Sample Code

Food Truck: Building a SwiftUI multiplatform app

Create a single codebase and app target for Mac, iPad, and iPhone.

Download

iOS 16.4+ | iPadOS 16.4+ | macOS 13.3+ | Xcode 14.3+

Overview

Using the Food Truck app, someone who operates a food truck can keep track of orders, discover the most-popular menu items, and check the weather at their destination. The sample implements the new `NavigationSplitView` to manage the app's views, `Layout` to show the main interface and pending orders, Swift Charts to show trends, and `WeatherService` to get weather data. Food Truck also implements Live Activities to show the remaining order preparation time with ActivityKit on the lock screen, and with `DynamicIsland` on the home screen.

You can access the source code for this sample on GitHub.

> Note
>
> This sample code project is associated with WWDC22 session 110492: State of the Union.

The Food Truck sample project contains two types of app targets:

- Simple app target you can build using personal team signing. This app runs in Simulator, and only requires a standard Apple ID to install on a device. It includes in-app purchase, and a widget extension that enable users to add a widget to their iOS Home Screen or the macOS Notification Center.

- Full-featured Food Truck All app target. The full app runs in Simulator, and on devices with an Apple Developer membership. It also allows you to create and sign in with passkeys.

Configure the sample code project

To configure the Food Truck app without an Apple Developer account, follow these steps:

1. In the Food Truck target's Signing & Capabilities panes click Add Account, and log in with your Apple ID.
2. Chose Your Name (Personal Team) from the team menu for the Food Truck and Widgets targets.
3. Build and run your app.
4. On iOS and iPadOS devices navigate to Settings > General > VPN & Device Management and trust your developer certificate.

To configure the Food Truck All app to run on your devices, follow these steps:

1. Open the sample with Xcode 14.3 or later.
2. Select the top-level Food Truck project.
3. For all targets, choose your team from the Team menu in the Signing & Capabilities pane, so Xcode can automatically manage your provisioning profile.
4. Add the Associated Domains capability, and specify your domain with the `webcredentials` service. For more information about the `webcredentials` service, see `Associated Domains Entitlement`.
5. Ensure an `apple-app-site-association` (AASA) file is present on your domain, in the `.well-known` directory, and it contains an entry for this app's App ID for the `webcredentials` service. For more information about the `apple-app-site-association` file, see Supporting associated domains.
6. In the `AccountManager.swift` file, replace all occurrences of `example.com` with the name of your domain.

> **Note**
>
> To use the weather forecast feature in the sample, you need to perform additional steps to configure WeatherKit, as described in the Configure the project for WeatherKit section below, or the sample will detect an error and use static data included in the project.

Create a multiplatform app

Food Truck is a multiplatform app, and there are no separate targets to run on macOS or iOS. Instead, there is only one app target that builds for macOS, iPadOS, and iOS.

Define a default navigation destination

The sample's navigation interface consists of a `NavigationSplitView` with a `Sidebar` view, and a `NavigationStack`:

```
NavigationSplitView {
    Sidebar(selection: $selection)
} detail: {
    NavigationStack(path: $path) {
        DetailColumn(selection: $selection, model: model)
    }
}
```

At app launch, the sample presents the `TruckView` as the default view. The `Panel` enum encodes the views the user can select in the sidebar, and hence appear in the detail view. The value corresponding to `TruckView` is `.truck`, and the app sets this to be the default selection.

```
@State private var selection: Panel? = Panel.truck
```

Construct a dynamic layout

In the Truck view, the New Orders panel shows the five most-recent orders, and each order shows a `DonutStackView`, which is a diagonal stack of donut thumbnails. The `Layout` protocol allows the app to define a `DiagonalDonutStackLayout` that arranges the donut thumbnails into the diagonal layout. The layout's `placeSubviews(in:proposal:subviews:cache:)` implementation calculates the donuts' positions.

```
for index in subviews.indices {
    switch (index, subviews.count) {
    case (_, 1):
        subviews[index].place(
            at: center,
            anchor: .center,
            proposal: ProposedViewSize(size)
        )

    case (_, 2):
        let direction = index == 0 ? -1.0 : 1.0
        let offsetX = minBound * direction * 0.15
        let offsetY = minBound * direction * 0.20
        subviews[index].place(
            at: CGPoint(x: center.x + offsetX, y: center.y + offsetY),
            anchor: .center,
            proposal: ProposedViewSize(CGSize(width: size.width * 0.7, height: size.
        )
    case (1, 3):
        subviews[index].place(
            at: center,
            anchor: .center,
            proposal: ProposedViewSize(CGSize(width: size.width * 0.65, height: size
        )

    case (_, 3):
        let direction = index == 0 ? -1.0 : 1.0
        let offsetX = minBound * direction * 0.15
        let offsetY = minBound * direction * 0.23
        subviews[index].place(
            at: CGPoint(x: center.x + offsetX, y: center.y + offsetY),
            anchor: .center,
            proposal: ProposedViewSize(CGSize(width: size.width * 0.7, height: size.
        )
```

Display a chart of popular items

The sample contains several charts. The most popular items are shown on the `TopFiveDonutsView`. This chart is implemented in `TopDonutSalesChart`, which uses a `BarMark` to construct a bar chart.

```
Chart {
    ForEach(sortedSales) { sale in
        BarMark(
            x: .value("Donut", sale.donut.name),
            y: .value("Sales", sale.sales)
        )
        .cornerRadius(6, style: .continuous)
        .foregroundStyle(.linearGradient(colors: [Color("BarBottomColor"), .accentCo
        .annotation(position: .top, alignment: .top) {
            Text(sale.sales.formatted())
                .padding(.vertical, 4)
```

```
                    .padding(.horizontal, 8)
                    .background(.quaternary.opacity(0.5), in: Capsule())
                    .background(in: Capsule())
                    .font(.caption)
            }
        }
    }
```

The *x* axis of the chart shows labels with the names and thumbnails of the items that correspond to each data point.

```
.chartXAxis {
    AxisMarks { value in
        AxisValueLabel {
            let donut = donutFromAxisValue(for: value)
            VStack {
                DonutView(donut: donut)
                    .frame(height: 35)

                Text(donut.name)
                    .lineLimit(2, reservesSpace: true)
                    .multilineTextAlignment(.center)
            }
            .frame(idealWidth: 80)
            .padding(.horizontal, 4)

        }
    }
}
```

Obtain a weather forecast

The app shows a forecasted temperature graph in the Forecast panel in the Truck view. The app obtains this data from the WeatherKit framework.

```
.task(id: city.id) {
    for parkingSpot in city.parkingSpots {
        do {
            let weather = try await WeatherService.shared.weather(for: parkingSpot.l
            condition = weather.currentWeather.condition
            willRainSoon = weather.minuteForecast?.contains(where: { $0.precipitatic
            cloudCover = weather.currentWeather.cloudCover
            temperature = weather.currentWeather.temperature
            symbolName = weather.currentWeather.symbolName

            let attribution = try await WeatherService.shared.attribution
            attributionLink = attribution.legalPageURL
            attributionLogo = colorScheme == .light ? attribution.combinedMarkLightU

            if willRainSoon == false {
                spot = parkingSpot
                break
            }
        } catch {
            print("Could not gather weather information...", error.localizedDescript
            condition = .clear
            willRainSoon = false
            cloudCover = 0.15
        }
    }
}
```

Configure the project for WeatherKit

The data from the `WeatherService` instance in WeatherKit requires additional configuration for the Food Truck All target. If you don't configure WeatherKit, the sample will detect an error and use static data in the project instead.

1. Create a unique App ID on the Provisioning Portal, and select the WeatherKit service on the App Services tab.

2. In Xcode, for the Food Truck All target on the Signing & Capabilities tab, change the bundle ID to be the same as the App ID from step 1, and add the WeatherKit capability.

3. For the Widgets target on the Signing & Capabilities tab, change the bundle ID to make the part before `.Widgets` the same as the bundle ID for the Food Truck All target.

4. Wait 30 minutes while the service registers your app's bundle ID.

5. Build and run the Food Truck All target.

Track preparation time with Live Activity

The app allows the food truck operator to keep track of order preparation time, which is guaranteed to be 60 seconds or less. To facilitate this, the app implements a toolbar button on the order details screen for orders with `placed` status. Tapping this button changes the order status to `preparing`, and creates an `Activity` instance to start a Live Activity, which shows the countdown timer and order details on an iPhone lock screen.

```
let timerSeconds = 60
let activityAttributes = TruckActivityAttributes(
    orderID: String(order.id.dropFirst(6)),
    order: order.donuts.map(\.id),
    sales: order.sales,
    activityName: "Order preparation activity."
)

let future = Date(timeIntervalSinceNow: Double(timerSeconds))

let initialContentState = TruckActivityAttributes.ContentState(timerRange: Date.now.

let activityContent = ActivityContent(state: initialContentState, staleDate: Calenda

do {
    let myActivity = try Activity<TruckActivityAttributes>.request(attributes: activ
        pushType: nil)
    print(" Requested MyActivity live activity. ID: \(myActivity.id)")
    postNotification()
} catch let error {
    print("Error requesting live activity: \(error.localizedDescription)")
}
```

The app also implements `DynamicIsland` to show the same information as on the lock screen in the Dynamic Island on iPhone 14 Pro devices.

```
DynamicIsland {
    DynamicIslandExpandedRegion(.leading) {
        ExpandedLeadingView()
    }

    DynamicIslandExpandedRegion(.trailing, priority: 1) {
        ExpandedTrailingView(orderNumber: context.attributes.orderID, timerInterval:
            .dynamicIsland(verticalPlacement: .belowIfTooWide)
    }
} compactLeading: {
    Image("IslandCompactIcon")
        .padding(4)
        .background(.indigo.gradient, in: ContainerRelativeShape())

} compactTrailing: {
    Text(timerInterval: context.state.timerRange, countsDown: true)
        .monospacedDigit()
        .foregroundColor(Color("LightIndigo"))
        .frame(width: 40)
} minimal: {
    Image("IslandCompactIcon")
        .padding(4)
        .background(.indigo.gradient, in: ContainerRelativeShape())
}
.contentMargins(.trailing, 32, for: .expanded)
.contentMargins([.leading, .top, .bottom], 6, for: .compactLeading)
.contentMargins(.all, 6, for: .minimal)
.widgetURL(URL(string: "foodtruck://order/\(context.attributes.orderID)"))
```

Tapping the same button again changes the status to `complete`, and ends the Live Activity. This removes the Live Activity from the lock screen and from the Dynamic Island.

```
Task {
    for activity in Activity<TruckActivityAttributes>.activities {
        // Check if this is the activity associated with this order.
        if activity.attributes.orderID == String(order.id.dropFirst(6)) {
            await activity.end(nil, dismissalPolicy: .immediate)
        }
    }
}
```

See Also

Creating an app

{} Destination Video

Leverage SwiftUI to build an immersive media experience in a multiplatform app.

{} Hello World

Use windows, volumes, and immersive spaces to teach people about the Earth.

{} Backyard Birds: Building an app with SwiftData and widgets

Create an app with persistent data, interactive widgets, and an all new in-app purchase experience.

{} Fruta: Building a feature-rich app with SwiftUI

Create a shared codebase to build a multiplatform app that offers widgets and an App Clip.

Migrating to the SwiftUI life cycle

Use a scene-based life cycle in SwiftUI while keeping your existing codebase.

`protocol App`

A type that represents the structure and behavior of an app.