Smart User Engagement with Dynamic Notifications in Mobile App

Introduction

This document explains how Mobile App uses smart local notifications to improve user engagement. The notification system dynamically triggers reminders if users don't open the app within a specified number of days. It personalizes the notification timing based on user behavior and is fully controlled via Firebase Remote Config.

Key Features

- Dynamic Inactivity Trigger: Sends reminders if the app isn't opened for 1-3 days (configurable).
- Personalized Notification Time: Defaults to 2 PM local time or adjusts based on the user's last active time.
- Firebase Remote Config Control: Instantly change notification settings without updating the app.
- Randomized Messages: Keeps content fresh with a predefined set of motivational messages.
- Interactive Notifications: Includes "Open App" and "Close" actions for quick user responses.

Why Cancelling Previous Notifications is Important

When a user opens the app again, the previously scheduled notification may become irrelevant. If we don't cancel outdated notifications, users might receive reminders even though they're active, leading to a **poor user experience**.



- 1. Track User Interaction Using Notification Response Handlers Detect if the notification was **opened**, **dismissed**, or **ignored**.
- 2. Reschedule Notifications After User Dismissal or Timeout Automatically reschedule if the notification is dismissed or ignored.
- 3. Remote Config Controls Settings Firebase Remote Config manages notification frequency and enable/disable status.

Complete Swift Code Implementation

1. AppDelegate to Handle Notifications

UNUserNotificationCenter.current().delegate = self

import UIKit import UserNotifications

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate, UNUserNotificationCenterDelegate {
```

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:

```
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
     UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) {
granted, error in
       if granted {
          print("Notification permission granted.")
          print("Notification permission denied.")
       }
```

func userNotificationCenter(center: UNUserNotificationCenter, didReceive response: UNNotificationResponse, withCompletionHandler completionHandler: @escaping () -> Void) { let identifier = response.actionIdentifier

```
switch identifier {
```

return true

}

```
case "OPEN APP":
       print("User opened the app from notification.")
       UserEngagementHelper.shared.scheduleEngagementNotification(lastActiveDate:
Date())
    case "CLOSE":
       print("User dismissed the notification.")
       UserEngagementHelper.shared.rescheduleAfterDismissal()
    default:
       print("Notification ignored.")
       UserEngagementHelper.shared.rescheduleAfterDismissal()
    }
    completionHandler()
  }
```

📦 2. UserEngagementHelper for Scheduling & Rescheduling

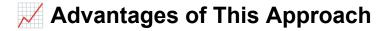
```
import Foundation
import UserNotifications
import FirebaseRemoteConfig
class UserEngagementHelper {
  static let shared = UserEngagementHelper()
  private let remoteConfig = RemoteConfig.remoteConfig()
  private init() {}
  func scheduleEngagementNotification(lastActiveDate: Date?) {
     remoteConfig.fetchAndActivate { [weak self] status, error in
       guard let self = self else { return }
       let isNotificationEnabled = self.remoteConfig["isNotificationEnabled"].boolValue
       let triggerDays = self.remoteConfig["triggerDays"].numberValue?.intValue ?? 2
       guard isNotificationEnabled else { return }
       self.cancelScheduledNotifications()
       let triggerDate = self.calculateTriggerDate(from: lastActiveDate, days: triggerDays)
       self.scheduleLocalNotification(at: triggerDate)
    }
```

```
}
  func rescheduleAfterDismissal() {
     remoteConfig.fetchAndActivate { [weak self] status, error in
       guard let self = self else { return }
       let triggerDays = self.remoteConfig["triggerDays"].numberValue?.intValue?? 2
       let rescheduleDate = Calendar.current.date(byAdding: .day, value: triggerDays, to:
Date())!
       self.cancelScheduledNotifications()
       self.scheduleLocalNotification(at: rescheduleDate)
    }
  }
  private func cancelScheduledNotifications() {
     UNUserNotificationCenter.current().removePendingNotificationRequests(withIdentifiers:
["MOBILE APP REMINDER"])
     print("Previous notifications canceled.")
  }
  private func calculateTriggerDate(from lastActiveDate: Date?, days: Int) -> Date {
     let calendar = Calendar.current
     let now = Date()
     var triggerDate = calendar.date(bySettingHour: 14, minute: 0, second: 0, of: now)!
     if let lastActive = lastActiveDate {
       triggerDate = calendar.date(byAdding: .day, value: days, to: lastActive) ?? triggerDate
    }
     return triggerDate
  }
  private func scheduleLocalNotification(at date: Date) {
     let content = UNMutableNotificationContent()
     content.title = "Mobile App Awaits! \sum_"
     content.body = ["Ready for your next big catch?", "Don't miss out on today's best fishing
spots!", "Explore new waters and log your catches!", "The fish are waiting! Open the app
now!"].randomElement()!
     content.sound = .default
     content.categoryldentifier = "MOBILE APP NOTIFICATION"
```

```
let trigger = UNCalendarNotificationTrigger(dateMatching:
Calendar.current.dateComponents([.year, .month, .day, .hour, .minute], from: date), repeats:
false)
     let request = UNNotificationRequest(identifier: "MOBILE APP REMINDER", content:
content, trigger: trigger)
     configureNotificationActions()
     UNUserNotificationCenter.current().add(request) { error in
       if let error = error {
          print("Failed to schedule notification: \(error.localizedDescription)")
          print("Notification scheduled for \(date)")
    }
  }
  private func configureNotificationActions() {
     let openAppAction = UNNotificationAction(identifier: "OPEN APP", title: "Open App",
options: [.foreground])
     let closeAction = UNNotificationAction(identifier: "CLOSE", title: "Close", options:
[.destructive])
     let category = UNNotificationCategory(identifier: "MOBILE APP NOTIFICATION", actions:
[openAppAction, closeAction], intentIdentifiers: [], options: [])
     UNUserNotificationCenter.current().setNotificationCategories([category])
  }
}
```

*** How It Works**

- 1. User Opens the App from Notification:
 - Resets the schedule based on the new activity.
- 2. User Clicks "Close":
 - Cancels the current notification and reschedules the next notification after the triggerDays interval.
- 3. User Ignores the Notification:
 - Automatically reschedules the next notification after the defined interval.



- No Redundant Notifications: Cancels outdated reminders.
- Smart Rescheduling: Adapts to user behavior.
- **Dynamic Control:** Firebase Remote Config controls frequency without updates.
- Improved Engagement: Keeps notifications relevant.

III Future Improvements

- Track Interaction Metrics: Analyze how users interact with notifications.
- A/B Testing: Test different message styles, timings, and frequencies.
- Personalized Notifications: Include recent catches or stats for relevance.

* Conclusion

This smart notification system helps Mobile App improve user retention by sending personalized, timely reminders. It's flexible, data-driven, and optimized for an excellent user experience.