VNiVERSiDAD D SALAMANCA

Documentación y Manual de Desarrollador
Desarrollo de APP Avanzadas
Álvaro García Sánchez
70924450V

1. Arquitectura de la APP

Como se indica en el enunciado, se pide una arquitectura modelo-vista-vista-modelo (**MVVM**) en la que se han estructurado los archivos de la siguiente manera:

- ➔ La Vista principal se implementa sobre el ContentView() de la aplicación por defecto
- ➔ El modelo general de datos es Manufacturer, se explican todos los campos más adelante.
- ➔ Las funciones que se encargan de actualizar el modelo y las vistas se implementan sobre el fichero ManufacturerViewModel.
- ➔ La vista número 2 será DetailManufacturerView que se instancia al presionar un elemento de la vista principal.
- ➔ Finalmente, la vista número 3 será DetailBeerView que aparece al presionar un elemento de la vista número 2.
- ➔ El resto de funciones auxiliares se explican con más detalle en los siguientes apartados
- ➔ Los datos iniciales de la aplicación se cargan desde un fichero json.

Con esta arquitectura de diseño se logra el objetivo de llevar a cabo la separación del apartado de la interfaz de usuario (View) de la parte lógica (Model).

2. Vista principal (ContentView)

En esta vista, tal y como se pide en el enunciado, se representa un elemento **List** que a su vez contiene dos secciones separando fabricantes entre nacionales e importados. En cada una de estas secciones se instancian elementos **ManufacturerRow** que representan la imagen y el nombre de un fabricante concreto.

Por otro lado, en la barra de navegación de la vista se crea un botón **“+”** para mostrar una nueva vista con un formulario que nos permite añadir un nuevo fabricante a la lista. Los detalles de esta vista son los siguientes:

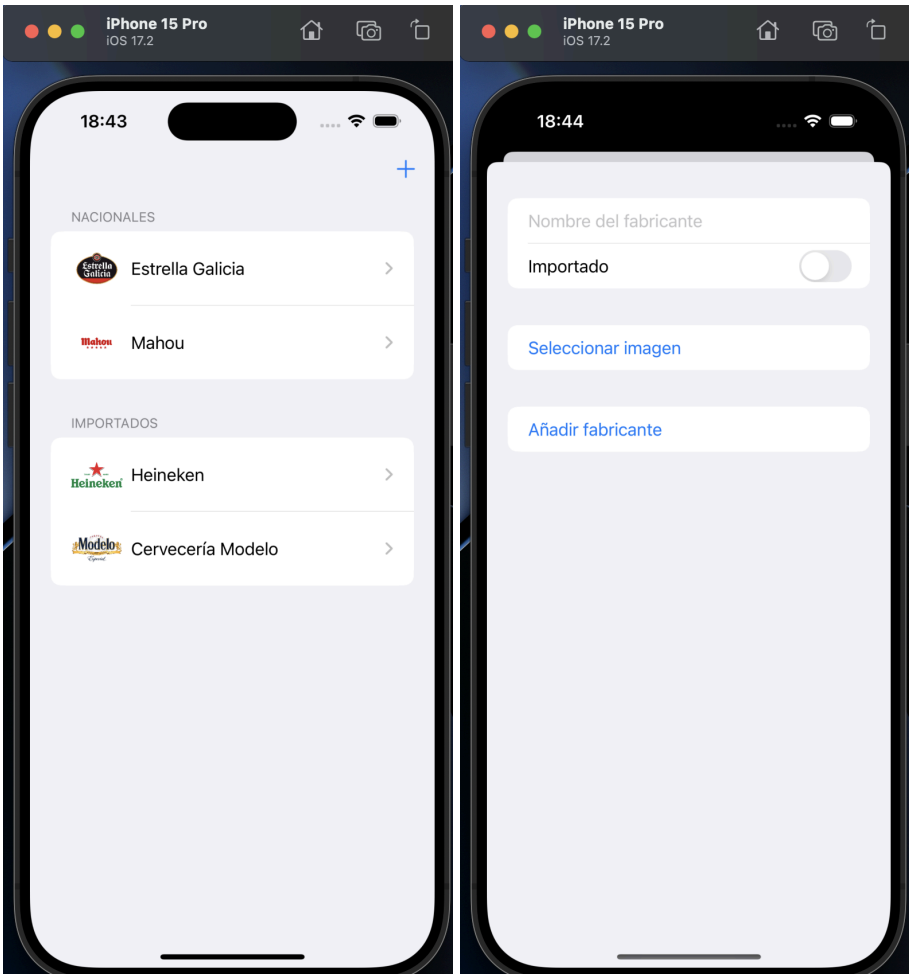
2.1. AddManufacturerView

En esta vista se muestra un formulario en el que el usuario podrá introducir un nombre para el nuevo fabricante, un **toogle** para seleccionar si el fabricante va a ser nacional o importado y en consecuencia colocarlo correctamente en la lista, y finalmente, un boton de **“Seleccionar imagen”** que abrirá la galería del dispositivo para que el usuario pueda elegir una imagen para dicho fabricante. Al final de la vista, se muestra el botón **“Añadir fabricante”** que guardará los datos en el modelo del nuevo fabricante, cerrará la vista del formulario y añadirá el nuevo elemento a la vista principal.

Por otro lado, se implementa también la función de SwiftUI que permite eliminar un elemento de la **List** haciendo un gesto de desplazamiento. Esto se gestiona desde una función auxiliar en la **ContentView** que manda la información necesaria a otra función

implementada en el **ManufacturerViewModel** que eliminará el fabricante seleccionado de forma permanente.

Finalmente, al presionar cualquiera de los fabricantes de la **List**, se mostrará la vista número 2 del enunciado que muestra una vista detallada de las cervezas que ofrece el fabricante. Está vista tiene el nombre **DetailManufacturerView**.



3. Vista 2 (DetailManufacturerView)

Una vez presionamos cualquier elemento de la List de la vista anterior, se muestra la vista de detalle del fabricante seleccionado. En concreto en esta vista, se muestra la lista de cervezas que tiene dicho fabricante en su catálogo. Adicionalmente, en la barra de navegación dispondremos de un botón **"+"** que nos permitirá añadir una nueva cerveza al catálogo y otro botón **"⇅"** que nos permitirá ordenar los elementos de la lista en función de su nombre, calorías o graduación alcohólica. Por otro lado, debajo de la barra de navegación, se crea un elemento **"SearchBar"** o barra de búsqueda que nos permitirá encontrar la cerveza que queramos siempre y cuando esté disponible en el catálogo. Vamos a ver estas dos vistas auxiliares:

3.1. AddBeerView

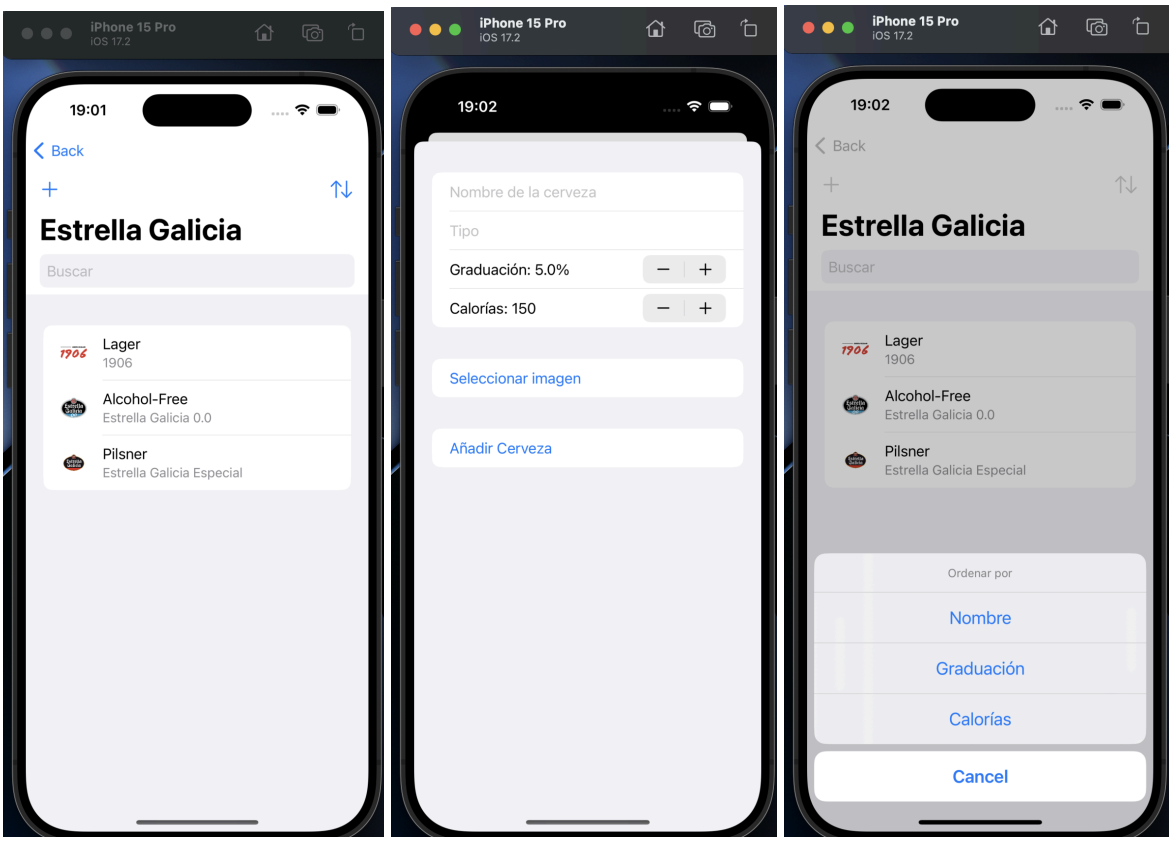
Como mencionaba anteriormente, en esta vista auxiliar podremos añadir una nueva cerveza al catálogo del fabricante mediante un formulario. El funcionamiento es similar al ya comentado antes para añadir un nuevo fabricante. Podremos darle un valor de nombre, imagen, calorías y graduación y, una vez hecho esto, se presiona el botón **“Añadir cerveza”** y se crea un nuevo **BeerRow** que se mostrará en la lista de la vista de detalle.

3.2. SortBeer

Como tal esto no es una vista si no una función que simplemente muestra un pequeño **popup** en la parte inferior del dispositivo. En este **popup** se muestran las opciones que existen para ordenar los elementos de la lista y simplemente haciendo click, estos se ordenan.

Finalmente, nos encontramos en la esquina superior izquierda un botón **“Back”** para volver a la vista número 1 aunque también puede hacerse mediante un gesto de desplazamiento lateral de SwiftUI. En esta vista también se pueden eliminar elementos con un gesto de desplazamiento.

Por otro lado, en el siguiente punto explicaré en detalle la visualización de la vista número 3 al seleccionar una cerveza en concreto de la lista.

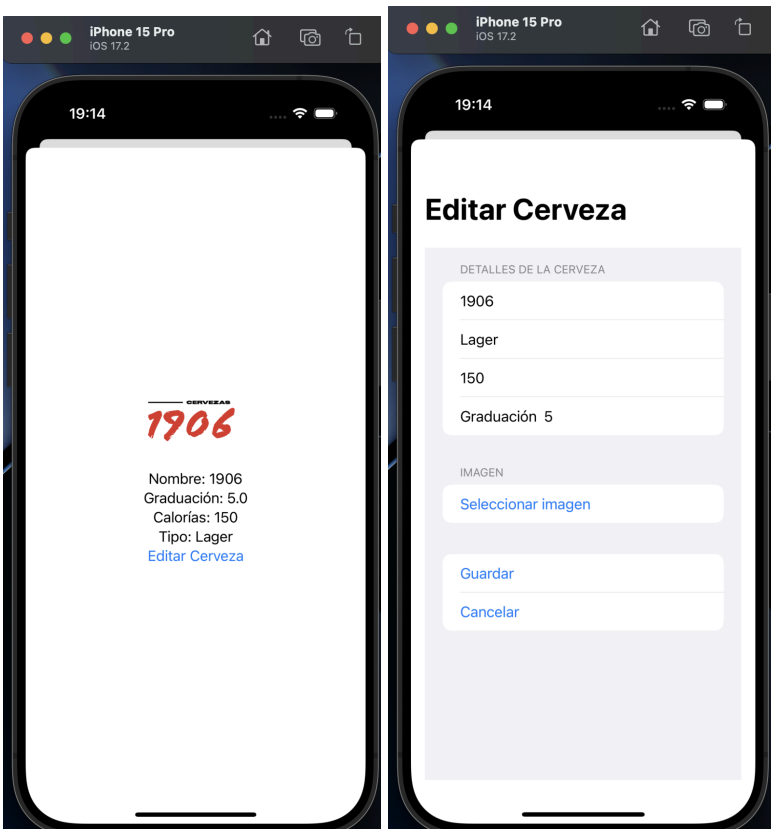


4. Vista 3 (DetailBeerView)

Finalmente, en esta última vista, se mostrarán los detalles de la cerveza que no se ven en la vista anterior como las calorías, graduación, etc. En esta vista se incluye un botón que abre un formulario que permite al usuario modificar los campos de dicha cerveza seleccionada. Una vez presionado este botón se muestra EditBeerView. Vamos a ver sus detalles:

4.1. EditBeerView

En esta vista se muestra un formulario con los campos a modificar de la cerveza ya completados con los datos actuales. Una vez modificados se le da al usuario la opción mediante botones de “Cancelar” que dejará el formulario como estaba y cerrará la ventana, o “Guardar” que cambiará los datos de la cerveza y cerrará la ventana.



5. Modelo de datos (Manufacturer)

En este fichero, establecemos las estructuras de datos para los fabricantes y para las cervezas del siguiente modo:

```swift
//Modelo de datos de fabricante
struct Manufacturer: Identifiable, Codable {
    var id = UUID()
    var name: String
    var imageData: String
    var origin: Bool
    var beers: [Beer]
}

//Modelo de datos de cerveza
struct Beer: Identifiable, Hashable, Codable {
    var id = UUID()
    var name: String
    var imageBeer: String
    var graduation: Double
    var calories: Int
    var type: String
}
```

Si por algún motivo quisiéramos, por ejemplo añadir varias fotos de una cerveza en concreto, habría que modificar el campo **imageBeer** y hacerlo como un array de tipo Data.

6. Vista-Modelo (ManufacturerViewModel)

En este fichero se gestiona y actualiza todos los datos entre el modelo y la interfaz, las funciones que realiza son:

- ➔ Carga inicial de datos desde el json.
- ➔ Añadir fabricante
- ➔ Guardar imágenes
- ➔ Borrar fabricantes
- ➔ Añadir cerveza
- ➔ Actualizar cerveza

```swift
//Funcion para cargar el conjunto inicial de datos desde un fichero JSON
func loadInitialData() {
        if let url = Bundle.main.url(forResource: "InitialData", withExtension: "json") {
            do {
                let data = try Data(contentsOf: url)
                print(String(data: data, encoding: .utf8) ?? "Error al imprimir datos JSON")
                let decodedData = try JSONDecoder().decode([Manufacturer].self, from: data)
                manufacturers = decodedData
                print("Manufacturers count after loading: \(manufacturers.count)")
            } catch {
                print("Error loading initial data: \(error)")
            }
        }
    }
```

```swift
    // Función para agregar un nuevo fabricante
    func addManufacturer(_ manufacturer: Manufacturer) {
           if manufacturers.contains(where: { $0.id == manufacturer.id }) {
               // Si el fabricante ya existe, no lo agregamos nuevamente
               return
           }

           manufacturers.append(manufacturer)
       }

    //Funcion para guardar imagen en los assets del proyecto
    func saveImage(_ image: UIImage, forManufacturerWithID id: UUID, imageName: String) {
        if let index = manufacturers.firstIndex(where: { $0.id == id }) {
            // Guardar el nombre del archivo de imagen en lugar de los datos de la imagen
            manufacturers[index].imageData = imageName
        }
    }

    //Funcion para eliminar completamente un fabricante
    func deleteManufacturerAndBeers(at index: Int) {
            guard index < manufacturers.count else { return }
            manufacturers.remove(at: index)
        }

    //Funcion para añadir una cerveza a un fabricante seleccionado
    func addBeer(_ beer: Beer, toManufacturerWithID manufacturerID: UUID) {
            if let index = manufacturers.firstIndex(where: { $0.id == manufacturerID }) {
                manufacturers[index].beers.append(beer)
            }
        }
```

```swift
//Funcion para actualizar los datos de una cerveza de un fabricante
func updateBeer(_ updatedBeer: Beer) {
    for i in 0..<manufacturers.count {
        if let beerIndex = manufacturers[i].beers.firstIndex(where: { $0.id == updatedBeer.id }) {
            manufacturers[i].beers[beerIndex] = updatedBeer
            break
        }
    }
}
```

7. Conclusiones

Durante la realización del proyecto me he encontrado con múltiples problemas, sobre todo a la hora de gestionar las imágenes y de mantener las vistas actualizadas pese a utilizar instancias **@observableObject** del viewModel que en algunas de las vistas no se ven reflejados los cambios al momento de realizar una acción sino después de que se actualiza la vista.

Pese a todo ello ha sido una experiencia gratificante todo el proceso de familiarización con el entorno de desarrollo de SwiftUI ya que proporciona un amplio abanico de posibilidades a la hora de creación de aplicaciones para dispositivos Apple.

Por algunos motivos no me fue posible crear la vista número 4 opcional en la que se representan una serie de reseñas de cervezas por parte de los usuarios empleando una API REST y tampoco me ha sido posible resolver técnicamente el problema de guardar los

datos en un fichero externo después de una ejecución de la aplicación para luego poder usarlos en futuros usos.