**LOGIN**

**ContentView.swift**

Este archivo es la vista principal de la app en **SwiftUI**. Sirve como **punto de entrada** para determinar qué vista se mostrará dependiendo del estado de autenticación del usuario.

 **@EnvironmentObject** → Se usa para acceder a una instancia de PostViewModel, que maneja el estado de autenticación del usuario.

 login.authenticated es una variable en PostViewModel que indica el estado de autenticación del usuario.

**Group** → Se usa para agrupar múltiples vistas dentro de un solo cuerpo de View, ayudando a mejorar la organización.

 Cuando el usuario abre la app, ContentView evalúa login.authenticated:

* 0 → Muestra la pantalla de **Login**.
* 1 → Muestra la pantalla **Home** (cuando el usuario ha iniciado sesión correctamente).
* 2 → Muestra un mensaje de error con un botón para intentar de nuevo.

 **Interacción con PostViewModel**:

* PostViewModel es un @EnvironmentObject que gestiona el estado de autenticación.
* Cuando el usuario inicia sesión correctamente, authenticated cambia a 1 y se muestra Home().
* Si la autenticación falla, authenticated cambia a 2, y se muestra un mensaje de error.

**SWIFTUI**

**Explicación Paso a Paso**

1. **@main** → Marca esta estructura como el **punto de entrada** de la aplicación en SwiftUI.
2. **struct SwiftUI07App: App** → Define la estructura principal de la aplicación.
3. **var body: some Scene** → Indica que esta app tendrá una **escena principal**.
4. **let login = PostViewModel()** → Se crea una instancia del **ViewModel** encargado de la autenticación.
5. **WindowGroup { ContentView() }** → Define que ContentView será la primera vista que verá el usuario.
6. **.environmentObject(login)** → Se pasa login como un **@EnvironmentObject**, permitiendo que otras vistas accedan a este objeto sin necesidad de pasarlo manualmente.

**🚀 ¿Qué hace este código?**

✔ Configura la estructura de la aplicación con SwiftUI.  
✔ Crea una instancia de PostViewModel para gestionar la autenticación.  
✔ Inicia la aplicación mostrando ContentView().  
✔ Permite que cualquier vista acceda a PostViewModel a través de @EnvironmentObject.

📌 **¿Quieres que revisemos PostViewModel.swift para entender cómo funciona la autenticación?**

**MODEL**

**MODELO 1**

**📌 Explicación Paso a Paso**

1. **struct Modelo1: Decodable** → Define un modelo de usuario que **puede ser decodificado** desde un JSON.
2. **Propiedades de Modelo1**:
   * id → Identificador único del usuario.
   * name → Nombre del usuario.
   * email → Dirección de correo.
   * address → Objeto Address, que representa la dirección del usuario.
3. **Estructura Address**:
   * Contiene información de la calle, suite, ciudad y coordenadas (Geo).
4. **Estructura Geo**:
   * Contiene latitud (lat) y longitud (lng) en formato String.
5. **Decodable**:
   * Permite que esta estructura sea usada para convertir datos JSON a objetos de Swift automáticamente.

**MODELO 2**

**📌 Explicación Paso a Paso**

1. **struct Modelo2** → Representa la respuesta JSON con una **lista de usuarios** en data.
2. **struct User** → Representa la respuesta JSON cuando se obtiene **un solo usuario** en data.
3. **struct UserList** → Define la estructura de un usuario con los datos:
   * id → ID único del usuario.
   * first\_name → Nombre del usuario.
   * email → Dirección de correo electrónico.
   * avatar → URL de la imagen de perfil.

**POSTMODEL**

**Explicación**

1. **struct PostModel: Decodable** → Define una estructura que puede **decodificar** datos JSON en Swift.
2. **var token: String** → Representa el **token de autenticación** devuelto por el servidor tras un inicio de sesión exitoso.

**VIEW**

**DETALLEVIEW**

**Explicación Paso a Paso**

1. **Recibe un id como parámetro**, que representa el identificador del usuario que se mostrará en detalle.
2. **Crea una instancia de DetalleViewModel** con @StateObject, permitiendo que los datos de usuario se mantengan en la vista.
3. **Muestra un ProgressView() si la imagen del avatar aún no está disponible** (indicando que los datos aún se están cargando).
4. **Cuando los datos están listos**, muestra:
   * La imagen del usuario con AsyncImage.
   * El nombre (first\_name).
   * El correo (email).
5. **Cuando la vista aparece (onAppear())**, llama a user.fetch(id: id), lo que hace una petición a la API para obtener los datos del usuario.

### ****Cómo obtiene los datos**** DetalleViewModel****?****

DetalleViewModel.fetch(id:) probablemente hace una solicitud HTTP para obtener el usuario desde una API.

**HOMESWIFT**

**Explicación Paso a Paso**

1. **@EnvironmentObject var login: PostViewModel**
   * Permite acceder a PostViewModel, que maneja la autenticación del usuario.
   * Se usa para **cerrar sesión** y redirigir al usuario a la pantalla de inicio de sesión.
2. **@StateObject var json = Modelo1ViewModel()**
   * Modelo1ViewModel obtiene datos de una API y los almacena en datosModelo.
   * Se usa @StateObject para que los datos se mantengan en la vista.
3. **NavigationStack**
   * Permite **navegar** entre diferentes vistas.
4. **Carga de datos con if-else**
   * Si json.datosModelo **está vacío**, muestra ProgressView() para indicar que los datos se están cargando.
   * Si los datos ya están listos, muestra una **lista de usuarios** (List).
5. **Lista de usuarios con List**
   * Itera sobre json.datosModelo y muestra:
     + item.name → Nombre del usuario.
     + item.email → Correo del usuario.
     + item.address.geo.lat → Latitud de su ubicación.
     + item.address.geo.lng → Longitud de su ubicación.
6. **Botón "Salir" (Cerrar Sesión)**
   * Borra la sesión guardada en UserDefaults.
   * Cambia login.authenticated = 0 para volver a la pantalla de inicio de sesión.
7. **Botón "Siguiente"**
   * Usa NavigationLink(destination: Home2()) para navegar a la vista Home2.

**📌 Ejemplo de Uso**

Cuando el usuario **inicia sesión correctamente**, Home.swift: ✔ **Carga los datos de la API usando Modelo1ViewModel**.  
✔ **Muestra una lista de usuarios con su nombre, email y coordenadas**.  
✔ **Permite cerrar sesión con el botón "Salir"**.  
✔ **Permite navegar a Home2 con el botón "Siguiente"**.

**HOME 2**

**Explicación Paso a Paso**

1. **@StateObject var json2 = Modelo2ViewModel()**
   * Modelo2ViewModel obtiene datos de la API y los almacena en datosModelo.
   * Se usa @StateObject para que los datos **se mantengan actualizados en la vista**.
2. **Carga de datos con if-else**
   * Si json2.datosModelo.data **está vacío**, muestra ProgressView() mientras se obtienen los datos de la API.
   * Si los datos están listos, muestra una **lista de usuarios** (List).
3. **Lista de usuarios con List**
   * Itera sobre json2.datosModelo.data y muestra:
     + **Imagen del usuario (avatar)** con AsyncImage.
     + **Nombre del usuario (first\_name)** con font(.title).
     + **Correo electrónico (email)** con font(.subheadline).
4. **Cargar imágenes de forma asíncrona (AsyncImage)**
   * Descarga la imagen de perfil desde item.avatar.
   * Mientras la imagen carga, muestra un ProgressView().
   * La imagen se recorta con clipShape(Circle()) para darle un estilo circular.
5. **Navegación a DetalleView**
   * NavigationLink(destination: DetalleView(id: item.id))
     + Permite que, al tocar un usuario en la lista, la app **muestre su información detallada en DetalleView**.
6. **Título de la barra de navegación (.navigationBarTitle())**
   * Se establece "Json con imagen" como título en la parte superior de la pantalla.

**📌 Ejemplo de Uso**

Cuando el usuario **entra a Home2.swift**, la vista: ✔ **Carga datos de una API usando Modelo2ViewModel**.  
✔ **Muestra una lista de usuarios con su nombre, email e imagen de perfil**.  
✔ **Permite navegar a DetalleView para ver más información del usuario seleccionado**.

**LOGIN**

**📌 Explicación Paso a Paso**

1. **@State var email: String = ""** → Guarda el valor del email ingresado en el campo de texto.
2. **@State var password: String = ""** → Guarda la contraseña ingresada en el campo de texto.
3. **@EnvironmentObject var login: PostViewModel** → Accede a PostViewModel, que maneja la autenticación.
4. **ZStack con Color.blue.edgesIgnoringSafeArea(.all)**
   * Establece **un fondo azul** que cubre toda la pantalla.
5. **Formulario de inicio de sesión (VStack)**
   * **Campo de email (TextField)**
     + Tiene fondo blanco y usa un teclado optimizado para direcciones de correo.
   * **Campo de contraseña (SecureField)**
     + Oculta los caracteres ingresados.
   * **Botón "Entrar"**
     + Cuando se presiona, llama a login.login(email: password:), enviando las credenciales a PostViewModel.

**📌 ¿Cómo funciona este código en la app?**

1. El usuario ingresa su **email y contraseña**.
2. Cuando presiona "Entrar", se llama a login.login(email: password:) en PostViewModel.
3. PostViewModel se encarga de **verificar la autenticación** con la API.
4. Si las credenciales son correctas, cambia authenticated = 1, mostrando la pantalla Home().

**VIEWMODEL**

**DETALLEVIEWMODEL**

**Explicación Paso a Paso**

1. **class DetalleViewModel: ObservableObject**
   * Define un **ViewModel** que sigue el patrón **MVVM**.
   * ObservableObject permite que las vistas en SwiftUI **se actualicen automáticamente** cuando cambian los datos.
2. **Variables @Published**
   * @Published var first\_name = "" → Guarda el **nombre** del usuario y actualiza la vista automáticamente.
   * @Published var email = "" → Guarda el **correo** del usuario.
   * @Published var avatar = "" → Guarda la **URL de la imagen** del usuario.
3. **Función fetch(id:)**
   * Obtiene los datos de un usuario desde la API **ReqRes** usando su id.
4. **Construcción de la URL**

swift

CopiarEditar

guard let url = URL(string: "https://reqres.in/api/users/\(id)") else { return }

* + Convierte la URL en un objeto URL.
  + Si la URL no es válida, la función termina (return).

1. **Petición HTTP con URLSession**

swift

CopiarEditar

URLSession.shared.dataTask(with: url) { (data, response, error) in

* + Hace una petición a la API en **segundo plano**.

1. **Verificación de datos**

swift

CopiarEditar

guard let data = data else { return }

* + Si no hay datos en la respuesta, la función se detiene.

1. **Decodificación del JSON**

swift

CopiarEditar

let json = try JSONDecoder().decode(User.self, from: data)

* + Convierte los datos JSON en un objeto User.

1. **Actualización de la UI**

swift

CopiarEditar

DispatchQueue.main.async {

self.first\_name = json.data.first\_name

self.email = json.data.email

self.avatar = json.data.avatar

}

* + **Actualiza la vista en el hilo principal** para que SwiftUI muestre los datos en DetalleView.

1. **Manejo de errores**

swift

CopiarEditar

} catch let error as NSError {

print("Error en el JSON: \(error.localizedDescription)")

}

* + Si la decodificación del JSON falla, **imprime un mensaje de error en la consola**.

1. **Inicio de la tarea**

swift

CopiarEditar

}.resume()

* Inicia la tarea de red para obtener los datos del usuario.

**MODELO1VIEWMODEL**

1. **class Modelo1ViewModel: ObservableObject**
   * Define un **ViewModel** en SwiftUI, que sigue el patrón **MVVM**.
   * ObservableObject permite que la vista **se actualice automáticamente** cuando cambian los datos.
2. **@Published var datosModelo: [Modelo1] = []**
   * @Published permite que SwiftUI **actualice la vista automáticamente** cuando datosModelo cambie.
   * datosModelo almacena **una lista de usuarios** obtenidos de la API.
3. **init() { fetch() }**
   * Al crear una instancia de Modelo1ViewModel, llama automáticamente a fetch(), iniciando la obtención de datos.
4. **fetch()**
   * Obtiene una lista de usuarios desde la API **JSONPlaceholder**.
5. **Construcción de la URL**

swift

CopiarEditar

guard let url = URL(string: "https://jsonplaceholder.typicode.com/users") else { return }

* + Convierte la URL en un objeto URL.
  + Si la URL no es válida, la función se detiene (return).

1. **Petición HTTP con URLSession**

swift

CopiarEditar

URLSession.shared.dataTask(with: url) { (data, response, error) in

* + Hace una petición **en segundo plano** a la API.

1. **Verificación de datos**

swift

CopiarEditar

guard let data = data else { return }

* + Si no hay datos en la respuesta, la función se detiene.

1. **Decodificación del JSON**

swift

CopiarEditar

let json = try JSONDecoder().decode([Modelo1].self, from: data)

* + Convierte los datos JSON en un **array de Modelo1**.

1. **Actualización de la UI**

swift

CopiarEditar

DispatchQueue.main.async {

self.datosModelo = json

}

* + **Actualiza la vista en el hilo principal**, permitiendo que SwiftUI muestre los datos en la interfaz.

1. **Manejo de errores**

swift

CopiarEditar

} catch let error as NSError {

print("Error en el JSON: \(error.localizedDescription)")

}

* Si la decodificación del JSON falla, **imprime un mensaje de error en la consola**.

1. **Inicio de la tarea**

swift

CopiarEditar

}.resume()

* **Ejecuta la petición a la API**, permitiendo obtener los datos.

**MODELO2VIEWMODEL**

**Explicación Paso a Paso**

1. **class Modelo2ViewModel: ObservableObject**
   * Define un **ViewModel** que sigue el patrón **MVVM**.
   * ObservableObject permite que SwiftUI **actualice la vista automáticamente** cuando cambian los datos.
2. **@Published var datosModelo = Modelo2(data: [])**
   * @Published permite que SwiftUI **actualice la vista automáticamente** cuando datosModelo cambie.
   * datosModelo almacena la lista de usuarios obtenidos de la API.
3. **init() { fetch() }**
   * Al crear una instancia de Modelo2ViewModel, llama automáticamente a fetch(), iniciando la obtención de datos.
4. **fetch()**
   * Obtiene una lista de usuarios desde la API **ReqRes** en la página 2.
5. **Construcción de la URL**

swift

CopiarEditar

guard let url = URL(string: "https://reqres.in/api/users?page=2") else { return }

* + Convierte la URL en un objeto URL.
  + Si la URL no es válida, la función se detiene (return).

1. **Petición HTTP con URLSession**

swift

CopiarEditar

URLSession.shared.dataTask(with: url) { (data, response, error) in

* + Hace una petición **en segundo plano** a la API.

1. **Verificación de datos**

swift

CopiarEditar

guard let data = data else { return }

* + Si no hay datos en la respuesta, la función se detiene.

1. **Decodificación del JSON**

swift

CopiarEditar

let json = try JSONDecoder().decode(Modelo2.self, from: data)

* + Convierte los datos JSON en un objeto **Modelo2**, que contiene la lista de usuarios.

1. **Actualización de la UI**

swift

CopiarEditar

DispatchQueue.main.async {

self.datosModelo = json

}

* + **Actualiza la vista en el hilo principal**, permitiendo que SwiftUI muestre los datos en la interfaz.

1. **Manejo de errores**

swift

CopiarEditar

} catch let error as NSError {

print("Error en el JSON: \(error.localizedDescription)")

}

* Si la decodificación del JSON falla, **imprime un mensaje de error en la consola**.

1. **Inicio de la tarea**

swift

CopiarEditar

}.resume()

* **Ejecuta la petición a la API**, permitiendo obtener los datos.

**POSTVIEWMODEL**

**Explicación Paso a Paso**

1. **class PostViewModel: ObservableObject**
   * Define un **ViewModel** que maneja el login del usuario.
   * ObservableObject permite que SwiftUI **se actualice automáticamente** cuando cambia el estado de autenticación.
2. **@Published var authenticated = 0**
   * 0 → El usuario **no ha iniciado sesión**.
   * 1 → El usuario **ha iniciado sesión correctamente**.
   * 2 → Hubo un **error en la autenticación**.
3. **Verificación de sesión en init()**
   * Intenta recuperar el estado de la sesión almacenada en UserDefaults.
   * Si hay una sesión guardada (1), el usuario sigue autenticado.
   * Si no hay sesión guardada, el usuario debe volver a iniciar sesión.
4. **Función login(email: password:)**
   * **Construye la URL** https://reqres.in/api/login (API de ReqRes para autenticación).
   * **Convierte los parámetros (email, password) en JSON**.
   * **Configura la petición HTTP** con método POST.
   * **Ejecuta la solicitud HTTP** usando URLSession.
5. **Manejo de la respuesta**
   * Si la API responde correctamente y devuelve un token, se cambia authenticated = 1 y se guarda la sesión en UserDefaults.
   * Si la API devuelve un error (credenciales incorrectas), se cambia authenticated = 2.

**📌 Ejemplo de JSON de Respuesta**

Si la autenticación es exitosa, la API responde con:

json

CopiarEditar

{

"token": "QpwL5tke4Pnpja7X4"

}

Si la autenticación falla, devuelve:

json

CopiarEditar

**EXPLICACIÓN COMPLETA**

## **1. Estructura del Proyecto**

📂 **SwiftUI07App.swift**  
📂 **Views/** (Interfaz de Usuario)  
📂 **ViewModels/** (Lógica de Negocio)  
📂 **Models/** (Estructuras de Datos)

Cada carpeta tiene una función específica en el patrón MVVM:

* **Models (Modelos de Datos)** → Representa la estructura de la información.
* **ViewModels (Lógica de Negocio)** → Maneja la obtención y procesamiento de los datos.
* **Views (Interfaz de Usuario)** → Muestra los datos en pantalla y permite la interacción del usuario.

## 🔹 **2. Descripción de los Componentes**

A continuación, te explico cada parte del código con su función dentro del proyecto.

### 📌 ****2.1 SwiftUI07App.swift (Punto de Entrada)****

📂 **SwiftUI07App.swift**  
🔹 Es el archivo principal que inicia la aplicación.  
🔹 Configura PostViewModel() como @EnvironmentObject, permitiendo que toda la app acceda a la autenticación.

🔹 **Código clave**:

swift

CopiarEditar

@main

struct SwiftUI07App: App {

var body: some Scene {

let login = PostViewModel() // Maneja la autenticación del usuario.

WindowGroup {

ContentView().environmentObject(login) // Se pasa el ViewModel a la vista principal.

}

}

}

### 📌 ****2.2 Views (Interfaz de Usuario)****

📂 **Views/**  
Las vistas se encargan de la presentación de la información y la interacción del usuario.

#### 📌 **🔹 ContentView.swift (Vista Principal)**

* Muestra **Login()**, **Home()** o un mensaje de error según el estado de authenticated.
* Usa @EnvironmentObject var login: PostViewModel para determinar qué vista mostrar.

#### 📌 **🔹 Login.swift (Pantalla de Login)**

* Permite al usuario ingresar su **email** y **contraseña**.
* Llama a login.login(email: password:) para autenticar.
* Si el login es correcto, cambia authenticated = 1.

#### 📌 **🔹 Home.swift (Lista de Usuarios - Modelo1)**

* Muestra una lista de usuarios obtenidos de Modelo1ViewModel().
* Permite cerrar sesión eliminando la sesión guardada.

#### 📌 **🔹 Home2.swift (Lista de Usuarios con Imagen - Modelo2)**

* Muestra una lista de usuarios con imagen usando Modelo2ViewModel().
* Usa NavigationLink para acceder a DetalleView.

#### 📌 **🔹 DetalleView.swift (Vista de Detalle de Usuario)**

* Muestra información específica de un usuario (first\_name, email, avatar).
* Llama a DetalleViewModel().fetch(id: id) para obtener los datos de un usuario.

### 📌 ****2.3 ViewModels (Lógica de Negocio)****

📂 **ViewModels/**  
Los **ViewModels** manejan la obtención y actualización de datos en la aplicación.

#### 📌 **🔹 PostViewModel.swift (Manejo de Autenticación)**

* Se encarga de **iniciar sesión** enviando credenciales a la API https://reqres.in/api/login.
* Guarda el estado de autenticación en UserDefaults.

🔹 **Flujo de Autenticación**:

1. El usuario ingresa email y contraseña en Login.swift.
2. Llama a PostViewModel().login(email: password:).
3. Si la API devuelve un token, cambia authenticated = 1.
4. Si hay un error, cambia authenticated = 2.

#### 📌 **🔹 Modelo1ViewModel.swift (Obtiene Lista de Usuarios - Modelo1)**

* Obtiene datos desde https://jsonplaceholder.typicode.com/users.
* Guarda la información en datosModelo para ser usada en Home.swift.

#### 📌 **🔹 Modelo2ViewModel.swift (Obtiene Lista de Usuarios con Imagen - Modelo2)**

* Obtiene datos desde https://reqres.in/api/users?page=2.
* Guarda la información en datosModelo para ser usada en Home2.swift.

#### 📌 **🔹 DetalleViewModel.swift (Obtiene Datos de un Usuario Específico)**

* Obtiene información de un usuario desde https://reqres.in/api/users/{id}.
* Se usa en DetalleView.swift.

### 📌 ****2.4 Models (Estructuras de Datos)****

📂 **Models/**  
Los modelos definen la estructura de los datos recibidos desde la API.

#### 📌 **🔹 PostModel.swift (Modelo para la Autenticación)**

* Representa la respuesta de la API cuando el usuario inicia sesión.

swift

CopiarEditar

struct PostModel: Decodable {

var token: String // Si `token` está presente, el login es exitoso.

}

#### 📌 **🔹 Modelo1.swift (Modelo para Usuarios en Home)**

* Representa un usuario con nombre, email y dirección.

#### 📌 **🔹 Modelo2.swift (Modelo para Usuarios con Imagen en Home2)**

* Representa un usuario con nombre, email y avatar.

## 🔹 **3. Flujo de la Aplicación**

A continuación, te explico cómo fluye la aplicación cuando un usuario la usa.

### ✅ ****Inicio de la App****

1. SwiftUI07App.swift carga ContentView().
2. ContentView verifica si el usuario está autenticado (authenticated):
   * 0 → Muestra Login.swift.
   * 1 → Muestra Home.swift.
   * 2 → Muestra mensaje de error.

### ✅ ****Inicio de Sesión****

1. El usuario ingresa email y contraseña en Login.swift.
2. Se llama a PostViewModel().login(email: password:).
3. Si la autenticación es correcta, cambia authenticated = 1.
4. ContentView muestra Home.swift.

### ✅ ****Visualización de Datos****

1. Home.swift usa Modelo1ViewModel().fetch() para obtener usuarios.
2. El usuario puede navegar a Home2.swift, donde se usan imágenes (Modelo2ViewModel().fetch()).
3. Al seleccionar un usuario, se abre DetalleView.swift con DetalleViewModel().fetch(id:).

### ✅ ****Cerrar Sesión****

1. En Home.swift, el usuario presiona "Salir".
2. Se elimina la sesión en UserDefaults.
3. authenticated cambia a 0 y la app regresa a Login.swift.

## 🔹 **4. ¿Qué Hace Bien Este Proyecto?**

✔ **Estructura bien organizada en MVVM.**  
✔ **Uso de @Published y @EnvironmentObject para actualizar la UI dinámicamente.**  
✔ **Uso de URLSession para hacer llamadas a APIs reales.**  
✔ **Autenticación con UserDefaults para recordar la sesión del usuario.**  
✔ **Carga de imágenes asíncronas con AsyncImage.**

## 🔹 **5. Posibles Mejoras**

🔥 **Manejar errores en la autenticación con alertas.**  
🔥 **Agregar validaciones para email y contraseña antes de enviar la petición.**  
🔥 **Usar @AppStorage en lugar de UserDefaults.standard.setValue().**  
🔥 **Mejorar la presentación de DetalleView.swift con más información.**  
🔥 **Añadir un botón para actualizar los datos manualmente.**

### 🚀 ****Conclusión****

Este proyecto está bien estructurado y sigue **MVVM correctamente**.  
Usa llamadas a API reales para **autenticación y carga de datos**.  
Tiene una **navegación fluida** y maneja la sesión del usuario con UserDefaults.

📌 **¿Quieres que implementemos algunas mejoras?** 🔥