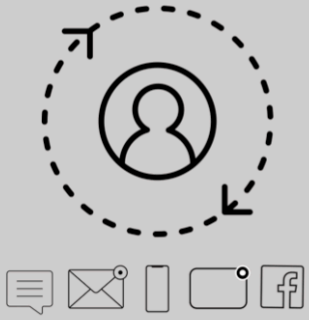
appICE

AI & ML DRIVEN

MARKETING AUTOMATION

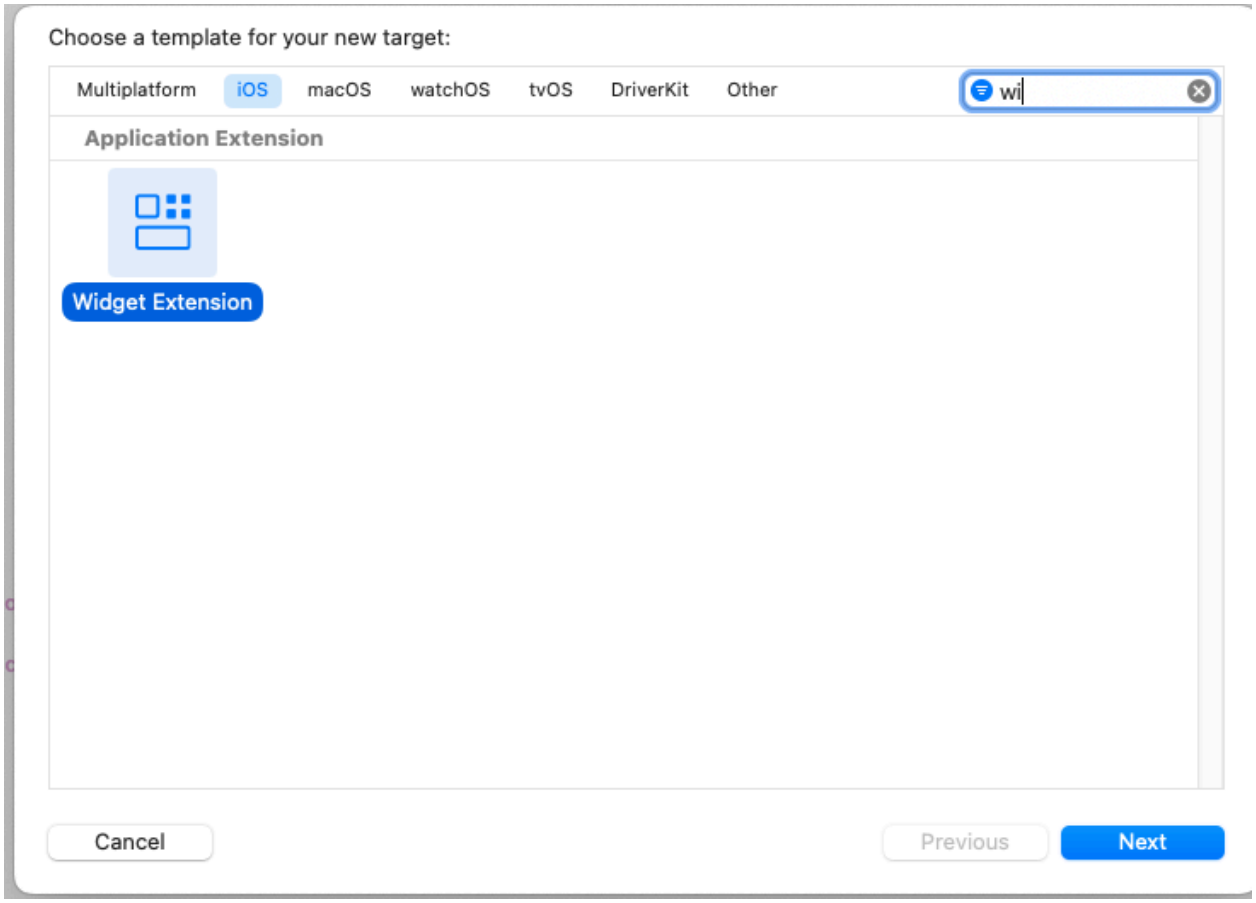


applCE Product Manual

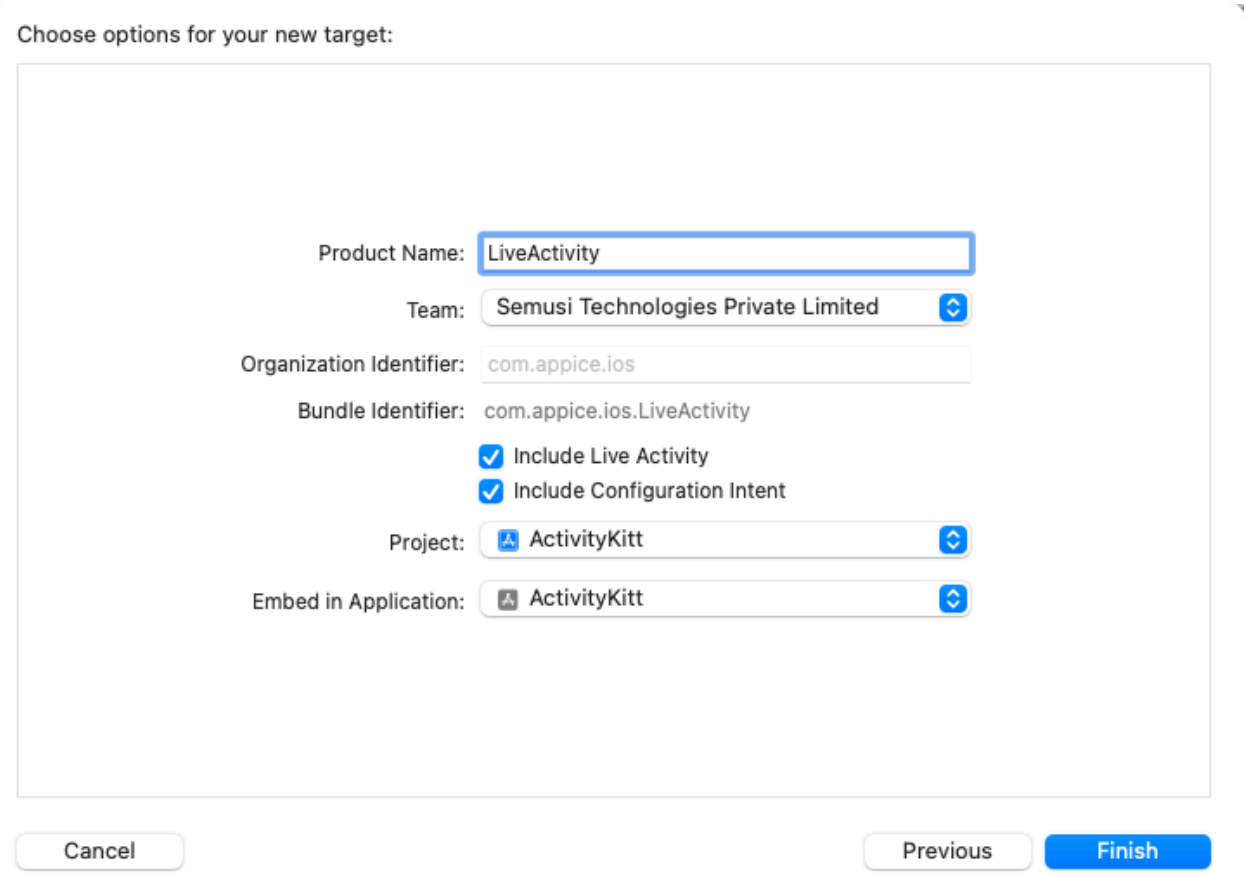
Submitted To

LIVE ACTIVITY for iOS

Step 1: File- > New- > Target->WidgetExtension



Step 2: Create Extension Name Ex: LiveActivity and select Include Live Activity



Step 3: In **LiveActivityBundle.swift** file

```
import WidgetKit
import SwiftUI
@main
struct LiveActivityBundle: WidgetBundle {
   var body: some Widget {
      LiveActivity()
      LiveActivityLiveActivity()
   }
}
```

Step 4: Sample code In **LiveActivityLiveActivity.swift**

```
import ActivityKit
import WidgetKit
import SwiftUI
```

```
struct DeliveryWidget: Widget {
  var body: some WidgetConfiguration {
    ActivityConfiguration(for: DeliveryAttributes.self) { context in
      // Create the presentation that appears on the Lock Screen and as a
      // banner on the Home Screen of devices that don't support the
      // Dynamic Island.
      LockScreenView(context: context)
    } dynamicIsland: { context in
      // Create the presentations that appear in the Dynamic Island.
      DynamicIsland {
        //Extract needed information from Dict
        let allMainAttributes = context.attributes.mainAttributes
        let numberOfItems = allMainAttributes["numberOfItems"]?.value as? Int ?? 0
        let alldynamicAttributes = context.state.dynamicAttributes
        let estimatedTimeRemaining = alldynamicAttributes?["estimatedTimeRemaining"]?.value as?
Int ?? 0
        let customerName = alldynamicAttributes?["customerName"]?.value as? String ?? "CK-1"

        //let allContentStateDynamicAttribute = context.state.d
        // Create the expanded presentation.
        //TODO: - @(Chhavi Krishan) Need to be disscussed
        DynamicIslandExpandedRegion(.leading) {
          Label(
            "\(numberOfItems)", systemImage: "bag")
            .foregroundColor(.indigo)
            .font(.title2)
        }
        DynamicIslandExpandedRegion(.trailing) {
          Label(formattedTimeRemaining(estimatedTimeRemaining), systemImage: "timer")
            .foregroundColor(.indigo)
            .font(.title2)
            .padding(.trailing, 10)
        }
        DynamicIslandExpandedRegion(.center) {
          Text("\(customerName) is on their way!")
            .lineLimit(1)
            .font(.caption)
            .padding(.top , 2)
        }
```

```
            DynamicIslandExpandedRegion(.bottom) {
                Button {
                    // Deep link into your app.
                } label: {
                    Label("Call driver \(customerName)", systemImage: "phone")
                }
                .foregroundColor(.indigo)
            }
        } compactLeading: {
            let allMainAttributes = context.attributes.mainAttributes
            let numberOfItems = allMainAttributes["numberOfItems"]?.value as? Int ?? 0
            Label {
                Text("\(numberOfItems)")
            } icon: {
                Image(systemName: "bag")
                    .foregroundColor(.indigo)
            }
            .font(.caption2)
        } compactTrailing: {
            let alldynamicAttributes = context.state.dynamicAttributes
            let estimatedTimeRemaining = alldynamicAttributes?["estimatedTimeRemaining"]?.value as?
Int ?? 0
            Label(formattedTimeRemaining(estimatedTimeRemaining), systemImage: "timer")
                .foregroundColor(.indigo)
                .font(.caption2)
        } minimal: {
            let alldynamicAttributes = context.state.dynamicAttributes
            let estimatedTimeRemaining = alldynamicAttributes?["estimatedTimeRemaining"]?.value as?
Int ?? 0
            VStack(alignment: .center) {
                Image(systemName: "timer")
                Text(formattedTimeRemaining(estimatedTimeRemaining))
                    .multilineTextAlignment(.center)
                    .font(.caption2)
            }
        }
        .keylineTint(.cyan)
        .contentMargins(.trailing, 8, for: .expanded)
    }
}
```

```
  // Formats the given time interval into a string representing the total minutes
   private func formattedTimeRemaining(_ timeInterval: Int) -> String {
      let minutes = Int(timeInterval) / 60
      return String(format: "%02d", minutes)
   }
}
```

Step 5: Sample code In **LiveActivity.swift** file

```
import WidgetKit
import SwiftUI
struct Provider: TimelineProvider {
   func placeholder(in context: Context) -> SimpleEntry {
      SimpleEntry(date: Date())
   }
   func getSnapshot(in context: Context, completion: @escaping (SimpleEntry) -> ()) {
      let entry = SimpleEntry(date: Date())
      completion(entry)
   }
   func getTimeline(in context: Context, completion: @escaping (Timeline<Entry>) -> ()) {
      var entries: [SimpleEntry] = []
      // Generate a timeline consisting of five entries an hour apart, starting from the current date.
      let currentDate = Date()
      for hourOffset in 0 ..< 5 {
         let entryDate = Calendar.current.date(byAdding: .hour, value: hourOffset, to: currentDate)!
         let entry = SimpleEntry(date: entryDate)
         entries.append(entry)
      }
      let timeline = Timeline(entries: entries, policy: .atEnd)
      completion(timeline)
   }
}
struct SimpleEntry: TimelineEntry {
   let date: Date
}
struct LiveActivityEntryView : View {
   var entry: Provider.Entry
```

```
    var body: some View {
        Text(entry.date, style: .time)
    }
}
struct LiveActivity: Widget {
    let kind: String = "LiveActivity"
    var body: some WidgetConfiguration {
        StaticConfiguration(kind: kind, provider: Provider()) { entry in
            LiveActivityEntryView(entry: entry)
        }
        .configurationDisplayName("My Widget")
        .description("This is an example widget.")
    }
}
struct LiveActivity_Previews: PreviewProvider {
    static var previews: some View {
        LiveActivityEntryView(entry: SimpleEntry(date: Date()))
            .previewContext(WidgetPreviewContext(family: .systemSmall))
    }
}
```

Step 6: Create a **WidgetViews.swift** file

```
import SwiftUI
import WidgetKit
struct LockScreenView: View {

    let context: ActivityViewContext<DeliveryAttributes>
    @State private var orderStatus: Int = 0
    @State private var titleTextColor: String = "#9c27b0" // Default color for tesing

    // Update ui based on the order status
    private var orderTitle: String {
        let allAttributes = context.state.dynamicAttributes
        let serviceName = allAttributes?["orderTitle"]?.value as? String ?? "Test Order"
        return serviceName
    }
    private var formattedTime: String {
        let minutes = getEstimatedTimeRemaining() / 60
```

```
        let seconds = getEstimatedTimeRemaining() % 60
        return String(format: "%02d:%02d", minutes, seconds)
    }
    private var orderSubtitle: String {
        switch orderStatus {
        case 1...10: return "Delivery successfully."
        case 10...40: return "\(context.state.dynamicAttributes?["serviceName"]?.value as? String ??
"Service-Name") | delivering in \(formattedTime) mins"
        default: return "Order preparation will begin shortly"
        }
    }
    // Retrieves the estimated time remaining directly from the dynamic attributes using a specific key
    private var orderStatusTime: Int {
        context.state.dynamicAttributes?["estimatedTimeRemaining"]?.value as? Int ?? 0
    }
    // Defines a countdown from now to the estimated time remaining
    private var countdownTimer: ClosedRange<Date> {
        Date()...Date().addingTimeInterval(Double(orderStatusTime))
    }
    private var imageOnTop: Bool {
        context.state.dynamicAttributes?["imageOnTop"]?.value as? Bool ?? true
    }
    var body: some View {
        HStack {
            VStack(alignment: .leading, spacing: 8) {
                Text(orderTitle)
                    .font(.headline)
                    .foregroundColor(.primary)
                if orderStatus > 10 {
                    Text("\(context.state.dynamicAttributes?["serviceName"]?.value as? String ??
"Service-Name") | Delivering in ")
                        .font(.subheadline)
                        .foregroundColor(.secondary) +
                    Text(timerInterval: countdownTimer, countsDown: true)
                        .font(.subheadline)
                        .bold()
                        .foregroundColor(.orange) +
                    Text(" mins")
                        .font(.subheadline)
                        .bold()
                        .foregroundColor(.orange)
```

```
    } else {
        Text(orderSubtitle)
            .font(.subheadline)
            .foregroundColor(.secondary)
    }
    if orderStatus > 0 {
        ProgressView(
            timerInterval: countdownTimer.lowerBound...countdownTimer.upperBound,
            countsDown: false,
            label: { EmptyView() },
            currentValueLabel: { EmptyView() }
        )
        .progressViewStyle(LinearProgressViewStyle(tint: .orange))
    }
}
.frame(maxWidth: .infinity, alignment: .leading)
VStack {
    //let nameAttribute = context.state.dynamicAttributes
    if imageOnTop {
        Image("app-logo")
            .resizable()
            .scaledToFit()
            .frame(width: 50, height: 50)
       Text(context.state.dynamicAttributes?["customerName"]?.value as? String ?? "CK_K")
            .font(.subheadline)
            .bold()
            .foregroundColor(.primary)
            .padding(.bottom, 5)
    } else {
        Text(context.state.dynamicAttributes?["customerName"]?.value as? String ?? "CK_K")
            .font(.subheadline)
            .bold()
            .foregroundColor(.primary)
            .padding(.bottom, 5)
        Image("app-logo")
            .resizable()
            .scaledToFit()
            .frame(width: 50, height: 50)
    }
```

```
        }
      }
      .padding()
      .cornerRadius(12)
      .onAppear {
        orderStatus = Int(getEstimatedTimeRemaining() / 60)
        updateTitleTextColor()
      }
      .onChange(of: orderStatus) { newValue in
        updateTitleTextColor()
      }
      .activityBackgroundTint(.black)
  }

  // Update the title text color dynamically
  private func updateTitleTextColor() {
    titleTextColor = context.attributes.mainAttributes["titleColor"]?.value as? String ?? "#9c27b0" //
Default
  }

  // TODO: - (@chhavi krishan) - TBD (how to use dynamic attributes estimated time. both approaches
achieve similar outcomes)
  // This function retrieves the estimated time remaining by iterating over dynamic attributes.
  private func getEstimatedTimeRemaining() -> Int {

    var estimatedTimeRemaining: Int = 0
    if let attributes = context.state.dynamicAttributes {
      for (_, value) in attributes {
        if let timeValue = value.value as? Int {
          estimatedTimeRemaining = timeValue
          break
        }
      }
    }
    return estimatedTimeRemaining
  }
}
```

Sample App for LiveActivity using Swift UI:

ActivityKitApp.swift

```
import SwiftUI
import UIKit
import ActivityKit
@main
struct ActivityKitApp: App {

   @UIApplicationDelegateAdaptor(AppDelegate.self) var appDelegate
   var body: some Scene {
      WindowGroup {
         ContentView()
      }
   }
}
class AppDelegate: NSObject, UIApplicationDelegate{

   //Get token to start live activity with push notification.
   func getPushToStartToken() {
      // Check if the iOS version is at least 17.2, which is required for this functionality
      if #available(iOS 17.2, *) {
         Task {
            for await data in Activity<DeliveryAttributes>.pushToStartTokenUpdates {
               // Convert data to string representation
               let token = data.map {String(format: "%02x", $0)}.joined()
                     print("Activity PushToStart Token: \(token)")
                     //TODO: - (@chhavi krishan) pass token to putAppAndUserData - TBD
               }
         }
      }
   }

   func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool {
      // appice Keys
      let nameArr = ["assets/isrgrootx1.der"]
```

```
        appICE.setupKeys(
          "b8a0da985627dc6933e156c69f9e90017925f8ad",
          withapiKey: "ddfc05dd1585e10ab73cf4f0b0b220ec",
          withappId:"5e4bce9a3d400534d7dc67a7",
          otherSdkdeviceId:"",
          region:"",
          baseUrl:"",
          certficates: nameArr
        )
        appICE.sharedInstance().startContext()
         //getPushToStartToken()
        return true
    }
}
```

ContentView.swift

```
import SwiftUI
import ActivityKit
import os
struct ContentView: View {

    @State private var token: String = ""
    let  activityManager = ActivityManager()
    var body: some View {
        VStack {
            Text("PushToken: \(token)")
                .padding()
            Button("Start") {
                // Payload data
                let payload: [String: Any] = [
                    "aps": [
                        "timestamp": 1728120486,
                        "event": "starrt",
                        "content-state": [
                            "dynamicAttributes": [
                                "customerName": "Karl.C",
                                "imageOnTop": false,
                                "serviceName": "Rice",
                                "orderTitle": "Your grocery order received",
                                "estimatedTimeRemaining": 800 //Seconds
                            ]
```

```
                ],
                "alert": [
                  "title": "Race Ended",
                  "body": "Tony Stark has finished the race!"
                ],
                "delivery-attributes": [
                  "itemTitle": "Rice Package",
                  "numberOfItems": 3,
                  "totalAmount": "$15.00",
                  "orderNumber": "ORD12345",
                  "deliveryType": "Express",
                  "titleColor": "#ff5722" //Color hex
                ]
              ]
            ]
            activityManager.startActivity(from: payload, completion: { newToken in
              DispatchQueue.main.async {
                token = newToken
              }
              print("PushToken: \(token)")
              os_log("PushToken: \(token)")
            })
          }
          Spacer(minLength: 50)
        }
        .padding()
    }
}
```

ActivityContent.swift

```
import ActivityKit
import os
class ActivityManager {
    private var deliveryActivity: Activity<DeliveryAttributes>?
    func startActivity(
        mainAttributes: [String: AnyCodable], // Main attributes for the activity
        contentStateAttributes: [String: AnyCodable], // Initial content state attributes for the activity.
        completion: @escaping (String) -> Void // Completion handler to return the push token
```

```
    ) {
        print("startActivity() is running")
        os_log("startActivity() is running")
        let attributes = DeliveryAttributes(
            mainAttributes: mainAttributes
        )
        let initialContentState = DeliveryAttributes.ContentState(
            dynamicAttributes: contentStateAttributes
        )
        let activityContent = ActivityContent(state: initialContentState, staleDate: nil)
        do {
            self.deliveryActivity = try Activity.request(attributes: attributes, content: activityContent,
pushType: .token)
            print("Requested a delivery Live Activity \(String(describing: self.deliveryActivity?.id)).")
            os_log("Requested a delivery Live Activity \(String(describing: self.deliveryActivity?.id)).")
            Task {
                for await data in self.deliveryActivity!.pushTokenUpdates {
                    let myToken = data.map { String(format: "%02x", $0) }.joined()
                    print("Token printed: \(myToken)")
                    os_log("Token Printed: \(myToken)")
                    completion(myToken)
                }
            }
        } catch let error {
            print("Error requesting delivery Live Activity \(error.localizedDescription).")
        }
    }
}
extension ActivityManager {

    func startActivity(from payload: [String: Any], completion: @escaping (String) -> Void) {
        // Extract necessary information from the payload dictionary.
        guard let aps = payload["aps"] as? [String: Any],
              let deliveryAttributes = aps["delivery-attributes"] as? [String: Any],
              let contentState = aps["content-state"] as? [String: Any],
              let dynamicAttributes = contentState["dynamicAttributes"] as? [String: Any] else {
            print("Payload failed!")
            return
        }
        // Convert delivery attributes to [String: AnyCodable] type
        let mainAttributes: [String: AnyCodable] = deliveryAttributes.reduce(into: [String: AnyCodable]()) {
result, entry in
```

```
            result[entry.key] = AnyCodable(entry.value)
        }
        // Convert dynamic attributes to [String: AnyCodable] type
        let contentStateAttributes: [String: AnyCodable] = dynamicAttributes.reduce(into: [String:
AnyCodable]()) { result, entry in
            result[entry.key] = AnyCodable(entry.value)
        }
        // Call startActivity with the extracted attributes and content state.
        startActivity(mainAttributes: mainAttributes, contentStateAttributes: contentStateAttributes,
completion: completion)
    }
}
```

ActivityAttributes.swift

```
import Foundation
#if canImport(ActivityKit)
import ActivityKit
#endif
// Protocol representing a codable type that can hold any value.
// Any type conforming to this protocol must be Codable, Equatable, and Hashable.
protocol AnyCodableProtocol: Codable, Equatable, Hashable {
    var value: Any { get } // stored value, which can be of any data type (e.g., Int, String, Bool)
    init<T>(_ value: T) // Initializer for storing any value.
}
// Struct that conforms to AnyCodableProtocol to store any value and encode/decode it.
struct AnyCodable: AnyCodableProtocol {
    var value: Any // stored value of any type.
    // Generic initializer that accepts a value of any type.
    init<T>(_ value: T) {
        self.value = value
    }
    // MARK: - Codable Conformance
    // Custom initializer to decode the value from a decoder.
    init(from decoder: Decoder) throws {
        let container = try decoder.singleValueContainer()
        if let intValue = try? container.decode(Int.self) {
            self.value = intValue // Decode as Int if possible.
        } else if let doubleValue = try? container.decode(Double.self) {
            self.value = doubleValue // Decode as Double if possible.
        } else if let stringValue = try? container.decode(String.self) {
```

```
            self.value = stringValue // Decode as String if possible.
        } else if let boolValue = try? container.decode(Bool.self) {
            self.value = boolValue // Decode as Bool if possible.
        } else {
            // Throw an error if the type is unsupported.
            throw DecodingError.dataCorruptedError(in: container, debugDescription: "Unsupported type")
        }
    }
    // Custom method to encode the value to an encoder.
    func encode(to encoder: Encoder) throws {
        var container = encoder.singleValueContainer()
        switch value {
        case let intValue as Int:
            try container.encode(intValue) // Encode as Int if value is Int.
        case let doubleValue as Double:
            try container.encode(doubleValue) // Encode as Double if value is Double.
        case let stringValue as String:
            try container.encode(stringValue) // Encode as String if value is String.
        case let boolValue as Bool:
            try container.encode(boolValue) // Encode as Bool if value is Bool.
        default:
            // Throw an error if the type is unsupported.
            throw EncodingError.invalidValue(value, EncodingError.Context(codingPath:
encoder.codingPath, debugDescription: "Unsupported type"))
        }
    }
    // MARK: - Equatable Conformance
    // Equatable conformance to compare two AnyCodable instances.
    static func == (lhs: AnyCodable, rhs: AnyCodable) -> Bool {
        return String(describing: lhs.value) == String(describing: rhs.value)
    }
    // MARK: - Hashable Conformance
    // Hashable conformance to hash the value.
    func hash(into hasher: inout Hasher) {
        hasher.combine(String(describing: value))
    }
}
// MARK: - DeliveryAttributes
// DeliveryAttributes representing the attributes of a delivery activity. conforming to
ActivityAttributes.
struct DeliveryAttributes: ActivityAttributes {
```

```
    // Main attributes of the delivery, represented as a dictionary with AnyCodable values.
    var mainAttributes: [String: AnyCodable]

    // ContentState representing the content state, which may contain dynamic attributes.
    struct ContentState: Codable, Hashable {

        // Optional dictionary containing dynamic attributes of the content state.
        var dynamicAttributes: [String: AnyCodable]?

        // Coding keys used for encoding and decoding dynamic attributes.
        enum CodingKeys: String, CodingKey {
            case dynamicAttributes
        }
    }
}
```

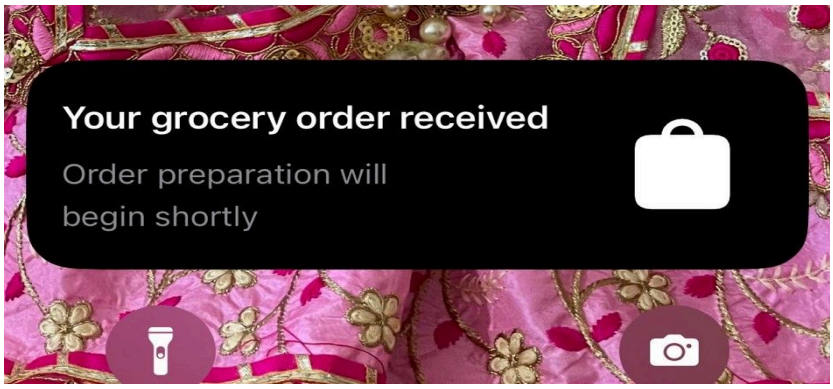
SAMPLE PAYLOAD:


```
{
  "aps": {
    "timestamp": 1728986081,
    "event": "update",
    "content-state": {
      "dynamicAttributes": {
        "customerName": "Ridway",
"imageOnTop": true,
        "serviceName": "Rice",
 "orderTitle": "Your grocery order delivered",
 "estimatedTimeRemaining": 1700
      }
    },
    "alert": {
      "title": "Race Ended",
      "body": "Tony Stark has finished the race!"
    },
"attributes-type": "DeliveryAttributes",
    "delivery-attributes": {
      "itemTitle": "Rice Package1",
      "numberOfItems": 4,
      "totalAmount": "151.00",
      "orderNumber": "ORD12345",
      "deliveryType": "Express"
    }
```

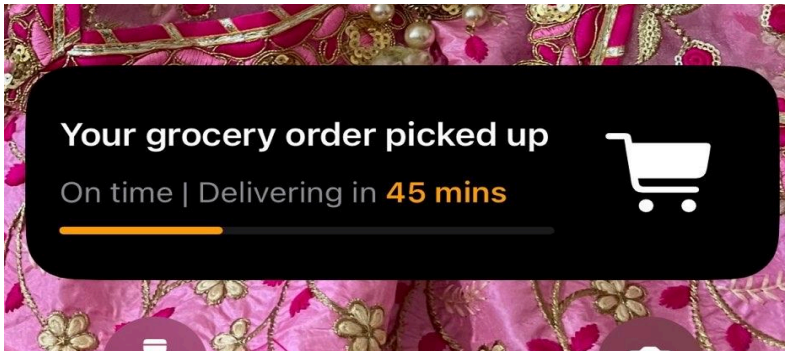
```
  }
}
```

Live Activity Order Funnel Movement

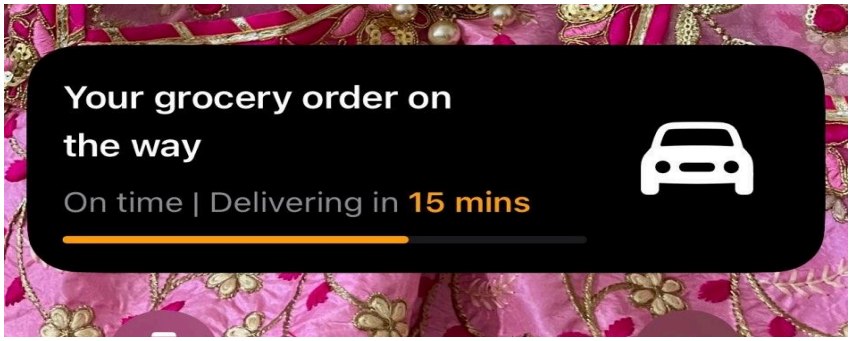
LiveActivity Step 1 : Order confirmation



LiveActivity Step 2 : Order picked by agent



LiveActivity Step 3 : Out for delivery



LiveActivity Step 4 : Delivery confirmation