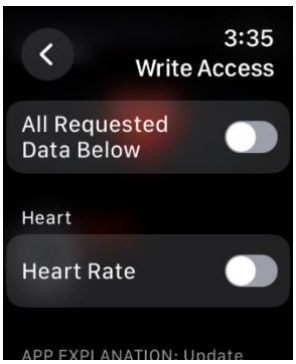

Figure 5

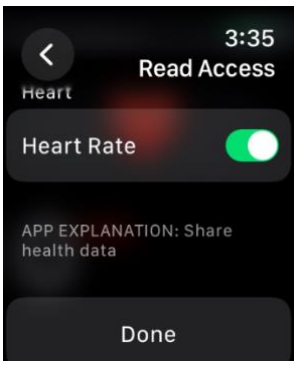

Figure 6

UNIVERSITY of HOUSTON

Hints:

HINTS

As HealthKit will not be fully covered in the class, here you may find some useful code-snippets. Copy-paste protection activated ☺

1. Perform some function on view appearance:

```
.onAppear(perform: start)
```

2. Request for the authorization:

```
func autorizeHealthKit() {
    let healthKitTypes: Set = [
    HKObjectType.quantityType(forIdentifier:
        HKQuantityTypeIdentifier.heartRate)!]

    healthStore.requestAuthorization(toShare: healthKitTypes, read:
        healthKitTypes) { _, _ in }
}
```

3. Starting query to collect data.

```
private func startHeartRateQuery(quantityTypeIdentifier: HKQuantityTypeIdentifier) {

    // 1 collect data from the device
    let devicePredicate = HKQuery.predicateForObjects(from: [HKDevice.local()])
    // 2 updating handler
    let updateHandler: (HKAnchoredObjectQuery, [HKSample]?, [HKDeletedObject]?, HKQueryAnchor?, Error?) -> Void = {
        query, samples, deletedObjects, queryAnchor, error in

    guard let samples = samples as? [HKQuantitySample] else {
        return
    }

    self.process(samples, type: quantityTypeIdentifier)

    }

    let query = HKAnchoredObjectQuery(type: HKObjectType.quantityType(forIdentifier: quantityTypeIdentifier)!, predicate:
        devicePredicate, anchor: nil, limit: HKObjectQueryNoLimit, resultsHandler: updateHandler)

    query.updateHandler = updateHandler

    healthStore.execute(query)

}
```

4. Creating workout emulation. That is for the ‘heart’ button.

```
private func createWorkoutEmulation(){
    // create configuration
    let configuration = HKWorkoutConfiguration()
    configuration.activityType = .running
    configuration.locationType = .outdoor
    // init store, create session and builder variables
    let healthStore = HKHealthStore()
    var session: HKWorkoutSession!
    var builder: HKWorkoutBuilder!
    // create session
    do {
        session = try HKWorkoutSession(healthStore: healthStore, configuration: configuration)
        builder = session.associatedWorkoutBuilder()
    } catch {
        return
    }
    // start session and builder
    session.startActivity(with: Date())
    builder.beginCollection(withStart: Date()) {(success, error) in

    }
}
```

5. Process last heart rate value

```
private func process(_ samples: [HKQuantitySample], type: HKQuantityTypeIdentifier) {
    var lastHeartRate = 0.0

    for sample in samples {
        if type == .heartRate {
            lastHeartRate = sample.quantity.doubleValue(for: heartRateQuantity)
        }
        self.value = Int(lastHeartRate)
    }
}
```

6. Function to run after the view will appear:

```
    func start() {
        // authorize monitoring
        autorizeHealthKit()

        // begin collecting heart data
        startHeartRateQuery(quantityTypeIdentifier: .heartRate)
    }
```

Again, don’t forget that data simulation might require app restart.