

DESARROLLO DE APLICACIONES AVANZADAS

BEER APP - MANUALES

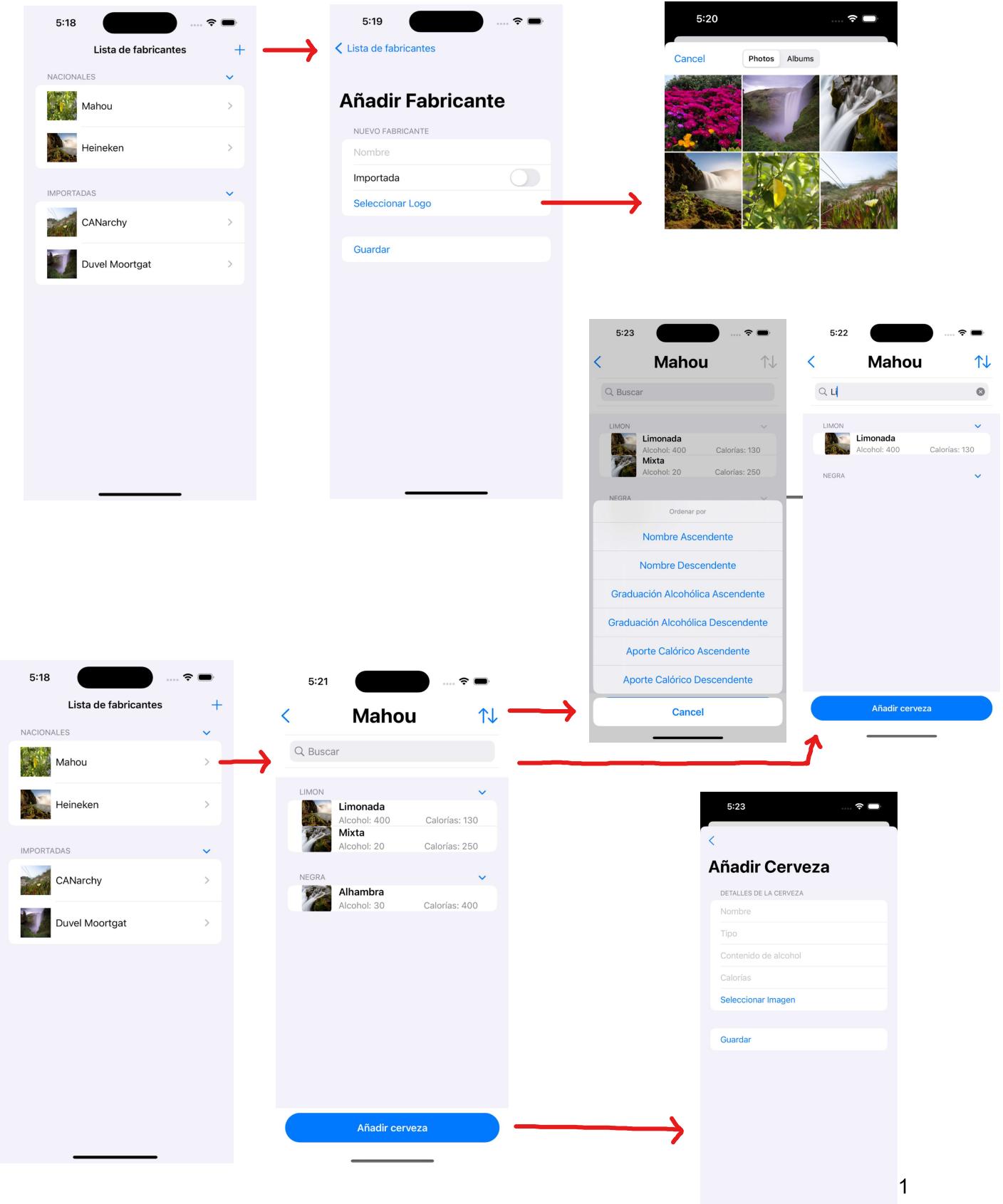
Natalia Vicente Sánchez - 70918190]

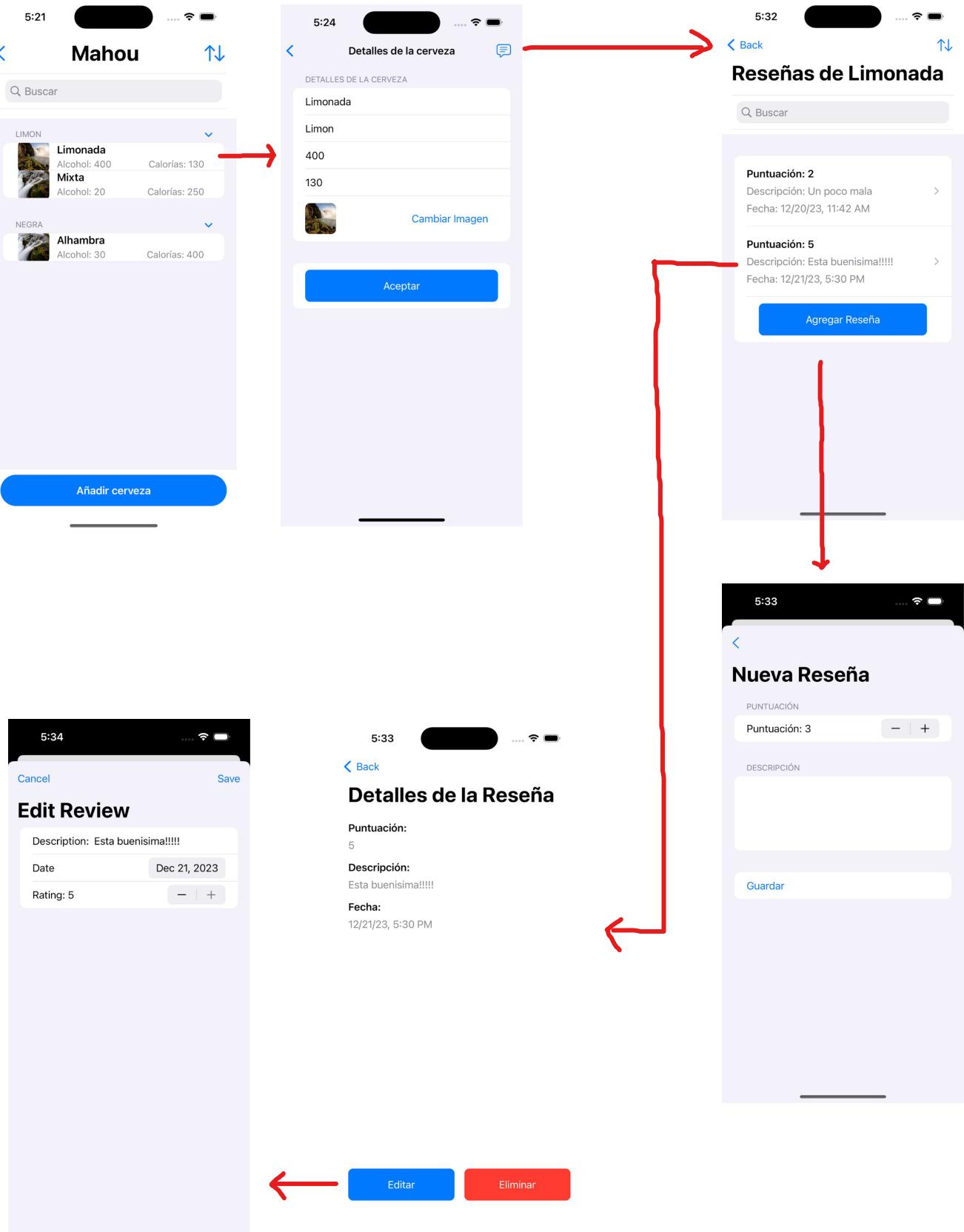
| | |
|--|-----------|
| 1. Manual del usuario..... | 1 |
| 2. Manual del desarrollador..... | 8 |
| 2.1 Vistas..... | 9 |
| 2.1.1 ContentView..... | 10 |
| 2.1.2 BeersView..... | 11 |
| 2.1.3 BeerDetailView..... | 16 |
| 2.2. Vista-Modelo..... | 17 |
| 2.2.1 Guardar cerveza..... | 17 |
| 2.2.2 Borrar cerveza..... | 18 |
| 2.2.3 Ordenar cervezas..... | 18 |
| 2.2.4 Actualizar cerveza..... | 19 |
| 2.2.5 Cargar datos..... | 19 |
| 2.2.5.1 Cargar datos de Documentos..... | 20 |
| 2.2.5.2 Cargar datos del Bundle..... | 20 |
| 2.2.6 Guardar datos..... | 20 |
| 2.3 Modelo..... | 21 |
| 2.4 Anotaciones..... | 22 |

1. Manual del usuario

En este manual se va mostrar el flujo de la aplicación para poder comprender mejor el funcionamiento de esta, esto se hará mostrando las diferentes vistas.

Vista principal:





2. Manual del desarrollador

En este manual se van a comentar los aspectos relevantes sobre la programación de la aplicación de cervezas, así como las funcionalidades destacadas y novedosas.

Se va a tratar, por tanto, de una **aplicación de cervezas** donde se van a poder consultar, añadir y eliminar diferentes fabricantes de estas, así como consultar las cervezas de cada uno con diferentes opciones como modificarlas, ordenarlas, borrarlas, filtrarlas o añadir más. Además, dentro de cada cerveza podremos, a parte de poder apreciar sus datos en detalle, consultar las reseñas que tenga cada una de estas junto con las mismas opciones que las cervezas.

Es importante mencionar que la aplicación será desarrollada en **Swift (SwiftUI)**, en el entorno **Xcode**. Este programa ha sido:

- Creado con Xcode 14.3.1 (14E300c)
- Codificado en Swift 5.8.1
- Compilado en iOS 16.4

Además, la arquitectura que va a seguir la aplicación será la de **Modelo-Vista-Vista-Modelo (MVVM)**. En la cual separaremos la lógica de la interfaz de usuario de la lógica de negocio. Donde tendremos:

- **Vista:** Correspondiente a toda la lógica de interfaz de usuario, estará compuesta a su vez por las siguientes vistas:
 - **ContentView:** Será la vista principal donde se mostrarán los fabricantes y sus diferentes opciones.
 - **AddManufacturerView:** Nos mostrará un formulario donde llenaremos los datos del nuevo fabricante que queremos añadir.
 - **BeersView:** Se trata de la vista que nos mostrará la lista de cervezas del fabricante seleccionado anteriormente, separada en secciones en función de su tipo, junto con sus datos y diferentes opciones para operar con ellas.
 - **BeerDetailView:** Nos mostrará de manera más amplia los detalles de la cerveza seleccionada, así como la posibilidad de modificarlos. También estará disponible la opción de consultar sus reseñas.
 - **AddBeerView:** Nos mostrará un formulario donde llenaremos los datos de la nueva cerveza que queremos añadir.
 - **ReviewListView:** Se trata de la vista que nos mostrará la lista de reseñas de la cerveza seleccionada anteriormente, separada en secciones, junto con sus datos y diferentes opciones para operar con ellas.

- **ReviewDetailView:** Nos mostrará de manera más amplia los detalles de la reseña seleccionada, así como la posibilidad de modificarlos o de eliminar la reseña.
- **AddReviewView:** Nos mostrará un formulario donde llenaremos los datos de la nueva reseña que queremos añadir.
- **EditReviewView:** Nos mostrará un formulario donde podremos editar los datos de la reseña que queremos modificar.
- **Vista-Modelo:** correspondiente a toda la lógica de negocio de nuestra aplicación, estará compuesta por tanto por las distintas funciones que me permitan el correcto funcionamiento de la aplicación.
- **Modelo:** Aquí se encuentra la descripción de los tipos de datos que se manejan a lo largo del transcurso de la aplicación.

2.1 Vistas

A continuación, se van a exponer una serie de vistas para poder tomarlas como referencia y comprender las restantes, las cuales serán prácticamente análogas:

2.1.1 ContentView

Se trata de la vista principal donde se van a mostrar los fabricantes, para ello, es importante tener instanciado el **viewModel**: `@StateObject var viewModel = ViewModel()`

Al cual accederemos para utilizar sus métodos y obtener y guardar los datos. Dentro de esta vista tendremos dos secciones, una para los fabricantes nacionales y otra para los no nacionales, para comentarlo vamos a tomar de referencia uno:

```
NavigationView {
    VStack {
        List {
            if (viewModel.manufacturers.count >= 1) { //>0 para que muestre si hay o no fabricantes
                //SECCION DE NACIONALES
                Section(header: Text("Nacionales")) {
                    ForEach($viewModel.manufacturers, id: \.id) { $manufacturer in
                        if !manufacturer.isImported {
                            //Si selecciona un fabricante
                            NavigationLink(destination: BeersView(manufacturer: manufacturer, viewModel:
                                viewModel),
                                tag: manufacturer,
                                selection: $selectedManufacturer) {
                                    //apartado foto y nombre
                                    HStack {
                                        if let imageData = manufacturer.logoImageData,
                                            let uilImage = UIImage(data: imageData) {
                                                Image(uilImage)
                                                    .resizable()
                                                    .frame(width: 50, height: 50)
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
```

```
        Text(manufacturer.name)
    }
    //accion de borrar deslizando
    .swipeActions(edge: .leading) {
        Button {
            viewModel.removeManufacturer(withId: manufacturer.id)
        } label :{
            Label("Borrar", systemImage: "trash.fill")
        }
        .tint(.red)
    }
}
}
```

- Como vemos, tenemos un **NavigationView** (que se utilizará en el resto de vistas), compuesto por un **VStack** que nos permitirá representar ambas listas o secciones de manera vertical, es decir, una debajo de la otra, en este caso la primera será la de los fabricantes nacionales.
- Podemos observar que comprobamos si hay más de un fabricante, en caso contrario, se muestra un mensaje diciendo que no los hay. Dentro de la sección, recorreremos la lista de fabricantes mediante un **ForEach**, y en caso de ser Nacional, como en este caso, lo mostraremos en su debida sección.
- Destacar que se ha utilizado el **NavLink** que, si pulsamos un fabricante, nos dirigirá a la vista de las cervezas, pasando a esa vista el **viewModel** sobre el que estamos trabajando y el fabricante que se ha seleccionado.
- Por último, se ha hecho uso del **HStack** para presentar de manera horizontal cada cerveza, teniendo primero la imagen y después su nombre. Además si se desliza hacia la derecha, podremos borrar el fabricante y toda su información asociada.

2.1.2 BeersView

En esta vista se van a mostrar todas las cervezas del fabricante seleccionado en la vista anterior, al igual que antes utilizaremos un **VStack** para representar cada dato de manera vertical:

```
 VStack {
    HStack {
        //Boton para volver hacia atras
        Button(action: {
            presentationMode.wrappedValue.dismiss()
        }) {
            Image(systemName: "chevron.left")
                .font(.title)
                .foregroundColor(.blue)
                .padding(.leading, 8) // Ajustar el espacio a la izquierda
        }
    }
}
```

```
    }
    Spacer()
    //Nombre del fabricante
    Text(manufacturer.name)
        .font(.largeTitle)
        .fontWeight(.bold)
    Spacer()
    //Botón para ordenar
    Button(action: {
        isShowingSortOptions = true
    }) {
        Image(systemName: "arrow.up.arrow.down")
            .font(.title)
            .padding(.trailing)
    }
}
.padding(.top, 8)
.padding(.bottom, 4)
.background(Color( UIColor.systemBackground))
SearchBar(searchText: $searchText)
    .padding(.horizontal)
    .padding(.bottom, 8)
```

- Arriba de la pantalla, mostraremos de manera alineada, en horizontal, gracias al **HStack**, el botón (flecha) para volver hacia atrás, el nombre del fabricante, y el botón para ordenar las cervezas.

- Debajo de esta línea, encontramos la barra de búsqueda o **earchBar** que nos permitirá filtrar las cervezas. Esta barra de búsqueda será:

```
struct SearchBar: UIViewRepresentable {
    @Binding var searchText: String

    class Coordinator: NSObject, UISearchBarDelegate {
        @Binding var searchText: String

        init(searchText: Binding<String>) {
            _searchText = searchText
        }

        func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
            self.searchText = searchText
        }
    }

    func makeCoordinator() -> Coordinator {
        return Coordinator(searchText: $searchText)
    }
}
```

```
func makeUIView(context: Context) -> UISearchBar {
    let searchBar = UISearchBar()
    searchBar.delegate = context.coordinator
    searchBar.placeholder = "Buscar"
    searchBar.autocapitalizationType = .none
    return searchBar
}

func updateUIView(_ uiView: UISearchBar, context: Context) {
    uiView.text = searchText
}
}
```

- Siguiendo debajo de la barra de búsqueda tendremos la representación, mediante una lista, de las cervezas:

```
List {
    ForEach(Array(groupedBeers.keys.sorted()), id: \.self) { key in
        Section(header: Text(key)) {
            ForEach(groupedBeers[key]!.filter {
                searchText.isEmpty || $0.name.localizedCaseInsensitiveContains(searchText)
            }) { beer in
                BeerRow(beer: beer, viewModel: viewModel)
                    .swipeActions(edge: .trailing) {
                        Button {
                            // Eliminar la cerveza
                            if let index = manufacturer.beers.firstIndex(where: { $0.id == beer.id }) {
                                viewModel.removeBeer(withId: beer.id)
                            }
                        } label: {
                            Label("Eliminar", systemImage: "trash")
                        }
                        .tint(.red)
                    }
            }
        }
    }
}
```

- Al igual que antes se recorren las cervezas del fabricante mediante un **ForEach**, las cuales vamos a agrupar en secciones en función del tipo:

```
private var groupedBeers: [String: [Beer]] {
    Dictionary(grouping: manufacturer.beers, by: { $0.type })
}
```

- Las mostraremos mediante el struct creado, el cual comentaremos a continuación, **BeerRow**. Al igual que antes se posibilita la acción de borrar una cerveza deslizando hacia la derecha.

Es importante destacar la última parte del código:

```
Button(action: {
    isAddingBeer = true
}) {
    Text("Añadir cerveza")
        .font(.headline)
        .foregroundColor(.white)
        .frame(maxWidth: .infinity)
        .frame(height: 50)
        .background(Color.blue)
        .cornerRadius(25)
        .padding(.horizontal)
}
.sheet(isPresented: $isAddingBeer) {
    AddBeerView(viewModel: viewModel, manufacturer: manufacturer, isAddingBeer:
$isAddingBeer)
}
.sheet(item: $selectedBeer) { beer in
    BeerDetailView(viewModel: viewModel, beer: beer)
}
Spacer()
}
.actionSheet(isPresented: $isShowingSortOptions) {
    ActionSheet(title: Text("Ordenar por"), buttons: [
        .default(Text("Nombre Ascendente")) {
            viewModel.sortBeers(.name(ascending: true), for: manufacturer.id)
        },
        .default(Text("Nombre Descendente")) {
            viewModel.sortBeers(.name(ascending: false), for: manufacturer.id)
        },
        .default(Text("Graduación Alcohólica Ascendente")) {
            viewModel.sortBeers(.alcoholContent(ascending: true), for: manufacturer.id)
        },
        .default(Text("Graduación Alcohólica Descendente")) {
            viewModel.sortBeers(.alcoholContent(ascending: false), for: manufacturer.id)
        },
        .default(Text("Aporte Calórico Ascendente")) {
            viewModel.sortBeers(.calories(ascending: true), for: manufacturer.id)
        },
        .default(Text("Aporte Calórico Descendente")) {
            viewModel.sortBeers(.calories(ascending: false), for: manufacturer.id)
        },
        .cancel()
    ])
}
```

- Se incluye debajo de todas las cervezas, el botón para **añadir cervezas**.

- Si se pulsa el botón anterior se dirige al usuario a la vista para añadir cervezas (**AddBeerView**), esto se realiza controlando el valor de la variable de tipo Bool, **isAddingBeer**, gracias a **.sheet**.
- Lo mismo ocurre cuando se selecciona una cerveza, en este caso, se redirige al usuario a la vista detallada de la cerveza (**BeerDetailView**).
- Si se pulsa el botón para ordenar las cervezas, se activa el **.actionSheet** gracias a otra variable (**isShowingSortOptions**), y en función de la opción seleccionada por el usuario, se llamará a la función **sortBeers** del viewModel.

Por último, cabe destacar el struct **BeerRow**, el cual mostrará los datos de la cerveza:

```
struct BeerRow: View {  
    var beer: Beer  
    var viewModel: ViewModel // Agrega viewModel como parámetro  
  
    @State private var isActive = false  
  
    var body: some View {  
        NavigationLink(destination: BeerDetailView(viewModel: viewModel, beer: beer), isActive: $isActive) {  
            EmptyView()  
        }  
        .opacity(0)  
        .background(  
            Button(action: {  
                isActive = true  
            }) {  
                HStack(spacing: 12) {  
                    if let imageData = beer.logoImageData,  
                        let uiImage = UIImage(data: imageData) {  
                        Image(uiImage: uiImage)  
                            .resizable()  
                            .frame(width: 50, height: 50)  
                            .aspectRatio(contentMode: .fit) // Ajusta la relación de aspecto  
                            .cornerRadius(8)  
                            .padding(.vertical, 4) // Añade espacio arriba y abajo de la imagen  
                            .padding(.leading, 4) // Elimina el espacio a la izquierda de la imagen  
                    }  
                    VStack(alignment: .leading, spacing: 4) {  
                        Text(beer.name)  
                            .font(.headline)  
                        HStack {  
                            Text("Alcohol: \(beer.alcoholContent)")  
                                .foregroundColor(.secondary)  
                            Spacer()  
                            Text("Calorías: \(beer.calories)")  
                                .foregroundColor(.secondary)  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```
        }
        .font(.subheadline)
    }
}
.padding([.top, .bottom], 8) // Agrega espacio arriba y abajo de cada sección de cerveza
.padding(.trailing, 12) // Ajusta el espacio a la derecha para mantener el equilibrio visual
}
.background(Color.white) // Añade un fondo blanco para cada fila de cerveza
.cornerRadius(8) // Redondea las esquinas de la fila de cerveza
)
}
}
```

- Al igual que antes, con el **NavLink** podremos dirigirnos a la vista en detalle de las cervezas, pasando como parámetro la cerveza seleccionada y el viewModel utilizado.

- De manera horizontal, como los fabricantes, mostraremos los **datos** de las cervezas, mediante una imagen, verticalmente el nombre y horizontalmente el grado alcohólico y las calorías.

2.1.3 BeerDetailView

En esta vista, no se va a comentar como se ha dispuesto la información, ya que es muy similar a todo lo anterior, con la salvedad de que se ha utilizado **TextField** para la introducción de nuevo datos. Sin embargo, es importante destacar el uso de un “**Image Picker**” para la selección de la imagen:

```
HStack {
    if let imageData = selectedImage ?? UIImage(data: beer.logoImageData ?? Data()) {
        Image(uiImage: imageData)
            .resizable()
            .frame(width: 50, height: 50)
            .cornerRadius(8)
    }
    Spacer()
    Button(action: { isShowingImagePicker.toggle() }) {
        Text("Cambiar Imagen")
    }
    .sheet(isPresented: $isShowingImagePicker) {
        ImagePicker(selectedImage: $selectedImage)
    }
    .padding()
}
```

- Controlaremos la aparición del *Picker*, como antes, mediante una variable de tipo Bool, **isShowingImagePicker**. El código del **Picker** a destacar es el siguiente:

```
struct ImagePicker: UIViewControllerRepresentable {
```

```
@Binding var selectedImage: UIImage?  
@Environment(\.presentationMode) var presentationMode  
  
class Coordinator: NSObject, UIImagePickerControllerDelegate, UINavigationControllerDelegate {  
    let parent: ImagePicker  
  
    init(parent: ImagePicker) {  
        self.parent = parent  
    }  
  
    func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {  
        if let pickedImage = info[.originalImage] as? UIImage {  
            parent.selectedImage = pickedImage  
        }  
        parent.presentationMode.wrappedValue.dismiss()  
    }  
  
    func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {  
        parent.presentationMode.wrappedValue.dismiss()  
    }  
}  
  
func makeCoordinator() -> Coordinator {  
    return Coordinator(parent: self)  
}  
  
func makeUIViewController(context: Context) -> UIViewController {  
    let picker = UIImagePickerController()  
    picker.delegate = context.coordinator  
    return picker  
}  
  
func updateUIViewController(_ uiViewController: UIImagePickerController, context: Context) {}  
}
```

2.2. Vista-Modelo

En esta parte del código encontraremos todas las funcionalidades del programa. Por ello, se van a mostrar algunas de las funcionalidades más destacadas:

2.2.1 Guardar cerveza

```
func saveBeer(name: String, type: String, alcoholContent: String, calories: String, logoImageData: Data?,  
manufacturer: ManufacturerModel) {  
    var modifiedManufacturer = manufacturer // Crear una copia mutable  
  
    let newBeer = Beer(  
        name: name,  
        type: type,  
        alcoholContent: alcoholContent,  
        calories: calories,  
        logoImageData: logoImageData,  
        reviews: []  
    )  
  
    modifiedManufacturer.beers.insert(newBeer, at: 0) // Modificar la copia  
  
    if let index = manufacturers.firstIndex(where: { $0.id == modifiedManufacturer.id }) {  
        manufacturers[index] = modifiedManufacturer // Actualizar en la lista de fabricantes  
    }  
}
```

Esta función recibirá los datos de la cerveza que se desea añadir así como el fabricante al que se le añadirá, creará un nuevo objeto **newBeer** al cual rellenará sus datos con los recibidos en la función y lo insertará como primer elemento en la lista de cervezas del fabricante, sin olvidar que también se modifica el antiguo fabricante por el nuevo fabricante con la cerveza añadida, buscando en la lista de fabricantes el ID y sustituyéndolo.

2.2.2 Borrar cerveza

```
func removeBeer(withId id: String) {  
    for index in 0..        if let beerIndex = manufacturers[index].beers.firstIndex(where: { $0.id == id }) {  
            manufacturers[index].beers.remove(at: beerIndex)  
            return  
        }  
    }  
}
```

Esta función recibirá el ID de la cerveza que se desea borrar, y por tanto, se realizará una búsqueda del fabricante el cual contenga dicha cerveza comprobando los ID. Una vez obtenido, se borra la cerveza de la lista del fabricante gracias a los dos ID que se manejan como índices.

2.2.3 Ordenar cervezas

```
func sortBeers(_ sortBy: SortType, for manufacturerID: String) {
    guard let index = manufacturers.firstIndex(where: { $0.id == manufacturerID }) else { return }

    switch sortBy {
    case .name(let ascending):
        manufacturers[index].beers.sort { ascending ? $0.name < $1.name : $0.name > $1.name }
    case .alcoholContent(let ascending):
        manufacturers[index].beers.sort { ascending ? $0.alcoholContent < $1.alcoholContent :
$0.alcoholContent > $1.alcoholContent }
    case .calories(let ascending):
        manufacturers[index].beers.sort { ascending ? $0.calories < $1.calories : $0.calories > $1.calories }
    }
}
```

Esta función recibe la opción de ordenación y el ID del fabricante al cual se le van a ordenar las cervezas. Es posible ordenar de manera ascendente o descendente el nombre, grado de alcohol y calorías. Esta es la **enumeración** utilizada para el tipo de ordenación:

```
enum SortType {
    case name(ascending: Bool)
    case alcoholContent(ascending: Bool)
    case calories(ascending: Bool)
}
```

2.2.4 Actualizar cerveza

```
func updateBeerDetails(_ editedBeer: Beer) {
    for index in 0..
```

En esta función se busca el índice o ID del fabricante al cual se le va a modificar la cerveza, esto es comparando el ID de la cerveza actualizada con el de la cerveza sin actualizar. Una vez encontrado, se sustituye la cerveza en la lista de cervezas del fabricante gracias a los índices.

2.2.5 Cargar datos

```
func loadData() {
    if let documentsDirectory = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
        let fileURL = documentsDirectory.appendingPathComponent("Manufacturers.json")

        /*if hasLaunchedBefore() {
            // Si es la primera vez, cargar datos del bundle
            print("PRIMERA VEZ")
            loadManufacturersFromBundle()
        } else */if FileManager.default.fileExists(atPath: fileURL.path) {
```

```
print("YA ACCEDI, ABRO DOCUMENTOS")
// Si no es la primera vez y hay datos en Documents, cargar desde allí
loadManufacturersFromJSON(fileURL: fileURL)
} else {
    // Si no es la primera vez pero no hay datos en Documents, cargar del bundle
    print("YA ACCEDI, ABRO BUNDLE")
    loadManufacturersFromBundle()
}
}
```

Esta es la función que se encarga de cargar los datos nada más abrir la aplicación, para empezar se declara la ruta hacia *Documentos*, en la que se espera encontrar un archivo **Manufacturers.json**. Inicialmente, se trató de implementar dentro, la función que detectaba si era la primera vez que se lanzaba la aplicación, sin embargo esto no se logró, se comentará más adelante.

Por tanto, si se accede a la aplicación, se comprobará si existe el json en *Documentos*, y en ese caso lo cargará, si no, se pasa a cargar los datos del *Bundle* del proyecto.

2.2.5.1 Cargar datos de Documentos

```
func loadManufacturersFromJSON(fileURL: URL) {
    do {
        let jsonData = try Data(contentsOf: fileURL)
        let decoder = JSONDecoder()
        let decodedManufacturers = try decoder.decode([ManufacturerModel].self, from: jsonData)
        self.manufacturers = decodedManufacturers
    } catch {
        print("Error cargando datos desde el archivo JSON en Documents: \(error.localizedDescription)")
    }
}
```

Esta función simplemente decodifica los datos del json ubicado en *Documentos*, una vez hecho esto, se actualizará la lista de fabricantes con los datos obtenidos.

2.2.5.2 Cargar datos del Bundle

```
func loadManufacturersFromBundle() {
    if let path = Bundle.main.path(forResource: "Manufacturers", ofType: "json") {
        do {
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))
            let decoder = JSONDecoder()
            let decodedManufacturers = try decoder.decode([ManufacturerModel].self, from: jsonData)
            self.manufacturers = decodedManufacturers
            setAppLaunched()
        } catch {
            print("Error cargando datos desde el bundle: \(error.localizedDescription)")
        }
    }
}
```

Esta función simplemente decodifica los datos del json ubicado en el *Bundle* del proyecto, una vez hecho esto, se actualizará la lista de fabricantes con los datos obtenidos.

2.2.6 Guardar datos

```
func saveDataToDocuments() {
    if let encodedData = try? JSONEncoder().encode(manufacturers) {
        if let documentsDirectory = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
            let fileURL = documentsDirectory.appendingPathComponent("Manufacturers.json")
            do {
                try encodedData.write(to: fileURL)
                print("Datos guardados correctamente en \(fileURL)")
            } catch {
                print("Error al guardar datos en Documents: \(error)")
            }
        }
    }
}
```

Esta función será utilizada cuando se detecte el cierre de la aplicación y exportará los datos en formato json a la ubicación utilizada anteriormente de *Documentos*. Las funciones que sirven para detectar el cierre de la aplicación son:

```
func subscribeToAppEvents() {
    NotificationCenter.default.addObserver(self, selector: #selector(saveDataOnAppExit), name:
UIApplication.willResignActiveNotification, object: nil)
    print("Se han agregado observadores para eventos de la app")
}

@objc func saveDataOnAppExit() {
    print("La app está siendo minimizada o cerrada. Guardando datos...")
    saveDataToDocuments()
}
```

Se añaden observadores que detectarán si se cierra la aplicación, en ese caso se guardarán, como se ha comentado anteriormente, los datos en formato json.

Para que esto funcione correctamente, la suscripción a los eventos junto con la carga de los datos se realizará al abrir la aplicación:

```
init() {
    subscribeToAppEvents()
    loadData()
}
```

2.3 Modelo

En esta parte se declarara el tipo de datos o información que se va a manejar, cabe destacar tres:

- Fabricantes
- Cervezas
- Reseñas

Para comprenderlo, se mostrará la parte de los fabricantes ya que las restantes son análogas:

```
struct ManufacturerModel: Codable, Identifiable, Hashable {  
    let id: String  
    var name: String  
    var isImported: Bool  
    var logoImageData: Data?  
    var beers: [Beer]  
  
    init(id: String = UUID().uuidString,  
         name: String,  
         isImported: Bool = false,  
         logoImageData: Data?,  
         beers: [Beer]) {  
  
        self.id = id  
        self.name = name  
        self.isImported = isImported  
        self.logoImageData = logoImageData  
        self.beers = beers  
  
    }  
}
```

Como vemos aparecen todos los datos que vamos a manejar de los fabricantes, así como su debida lista de cervezas, es importante destacar el uso de las propiedades Codable, Hashable e Identifiable, para el correcto funcionamiento de la aplicación.

2.4 Anotaciones

Cabe destacar que han surgido dos problemas durante el desarrollo de la aplicación:

1. No ha sido posible realizar la detección del primer lanzamiento de la aplicación, el código ha quedado comentado ya que siempre detectaba que era la primera vez que se cargaba la aplicación, y por tanto solo cargaba datos del *Bundle* del proyecto.
2. Cada vez que se realiza una modificación en la lista de los fabricantes redirige al usuario a la pantalla principal de los fabricantes, tras muchos intentos tratando de solucionarlo, no ha sido posible, sin embargo, los cambios se realizan correctamente.