SAP TechEd 2021 - ARKit Demo Steps

INTRODUCTION OF THE FIORIARKIT REPO

1. Show the GitHub repository https://github.com/SAP/cloud-sdk-ios-fioriarkit
2. Mention requirements of **iOS 14 or higher, Xcode 12 or higher**

GET YOUR REALITY FILE READY USING REALITY COMPOSER BY APPLE

In this section we introduce the idea behind using Reality Composer to create a reality file scene which can be used in Xcode in order to define the anchor image for reality kit to display the Fiori ARKit Annotation Cards.

This part of the demo is just a walk through out of time reasons.

1. Open Reality Composer with the **TechEd.rcproject** file
2. Explain the placed spheres and the anchor image
3. The anchor image helps Reality Kit to recognise the element in the real world. Point out that this also can be a 3D object which can be scanned through Reality Composer on iPad or iPhone.
4. Show rcproject file in Xcode project

INTRODUCTION OF THE XCODE PROJECT

This part of the demo will explain the basic setup of the Xcode project. The project will initially contain the following parts:

- Swift Package Dependencies of **FioriSwiftUI**
- The Assets Group including **TechEd.rcproject, Anchor Image (TechEd2021_Header_small)**
- The Utils Group including **FileManager+Extensions.swift** file for loading the rcproject files and extracting the scene.

1. Open the Xcode project
2. Explain that the FioriARKit is included in the FioriSwiftUI package. We add that as Swift Package Dependency.
3. Introduce the Assets Group
4. Introduce the Utils Group with the FileManager extension

IMPLEMENT THE CARD MODELS AND SAMPLE DATA
Before starting to implement the custom Card Views and the actual initialisation of the **reality** file and the **Content View**, I go into detail about the model conforming to the **CardItemModel** protocol.

This makes implementation of the model very easy as the protocol already gives us the template struct.

The Sample Data is there out of time reasons to not have to implement a backend connection. The sample data is an **Enum** providing an array of **Card Items.**

1. Create File **StringIdentifyingCardItem.swift** file and implement the CardItemModel protocol

```
import SwiftUI
import FioriARKit

public struct StringIdentifyingCardItem: CardItemModel {
    public var id: String
    public var title_: String
    public var descriptionText_: String?
    public var detailImage_: Image?
    public var actionText_: String?
    public var icon_: Image?

    public init(id: String, title_: String, descriptionText_: String? = nil,
detailImage_: Image? = nil, actionText_: String? = nil, icon_: Image? = nil) {
        self.id = id
        self.title_ = title_
        self.descriptionText_ = descriptionText_
        self.detailImage_ = detailImage_
        self.actionText_ = actionText_
        self.icon_ = icon_
    }
}
```

2. Create File **TechEdInfo.swift** file and implement the Enum with a static array of **StringIdentifyingCardItem** structs.
3. The array contains of 3 items: Welcome, Schedule, Profile
    1. These represent their IDs for later distinguishing the items to be displayed so we can pick the correct custom card view
    2. The icons are placed on the spheres and I am using system images from the SF Symbols library.

```
   public static let techEdCardItems = [ StringIdentifyingCardItem(id:
“Welcome",  title_: "Welcome to SAP TechEd 2021”, descriptionText_: Date().description,
detailImage_: Image(“TechEd2021_Header_small"), actionText_: nil, icon_: Image(systemName:
“play")), StringIdentifyingCardItem(id: “Schedule", title_: "Your upcoming sessions”,
descriptionText_: "SAP Fiori for Champs! /n 1:20pm”, detailImage_: nil, actionText_: nil, icon_:
Image(systemName: “calendar")), StringIdentifyingCardItem(id: “Profile", title_: “",
descriptionText_: “", detailImage_: Image(“profile"), actionText_: nil, icon_: Image(systemName:
"person.circle"))]
```

4. Point out that models can also be provided using JSON

CUSTOM CARD VIEWS - IMPLEMENTATION
There are different ways of implementing Card Views. The use of pre-defined cards is the easiest where the View Builder approach gives you the most flexibility. As I want to show the modularity and flexibility of the APIs, I am using the View Builder approach using SwiftUI

1. Create the CardViews group
2. Create the WelcomeCardView.swift SwiftUI file
    1. Make it generic <CardItem: CardItemModel>
    2. Add the model property
    3. Create a ZStack containing the image (which is a link) and a VStack containing the welcoming text elements
    4. Give the ZStack a frame and a corner radius

```
import SwiftUI
import FioriARKit

struct ScheduleCardView<CardItem: CardItemModel>: View {
        var model: CardItem
        var isSelected: Bool
        @State var color: Color = .white

        var body: some View {
            ZStack {
                Link(destination: URL(string: "https://www.sap.com")!) {
                    Image("teched_background")
                        .resizable()
                        .scaledToFit()
                }
                VStack(alignment: .leading) {
                    Text("Next Up")
                        .bold()
                        .foregroundColor(.white)
                    HStack {
                        Text("Cloud Native for Dummys:")
                            .font(.subheadline)
                            .foregroundColor(.white)
                        Text("\(Date().addingTimeInterval(600), style: .timer)")
                            .font(.subheadline)
                            .foregroundColor(.white)
                    }
                    Text("Speaker: John Doe")
                        .font(.footnote)
                        .bold()
                        .foregroundColor(.white)
                }
            }
            .frame(width: 245, height: 138)
            .cornerRadius(20)
        }
}
```

3. Create the ProfileCardView following the pattern of the WelcomeCardView

```
struct ProfileCardView<CardItem: CardItemModel>: View {
    var model: CardItem
    var isSelected: Bool
    @State var color: Color = .white

    var body: some View {
        ZStack {
            Link(destination: URL(string: "www.https://people.sap.com/kevin.muessig")!)
{
                Image("teched_background")
                    .resizable()
                    .scaledToFit()
            }
            HStack {
                Image("profile")
                    .resizable()
                    .frame(width: 100, height: 100, alignment: .center)
                    .mask(Circle())
                    .shadow(radius: 10)
                VStack {
                    Text("Kevin Muessig")
                        .font(.subheadline)
                        .bold()
                        .foregroundColor(.white)
                    Text("Developer Advocate")
                        .font(.footnote)
                        .foregroundColor(.white)
                }
            }
        }
        .frame(width: 245, height: 138)
        .background(color)
        .cornerRadius(20)
    }
}
```

4. Create the ScheduleCardView following the pattern of the WelcomeCardView

```
struct ScheduleCardView<CardItem: CardItemModel>: View {
        var model: CardItem
        var isSelected: Bool
        @State var color: Color = .white

        var body: some View {
            ZStack {
                Link(destination: URL(string: "https://www.sap.com")!) {
                    Image("teched_background")
                        .resizable()
                        .scaledToFit()
                }
                VStack(alignment: .leading) {
                    Text("Next Up")
                        .bold()
                        .foregroundColor(.white)
                    HStack {
                        Text("Cloud Native for Dummys:")
                            .font(.subheadline)
                            .foregroundColor(.white)
                        Text("\(Date().addingTimeInterval(600), style: .timer)")
                            .font(.subheadline)
                            .foregroundColor(.white)
                    }
                    Text("Speaker: John Doe")
                        .font(.footnote)
                        .bold()
                        .foregroundColor(.white)
                }
            }
            .frame(width: 245, height: 138)
            .cornerRadius(20)
        }
}
```

CREATE THE CONTENTVIEW FOR DISPLAYING THE CARDVIEWS
The content view will hold the code for defining and setting the **ARModel,** it also defines the container view for the CardItems which gets distinguished using the set ID. On appearance of the view we load the initial data.
The initial data is containing the card items (TechEdInfo.techEdCardItems), the anchor image from the assets and the loading strategy.

Point out the different types of loading strategy supporting the:

- USDZ Strategy: Requires a URL path to the .usdz file
- Reality Strategy: Requires a URL path to the .reality file and the name of the scene
- RCProject Strategy: Requires the name of the .rcproject file and the name of the scene

The RCProject strategy requires that the .rcproject file is part of the application bundle so that the file is available already during build time.

```
import SwiftUI
import FioriARKit

struct CustomARCardsContentView: View {
    @StateObject var arModel =
ARAnnotationViewModel<StringIdentifyingCardItem>()

    var body: some View {
        SingleImageARCardView(arModel: arModel,
                              image: Image("TechEd2021_Header_small"),
                              cardLabel: { cardmodel, isSelected in
                                switch cardmodel.id {
                                case "Welcome": WelcomeCardView(model:
cardmodel, isSelected: isSelected)
                                case "Profile": ProfileCardView(model:
cardmodel, isSelected: isSelected)
                                case "Schedule": ScheduleCardView(model:
cardmodel, isSelected: isSelected)
                                default:
                                    WelcomeCardView(model: cardmodel,
isSelected: isSelected)
                                }
                              })
            .onAppear(perform: loadInitialData)
    }

    func loadInitialData() {
        let cardItems = TechEdInfo.techEdCardItems
        guard let anchorImage = UIImage(named: "TechEd2021_Header_small") else {
return }

        let strategy = RCProjectStrategy(cardContents: cardItems, anchorImage:
anchorImage, physicalWidth: 0.1, rcFile: "TechEd", rcScene: "TechEd_Logo_Scene")
        arModel.load(loadingStrategy: strategy)
    }
}
```

CHANGE THE APPDELEGATE IN ORDER TO LOAD THE REALITY FILE AND DISPLAY THE CONTENT VIEW

The AppDelegate is the initial entry point of the app and the place where we load the reality file during runtime. The reality file got added to the runtime because we added it to the bundle. We use the FileManager extension to load the reality file from the bundle, extract the Reality Scene from it and safe the scene to the App Sandbox. From there it can be used with the Fiori ARKit APIs.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        let realityDir =
FileManager.default.makeDirectoryInDocumentsDirectory(FileManager.realityFiles)

        guard let extractRealityURL = Foundation
                .Bundle(for: TechEd.TechEdLogoScene.self)
                .url(forResource: "TechEd", withExtension: "reality"),
              let rcRealityData = try? Data(contentsOf: extractRealityURL)
        else { return true }

        let saveRealityURL = realityDir.appendingPathComponent("TechEd.reality")

        FileManager.default.saveDataToDirectory(saveRealityURL, saveData:
rcRealityData)

        // Create the SwiftUI view that provides the window contents.
        let contentView = CustomARCardsContentView()
//        let contentView = ContentView()

        // Use a UIHostingController as window root view controller.
        let window = UIWindow(frame: UIScreen.main.bounds)
        window.rootViewController = UIHostingController(rootView: contentView)
        self.window = window
        window.makeKeyAndVisible()
        return true
    }
```