MOV02 - Lab 12

Juan Pablo Arias Mora

Marzo 2021

1 Enunciado

"Las personas talentosas trabajando juntas se pulen los unos a los otros, pulen ideas, y lo que sale son piedras preciosas.." Steve Jobs

Hay una serie de tecnologías diferentes que se pueden utilizar cuando se necesitan almacenar datos en las aplicaciones. La mas sencilla vista en clase se denomina **UserDefaults**. La intención del siguiente laboratorio es presentar un uso de la misma ligado a la autenticación del usuario.

1.1 Versiones Permitidas

- MacOs 10.15.7 - Inglés/Español
- Xcode 12.4
- Simulador iPhone 12 Pro Max

1.2 Creación de aplicación base

Paso 1: Cree un nuevo proyecto en Xcode, utilizando el template App dentro de iOS, en la ventana de opciones para la configuración especifica de la nueva aplicación.

Product Name : userdefault
Organization Identifier : com.cenfotec.mov02.12
Interface : SwiftUI
Life Cycle : SwiftUI App
Languge : Swift
Use Core Data: Sin Seleccionar
Include Test: Sin Seleccionar

1.3 Persistencia utilizando UserDefaults

Paso 1: Abrir el archivo **ContentView.swift**. (El código en el archivo debe ser similar al que se muestra a continuación)

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
            .padding()
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Paso 2: SwiftUI tiene un **Property Wrapper** para leer valores de **UserDefaults**, que automáticamente invalida el **View** actual y actualiza el nuevo valor en el mismo. Es decir, este **Wrapper** observa una **Key** en **UserDefaults** y actualizará su **UI** si esa **key** cambia. Como primer vamos a remplazar el código dentro de **ContentView.swift**, con el código a continuación.

```
import SwiftUI

struct ContentView: View {
    @AppStorage("username") var username: String = "Anonymous"

        var body: some View {
            VStack {
                if let user = username{
                    Text("Welcome, \(user)")
                }
                Button("Log in") {
                    username = "John Doe"
                }
            }
        }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Paso 3: Compilamos y ejecutamos el código, una vez el simulador contenga la aplicación corriendo, notamos como el mensaje de bienvenida a **Anonymous**, pero si presionamos el botón **Login In**, y veremos como se cambia el valor del mensaje de bienvenida utilizando el nuevo valor presente en **username** (**John Doe**). Cambiar **username** causa que el nuevo **String** se escriba en **UserDefaults** inmediatamente, al mismo tiempo que se actualiza el **View**.

Paso 4: Realicemos un cambio en la asignación del nombre **John Doe** y ingrese su nombre.

```
username = "Juan Pablo"
```

Paso 5: Compilamos y ejecutamos el código, una vez el simulador contenga la aplicación corriendo, notamos como el mensaje de bienvenida continua siendo referente a **John Doe**, dejando de lado el valor de **Anonymous**, esto se debe a que el valor quedo almacenado dentro de los **UserDefaults** y estos son cargados al inicio de la ejecución de la aplicación. (ver figura 1)

Paso 6: Presionamos el botón **Login In**, y veremos como se cambia el valor del mensaje de bienvenida utilizando el nuevo valor presente en **username** (**Juan Pablo**). Se debe prestar la suficiente atención al comportamiento que se presento, ya que la aplicación fue compilada nuevamente mostrando el cambio ejecutado en el paso anterior, pero el archivo **UserDefaults** se mantiene presente, mientras la aplicación siga instalada dentro del teléfono.

Paso 7: Dentro del simulador presionamos el botón de **Home**. (ver figura 2 área roja). Esto nos llevara a la pantalla de inicio del teléfono, algo similar a la figura 3.

Figure 1: Mensaje Bienvenida John Doe

Paso 8: Nos colocamos en la parte inferior de la pantalla y realizamos el **Gesture** para poder listar todas las aplicaciones corriendo, el cual consiste en **Drag** desde la parte inferior de la pantalla hasta cercano al centro de la misma. Esto listará las aplicaciones siendo ejecutadas actualmente dentro del teléfono, similar a la figura 4.

Figure 2: Icono Home en el simulador

Figure 3: Pantalla Home

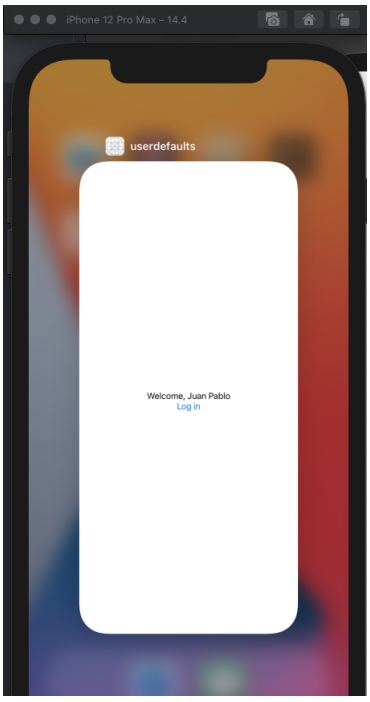

Figure 4: Aplicaciones siendo ejecutadas

Paso 9: Dentro de las lista nos aseguramos de terminar la ejecución de la aplicación **userdefaults**. Esto seleccionando la misma de la lista y desplazando el objeto hacia la parte superior de la pantalla.

Paso 10: Comprobamos que la misma no este siendo ejecutada, realizando el **Gesture** para poder listar todas las aplicaciones corriendo.

Paso 11: Una vez cerrada, vamos a proceder a reabrirla, esto se logra con simplemente dar **Click** sobre el icono de la aplicación

(ver figura 5), para de esta manera revisar que el ultimo valor asignado a **username** en los **UserDefaults** se mantenga.

Figure 5: Icono de Aplicación userdefault

Paso 12: Procedemos a desinstalar la aplicación, la manera mas sencilla es terminar la ejecución de la aplicación, nos posicionamos sobre el icono de la aplicación (ver figura 5), damos **Click** y mantenemos la presión unos segundos, esto debería mostrarnos un menú similar a la figura 6.

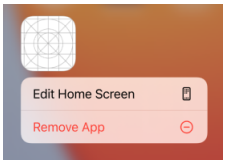

Figure 6: Icono de Aplicación listo para desinstalar

Paso 13: Presionamos la opción **Remove App**. En el mensaje de confirmación (ver figura 7) seleccionamos la opción **Delete App**.

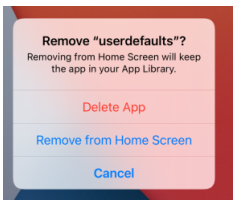

Figure 7: Mensaje de confirmación previo a desinstalar

Paso 14: Dándole sentido al segundo mensaje de confirmación, a este punto se va referir usualmente a los valores dentro de **UserDefaults**. (ver figura 8)

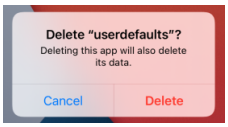

Figure 8: Segundo Mensaje de confirmación previo a desinstalar

Paso 15: Compilamos y ejecutamos el código, una vez el simulador contenga la aplicación corriendo, notamos como el mensaje de bienvenida vuelve a ser **Anonymous**. (ver figura 9)

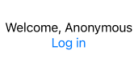

Figure 9: Mensaje Bienvenida Anonymous

Paso 16: Existe una forma de definir lo que se denomina un **App Group** para userDefaults. Puede utilizar este método al desarrollar un conjunto de aplicaciones, para compartir preferencias u otros datos entre las aplicaciones, o al desarrollar una extensión de aplicación, para compartir preferencias u otros datos entre la extensión y la aplicación que la contiene. Para más detalles puede consultar el siguiente enlace https://developer.apple.com/documentation/foundation/userdefaults/1409957-init. A nuestro nivel realizaremos el siguiente cambio en la definición de la variable **username**.

```
@AppStorage("username", store: UserDefaults(suiteName: "com.cenfotec.mov02.shared"))
    var username: String = "Anonymous"
```

Paso 17: Compilamos y ejecutamos el código, el funcionamiento de la aplicación debería mantenerse igual. **Importante**: **@AppStorage** escribe sus datos en **UserDefaults**, que no es un almacenamiento seguro. Como resultado, no debe guardar ningún dato personal con **@AppStorage**, porque es relativamente fácil de extraer, la única finalidad del laboratorio es explicar conceptos en la creación de aplicaciones básicas.

1.4 Simulación de Login usando UserDefaults

Paso 1: Vamos a nuevamente remplazar el código dentro de **ContentView.swift**, con el código a continuación.

```
import SwiftUI

struct ContentView: View {
    @AppStorage("status") var logged = false

    var body: some View {
        NavigationView{
            if logged{
                VStack{
                    Text ("User Logged In")
                        .navigationTitle("Home")
                        .navigationBarHidden(false)
                        .preferredColorScheme(.light)
                    Button(action:{
                        logged = false
                    }, label: {
                        Text("Logout")
                    })
                }
            }else{
                //Will add code later
                Text("Login Screen")
            }
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Paso 2: Procedemos a actualizar el **Preview**, el mismo debe ser similar a la figura 10.

Paso 3: Cambiaremos el valor de la variable **logged** de **false** a **true**.

```
@AppStorage("status") var logged = true
```

Paso 4: Procedemos a actualizar el **Preview**, el mismo debe ser similar a la figura 11. De esta manera podemos ver como dependiendo del valor de la variable **logged** depende la aparición de una u otra **View**

Paso 5: Cambiaremos el valor de la variable **logged** de **true** a **false**.

Paso 6: Crearemos un nuevo archivo de **SwiftUI View** con el nombre de **LoginView.swift**, dentro del mismo agregamos el siguiente código:

```
import SwiftUI

struct LoginView: View {
    @State var userName = ""
    @State var password = ""
```

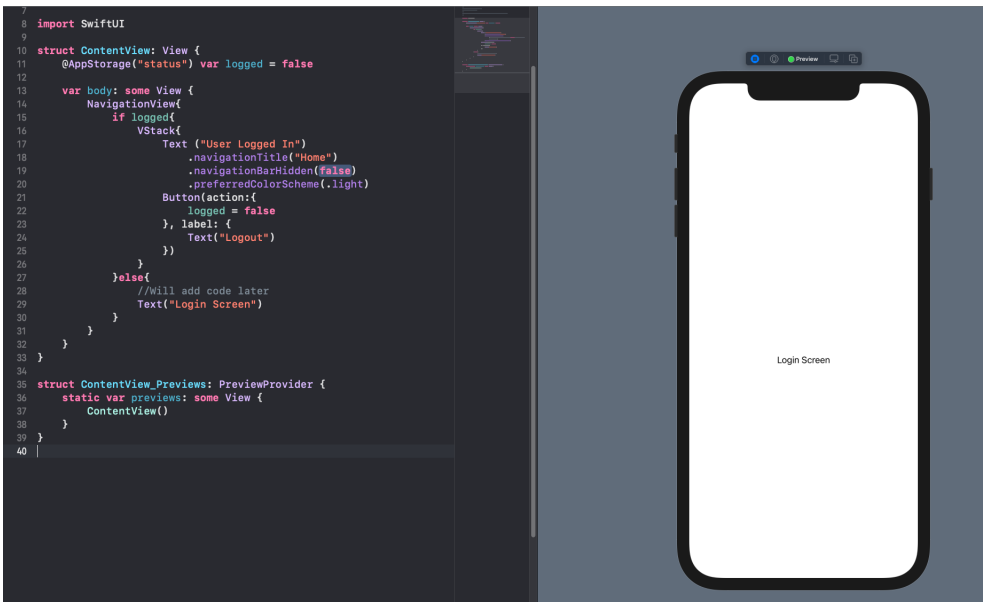

Figure 10: Mensaje de Login Screen en pantalla

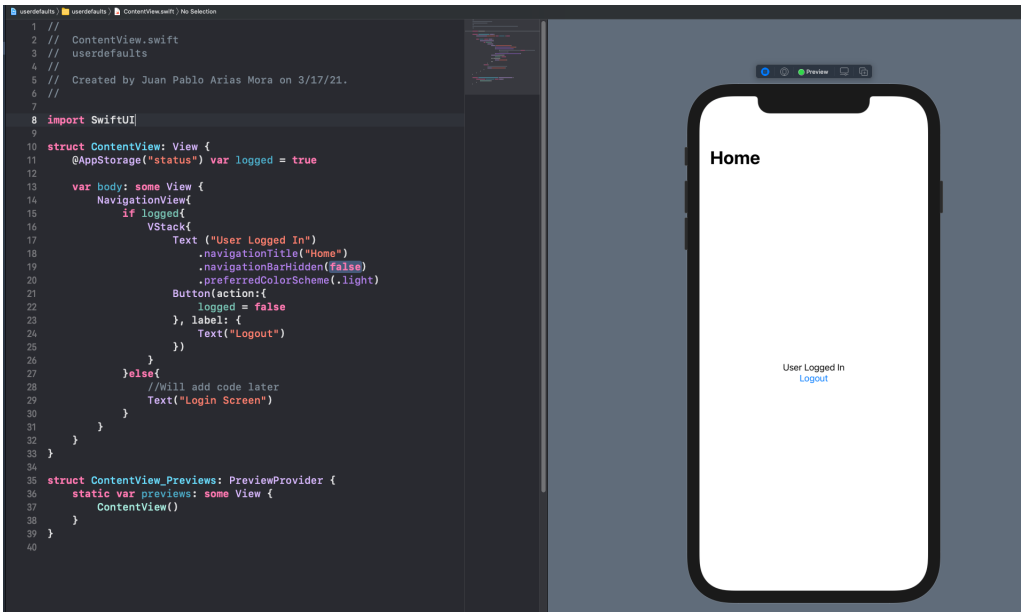

Figure 11: Mensaje de Usuario Login In en pantalla

```
@AppStorage("stored_User") var user = ""
@AppStorage("stored_Pass") var pass = ""
@AppStorage("status") var logged = false

var body: some View {
    VStack{
        Spacer(minLength: 0)
        Image(systemName: "person.icloud")
            .resizable()
            .aspectRatio(contentMode: .fit)
            .padding(.horizontal, 35)
            .padding(.vertical)
        HStack{
            VStack(alignment: .leading, spacing: 12, content: {
                Text("Login")
                    .font(.title)
```

```
                    .fontWeight(.bold)
                    .foregroundColor(.black)
                    //Check on the other view

                Text("Please sign in to continue")
                    .foregroundColor(Color.black.opacity(0.5))
            })
            Spacer(minLength: 0)

        }
        .padding(.leading,15)
        HStack{
            Image(systemName: "envelope")
                .font(.title2)
                .foregroundColor(.black)
                .frame(width: 35)
            TextField("EMAIL", text: $userName)
                .autocapitalization(.none)
        }
        .padding()
        .background(Color.white.opacity(userName == "" ? 0 : 0.12))
        .cornerRadius(15)
        .padding(.horizontal)

        HStack{
            Image(systemName: "lock")
                .font(.title2)
                .foregroundColor(.black)
                .frame(width: 35)
            SecureField("PASSWORD", text: $password)
                .autocapitalization(.none)
        }
        .padding()
        .background(Color.white.opacity(password == "" ? 0 : 0.12))
        .cornerRadius(15)
        .padding(.horizontal)
        .padding(.top)


        HStack(spacing: 15){
            Button(action: authenticateUserPassword, label:{
                Text("LOGIN")
                    .fontWeight(.heavy)
                    //.foregroundColor(.black)
                    .padding(.vertical)
                    .frame(width: UIScreen.main.bounds.width - 150)
                    //.background(Color("green"))
                    //.clipShape(Capsule())
            })
            .opacity(userName != "" && password != "" ? 1 : 0.5)
            .disabled(userName != "" && password != "" ? false: true )

        }
    }


}
func authenticateUserPassword(){
    if userName == user && password == pass {
        withAnimation(.easeOut){logged = true}
    }else{
```

```
                print("User Password Does not match")
            }
        }

    }

    struct LoginView_Previews: PreviewProvider {
        static var previews: some View {
            LoginView()
        }
    }
```

Paso 7: Utilizando el **Preview** debemos validar que el nuevo **LoginView** sea similar a la figura 14

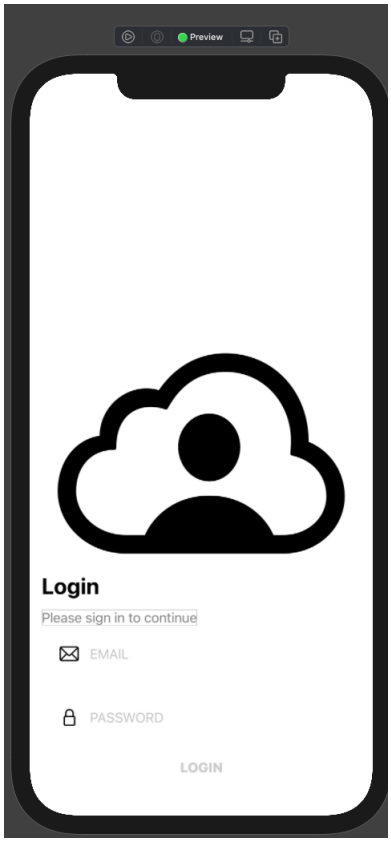

Figure 12: LoginView Preview

Paso 8: Nuevamente sobre el archivo **ContentView.swift**, reemplazamos el código:

```
Text("Login Screen")
```

Con el siguiente fragmento de código:

```
LoginView()
    .preferredColorScheme(.light)
    .navigationBarHidden(true)
```

Paso 9: Compilamos y ejecutamos el código, colocaremos el simulador al lado del código para proceder a analizar algunos efectos presentes en el UI en conjunto con las lineas de código que realizan la acción.

Paso 10: De primera entrada vemos dos **TextFields***, con el valor por defecto uno de **EMAIL** y otro de **PASSWORD**. Esto debido al siguiente código respectivamente:

```
TextField("EMAIL", text: $userName)
```

```
SecureField("PASSWORD", text: $password)
```

La diferencia entre **TextField** y **SecureField** en alto nivel residen en la capacidad de ocultar los valores que se ingresan sobre el segundo. Algo también que debemos mantener presente es la forma en que los valores ingresados se asocian con las variables por medio de la referencia de los mismos utilizando **$**

```
@State var userName = ""
@State var password = ""
```

Paso 11: Procedemos a ingresar un **Email**, si bien es cierto no tenemos de momento una validación de formato sobre el **Email**. Podemos observar como el color del **Font** cambia desde que ingresamos el primer carácter. (ver figura **??**

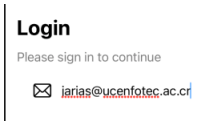

Figure 13: Email en UI

Este efecto se logra de la con el siguiente código:

```
HStack{...}
    .background(Color.white.opacity(userName == "" ? 0 : 0.12))
```

Paso 12: Procedemos a ingresar un **Password**. Podemos observar como el color del **Font** cambia desde que ingresamos el primer carácter, y en este caso por efecto del **SecureField** los caracteres estan ocultos. (ver figura **??**

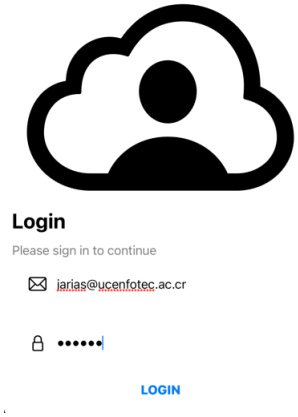

Figure 14: Password en UI

Este efecto se logra de la con el siguiente código:

```
HStack{...}
    .background(Color.white.opacity(password == "" ? 0 : 0.12))
```

Paso 13: Otro efecto interesante causado cuando se ingresan los valores de **Email** y **Password** es que el botón de **Login** se habilita y a su vez cambia de color. Este efecto se logra de la con el siguiente código:

```
Button(...)
    .opacity(userName != "" && password != "" ? 1 : 0.5)
    .disabled(userName != "" && password != "" ? false: true )
```

Paso 14: Podemos proceder a presionar el botón **Login** para ingresar a nuestra aplicación. Una vez presionado podemos observar que se nos muestra el siguiente mensaje en consola:

```
User Password Does not match
```

El mismo es generado por el siguiente código:

```
func authenticateUserPassword(){
    if userName == user && password == pass {
        withAnimation(.easeOut){logged = true}
    }else{
        print("User Password Does not match")
    }
}
```

Como se puede observar es a alto nivel, una simulación de una función de validación de valores, el mismo puede ser reemplazado luego por llamadas de API para conseguir los valores reales de un servicio de autenticación o manejo de contraseñas, pero nuevamente para efectos del laboratorio, lo manejamos de esta manera. Es importante agregar que el error mostrado se debe a que no inicializamos los valores, o más bien mantenemos como cadenas vacías los valores de

```
@AppStorage("stored_User") var user = ""
@AppStorage("stored_Pass") var pass = ""
```

Paso 15: Procedemos a cambiar los valores de cadenas vacías, por su **Email** y un **Password** temporal.

```
@AppStorage("stored_User") var user = "jarias@ucenfotec.ac.cr"
@AppStorage("stored_Pass") var pass = "123456789"
```

Paso 16: Compilamos y ejecutamos el código, ingresaremos los valores previamente definidos a nuestras variables, presionaremos el botón **Login**. En algunos casos la revisión de validación de los valores puede tardar, pero se debe a que la cuando se buscan valores en **UserDefaults** se realiza un barrido completo del archivo.

Paso 17: Vemos como nuestro segundo **View**, fue cargado al ingresar los valores correctos de **Email** y **Password**. Podemos dar **Click** sobre el botón de **Logout** para volver a la pantalla principal.

1.5 Agregando TouchID*/FaceID

Paso 1: Para poder hacer uso de la capacidad de Face Recognition, debemos de primera entrada, realizar una modificación en los permisos que posee nuestra aplicación. Esto se logra posicionándonos sobre el archivo padre de nuestro proyecto **userdefaults** (ver figura 15 morado), luego sobre la pestaña de **Info** (ver figura 15 rojo), de manera que localicemos una lista de **Keys** con sus respectivos valores. (ver figura 15 verde)

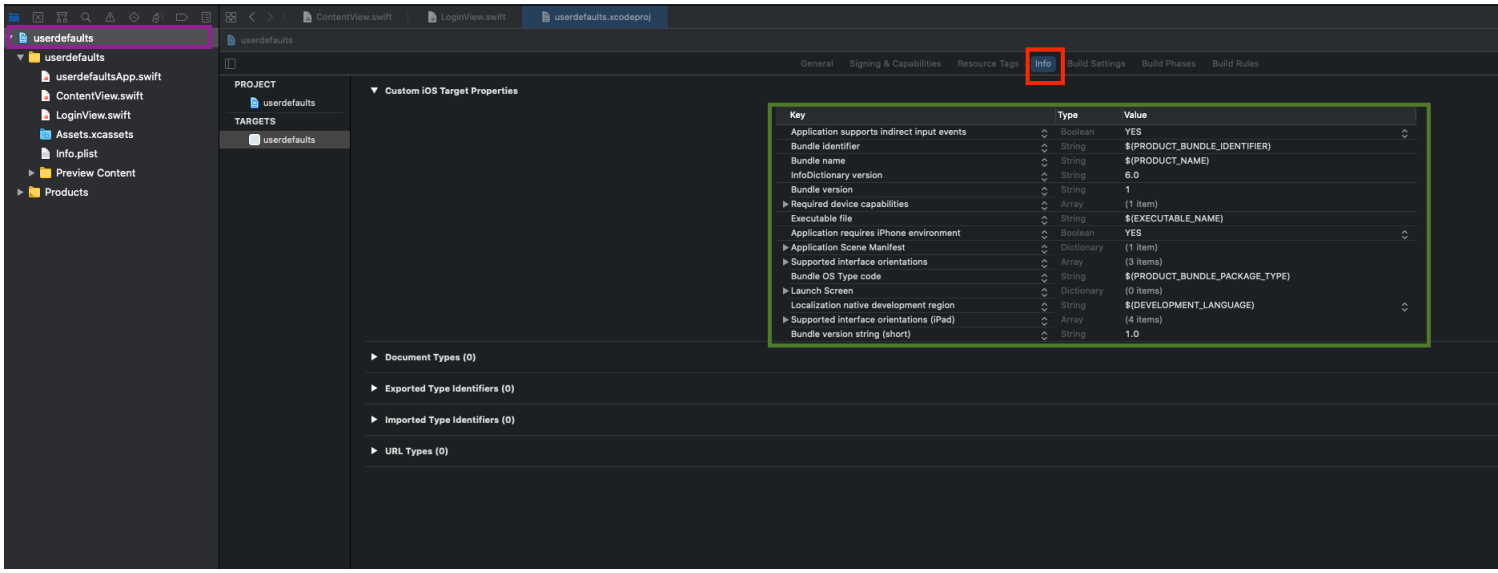

Figure 15: Info Keys de nuestro Target

Paso 2: Ingresamos un nuevo **Key** posicionándonos sobre un **Key** existente y le damos a la opción de +.(ver figura 16 morado)

Paso 3: Seleccionamos la opción de **Privacy - Face ID Usage Description** y le damos un valor de **This App uses FaceID to confirm your identity**.

Figure 16: Info Keys desplegar agregar Key

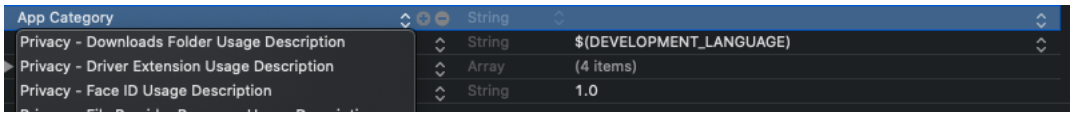

Figure 17: Nombre de Key predefinido a usar "Privacy - Face ID Usage Description"

Figure 18: Valor a Ingresar en el Key

Paso 4: A este nivel en realidad no agregamos permisos, simplemente mostramos un mensaje adaptado que cuando hagamos uso del **FaceID** por primera vez el usuario pueda entender para que queremos el uso del mismo. El set de permisos se ira otorgando cuando hagamos uso por primera vez de cada uno de ellos y esto a discreción del usuario.

Paso 5: Importamos la biblioteca encargada de manejo del **FaceID** a nivel de los **imports** dentro de **LoginView**

```
import LocalAuthentication
```

Paso 6: Agregamos el siguiente código quien es el encargado de revisar que el dispositivo tenga disponibilidad de **Biometrics**, justo después de la definición de **authenticateUserPassword**:

```
func getBiometricStatus()->Bool{
    let scanner = LAContext()
    // Biometry is available on the device
    if userName == user && scanner.canEvaluatePolicy(.deviceOwnerAuthentication, error: .none){
        return true
    }
    return false
}
```

Paso 7: También agregamos este otro código quien va a ser el encargado de gestionar realmente la validación del **FaceID**:

```
func authenticateUser(){
    let scanner = LAContext()
    scanner.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics,
                           localizedReason: "To Unlock \(userName)"){(status, error) in
        if error != nil{
            print("Error")
            print(error!.localizedDescription)
            return
        }
        withAnimation(.easeOut){logged = true}
    }
}
```

Este ultimo que agregamos es una versión realmente reducida de la validación a ejecutar, nuevamente para fines didácticos. Existen varias condiciones por validar dentro de la biométrica, principalmente en el manejo de errores.

Paso 8: Por ultimo debemos agregar el botón encargada de ejecutar el código de Biométrica. Para esto agregamos el siguiente codigo a dentro del ultimo **HStack**.

```
if getBiometricStatus(){
    Button(
        action: authenticateUser,
        label:{
            Image (systemName: LAContext().biometryType == .faceID ? "faceid" : "touchid")
```

```
                .font(.title)
                .foregroundColor(.black)
        }
    )
}
```

Paso 9: Compilamos y ejecutamos la aplicación.

Paso 10: Procedemos a ingresar el **Email** que tenemos definido como valido, y esta ves vemos como contiguo al botón de **Login**, tenemos un nuevo botón, con el icono de **FaceID**. (ver figura 19)

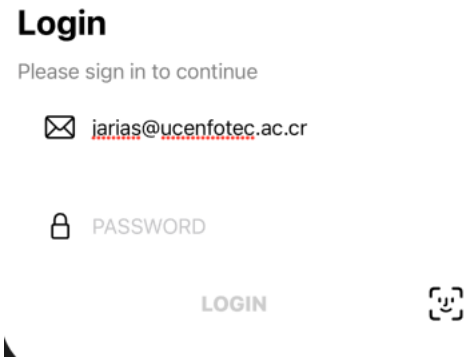

Figure 19: Icono de FaceID

Paso 11: Procedemos a presionar el botón de **FaceID**. En este momento se puede observar un error con el siguiente mensaje

`No identities are enrolled.`

El mismo se debe a que nuestro simulador, no tiene el **FaceID** marcado como **Enrolled**.

Paso 12: Vamos a la opción en la barra superior del simulador Features-Face ID-Enrrolled, y seleccionamos la opción. (ver figura 20)

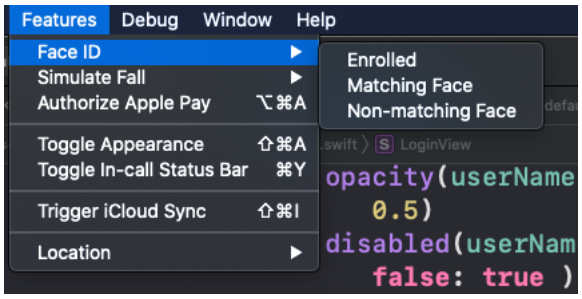

Figure 20: Enroll FaceID

Paso 13: Volvemos a presionar el icono de **FaceID**, vemos como ahora aparece el mensaje que definimos dentro de la **info** del proyecto en pantalla. (ver figura 21)

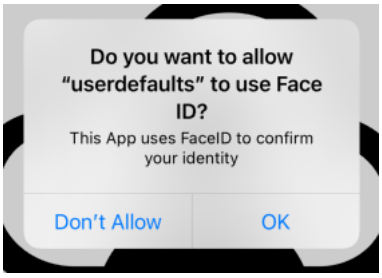

Figure 21: Permisos FaceID

Figure 22: FaceID esperando respuesta del simulador

Paso 14: Damos **click** sobre el botón de **OK**, para luego ver la usual animación del **FaceID** como si se hubiera congelado. (ver figura 21)

Paso 15: Para dar respuesta al mismo ya que nos encontramos en un entorno de simulación. Debemos nuevamente ir a la barra superior del simulador Features-Face ID y para este intento seleccionamos la opción **Non-matching Face**, para de esta manera recibir un mensaje similar al de la figura 22

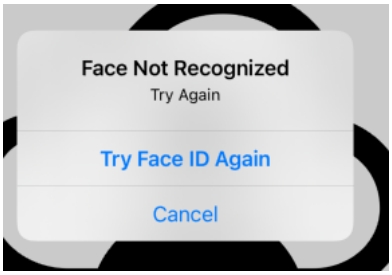

Figure 23: FaceID con resultado fallido

Paso 16: Damos **Click** en cancelar. Y volvemos a intentar desde le icono de **FaceID**.

Paso 17: Esta vez seleccionamos dentro de las opciones de Features-Face ID la opción **Matching Face** para esta vez, que nuestro usuario logre hacer **Login**