Rushad Daruwalla, Jay Devang Parikh, Sunny Yadav

CS 5520 IOS mobile development

Sakib Miazi

Tutorial Report: Implementing CoreLocation Geofencing

For our additional tutorial report, we decided to explore an iOS SDK feature that connects to what we worked on this semester but goes a bit beyond what we covered in class. We chose CoreLocation's geofencing using CLCircularRegion. This lets an app detect when the user enters or exits a specific place in the real world and react automatically, even if the app is not open. It brings together permissions, background events, notifications, and device movement, which fits well with how we have been thinking about event-driven design. It also lines up naturally with our FitQuest idea, because we can imagine tasks being triggered when we physically arrive at a gym, a running trail, or some other meaningful location.

The basic idea behind this tool is simple: we define a circular region with a center coordinate and a radius, and we ask iOS to monitor that region. Once monitoring is active, the system calls our delegate methods when the user crosses the boundary. One of the main advantages is that iOS handles the heavy work in the background, so the app does not need to constantly pull location updates or stay active. This makes it much more practical and battery friendly. Since these enter and exit events often happen when the app is not open, it pairs nicely with local notifications, which give the user immediate feedback when something important happens.

To keep the implementation clean, we organized the geofencing utility into a small manager class plus a simple view controller that shows how to use it. Step by step, the setup looks like this. First, we would create a GeofenceManager class that owns a CLLocationManager and acts as its delegate.

```
import CoreLocation
import UserNotifications
import UIKit

class GeofenceManager: NSObject, CLLocationManagerDelegate {

    static let shared = GeofenceManager()
    private let manager = CLLocationManager()

    override init() {
        super.init()
        manager.delegate = self
        manager.desiredAccuracy = kCLLocationAccuracyBest
    }
}
```

Here, we are using a shared singleton instance so the same manager can be reached from anywhere in the app if needed. We set the delegate to self and configure the desired accuracy. This sets up the core object that will interact with the system. Next, we'd need to handle permissions for both location and notifications. Since geofencing can trigger events when the app is not in the foreground, we can request "Always" authorization, and we could also ask the user for notification permission.

```
   func requestPermissions() {
        manager.requestAlwaysAuthorization()
        UNUserNotificationCenter.current()
            .requestAuthorization(options: [.alert, .sound, .badge]) {
_, _ in }
    }
```

This method can be called once early in the app's lifecycle, like when a relevant screen appears for the first time. It keeps both permission requests in one place, which makes the behavior easier to reason about and reuse. After permissions, the key step is defining and monitoring a region. We wrapped this in a startMonitoring function.

```
   func startMonitoring(center: CLLocationCoordinate2D,
                         radius: CLLocationDistance,
                         identifier: String) {
        let region = CLCircularRegion(center: center, radius: radius,
identifier: identifier)
        region.notifyOnEntry = true
        region.notifyOnExit = true
        manager.startMonitoring(for: region)
    }
```

Here we'd have a CLCircularRegion with the given center and radius, and we turn on both entry and exit notifications. The identifier string is important because it lets us distinguish between different locations later. Once startMonitoring is called, iOS starts watching that region in the background. The next part is reacting to what happens. When the user enters or exits one of our monitored regions, the system calls the corresponding delegate methods. In our manager, we'd implement them and use local notifications to inform the user.

```
   func locationManager(_ manager: CLLocationManager,
                        didEnterRegion region: CLRegion) {
        sendNotification("Arrived", "You entered
\(region.identifier).")
    }

    func locationManager(_ manager: CLLocationManager,
                        didExitRegion region: CLRegion) {
        sendNotification("Leaving", "You exited
\(region.identifier).")
    }
```

Both delegate methods call a helper that actually sends the notification. Keeping notification logic in a separate method makes it easier to change or expand in the future.

```
   private func sendNotification(_ title: String, _ body: String) {
        let content = UNMutableNotificationContent()
        content.title = title
        content.body = body

        UNUserNotificationCenter.current().add(
            UNNotificationRequest(identifier: UUID().uuidString,
                                  content: content,
                                  trigger:
UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false))
        )
    }
}
```

With this, our GeofenceManager is a complete, reusable utility. It knows how to request permissions, start monitoring regions, and respond to movement with notifications. To show how to actually integrate this into an iOS project, we would have a small view controller that uses the manager. In viewDidLoad, we request permissions and start monitoring a specific location (in this example, a "Gym" coordinate):

```
class GeofenceViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        GeofenceManager.shared.requestPermissions()

        let gym = CLLocationCoordinate2D(latitude: 42.3389, longitude:
-71.0910)
        GeofenceManager.shared.startMonitoring(center: gym,
                                                radius: 200,
                                                identifier: "Gym")
    }
}
```

This code is intentionally simple. It shows how another developer on the team could integrate the utility into a screen in just a couple of lines. Once this is set up, the system will automatically notify the app when the user enters or exits the gym region, and our delegate methods in GeofenceManager will send the appropriate notifications. In a FitQuest setting, that could trigger a workout reminder, log the start of a session, or count toward some sort of location-based streak without the user needing to press anything.

From a documentation point of view, there are just a couple of important things to keep in mind for this feature to actually work. The app's Info.plist needs the correct location usage descriptions, specifically NSLocationWhenInUseUsageDescription and NSLocationAlwaysAndWhenInUseUsageDescription, because iOS won't let us monitor regions without telling the user why we need their location. Some older iOS versions also look for NSLocationAlwaysUsageDescription, but the first two are the main ones we rely on now. Since we're sending local notifications whenever a region event happens, we also have to request notification permission, which we already take care of inside requestPermissions(). After that, the setup is pretty simple. We call requestPermissions() once, then use startMonitoring to add whatever geofenced regions we want, each with its own identifier so we know which one the user entered or exited. Once that's done, the system handles all the background monitoring automatically, and our code only runs when something actually happens.

FULL CODE EXAMPLE:

Geofence Manager

```
import CoreLocation
import UserNotifications
import UIKit


class GeofenceManager: NSObject, CLLocationManagerDelegate {

    static let shared = GeofenceManager()
    private let manager = CLLocationManager()

    override init() {
        super.init()
        manager.delegate = self
        manager.desiredAccuracy = kCLLocationAccuracyBest
    }

    func requestPermissions() {
        manager.requestAlwaysAuthorization()
        UNUserNotificationCenter.current()
            .requestAuthorization(options: [.alert, .sound, .badge]) {
_, _ in }
    }

    func startMonitoring(center: CLLocationCoordinate2D,
                         radius: CLLocationDistance,
                         identifier: String) {

        let region = CLCircularRegion(center: center, radius: radius,
identifier: identifier)
        region.notifyOnEntry = true
        region.notifyOnExit = true

        manager.startMonitoring(for: region)
    }

    func locationManager(_ manager: CLLocationManager,
                         didEnterRegion region: CLRegion) {
        sendNotification("Arrived", "You entered
\(region.identifier).")
    }

    func locationManager(_ manager: CLLocationManager,
                         didExitRegion region: CLRegion) {
        sendNotification("Leaving", "You exited
\(region.identifier).")
    }

    private func sendNotification(_ title: String, _ body: String) {
        let content = UNMutableNotificationContent()
        content.title = title
        content.body = body

        let request = UNNotificationRequest(
            identifier: UUID().uuidString,
            content: content,
            trigger: UNTimeIntervalNotificationTrigger(timeInterval:
1, repeats: false)
        )

        UNUserNotificationCenter.current().add(request)
    }
}
```

ViewController

```
import UIKit
import CoreLocation

class GeofenceViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        GeofenceManager.shared.requestPermissions()

        let gym = CLLocationCoordinate2D(latitude: 42.3389, longitude:
-71.0910)
```

```
        GeofenceManager.shared.startMonitoring(center: gym,
                                               radius: 200,
                                               identifier: "Gym")
    }
}
```

References:

“Core Location Framework.” *Apple Developer Documentation*, Apple Inc., https://developer.apple.com/documentation/corelocation.

“Region Monitoring and User Proximity.” *Apple Developer Documentation*, Apple Inc., https://developer.apple.com/documentation/corelocation/monitoring_the_user_s_proximity_to_geographic_regions

“User Notifications Framework.” *Apple Developer Documentation*, Apple Inc., https://developer.apple.com/documentation/usernotifications.