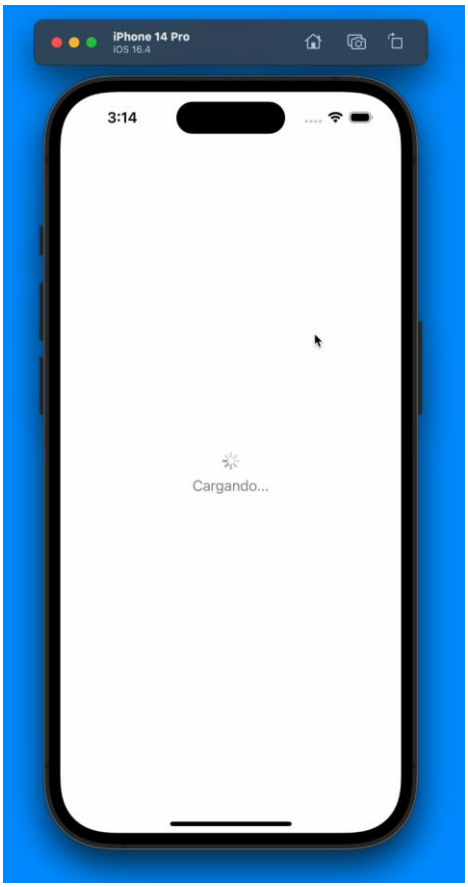
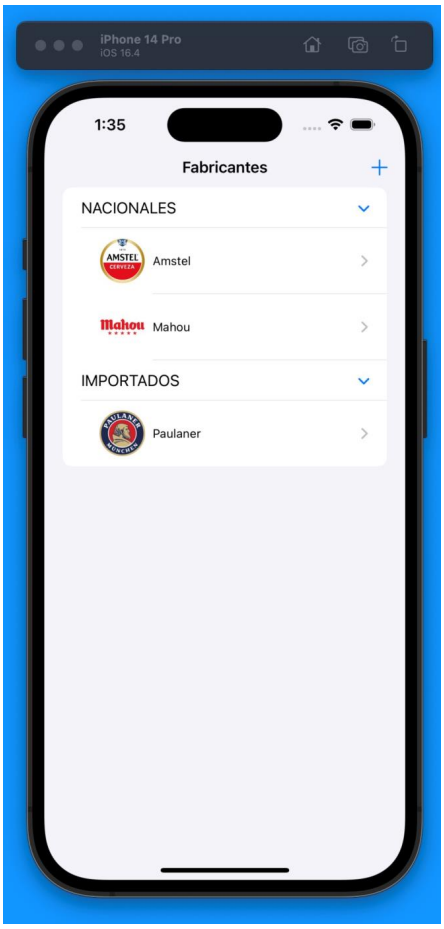
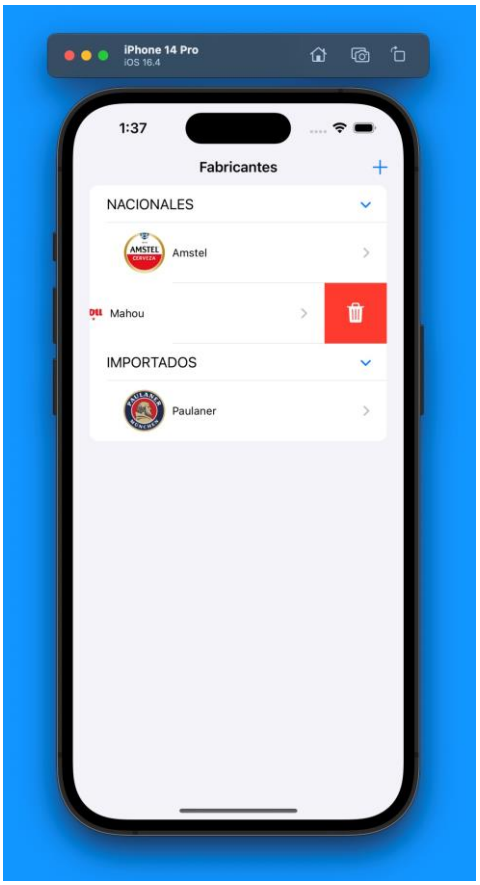
Contenido

Manual de usuario

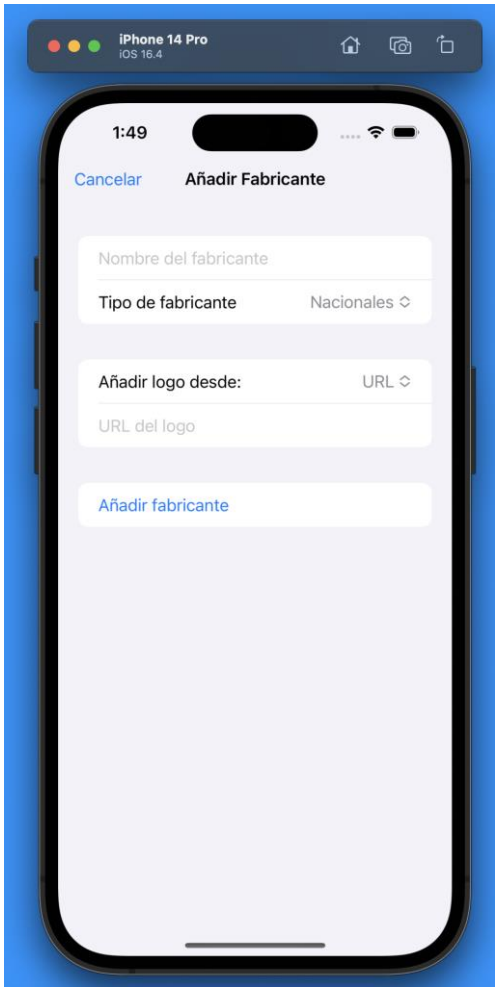
Al ejecutar la aplicación en primer lugar se cargan los distintos fabricantes añadidos, los cuales se dividen en nacionales e importados. Deslizando hacia la izquierda en cualquier fabricante podemos eliminarlo:

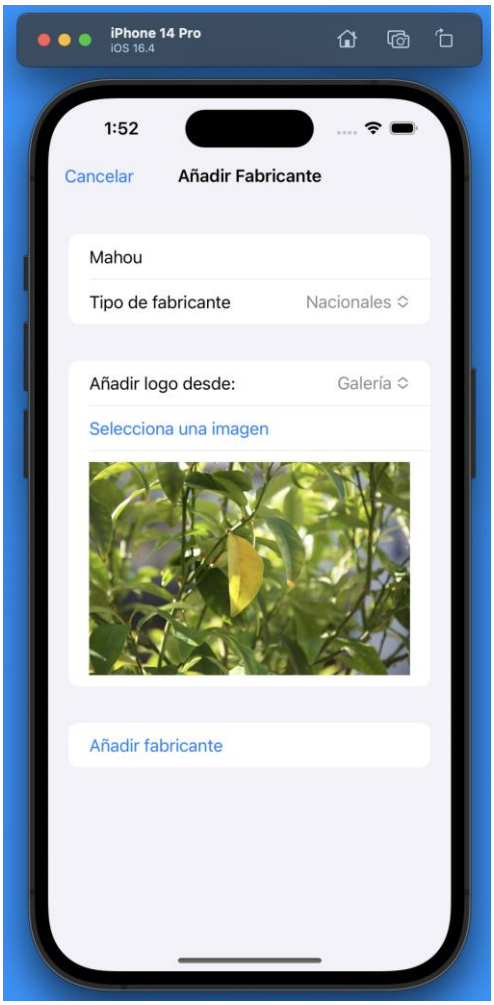




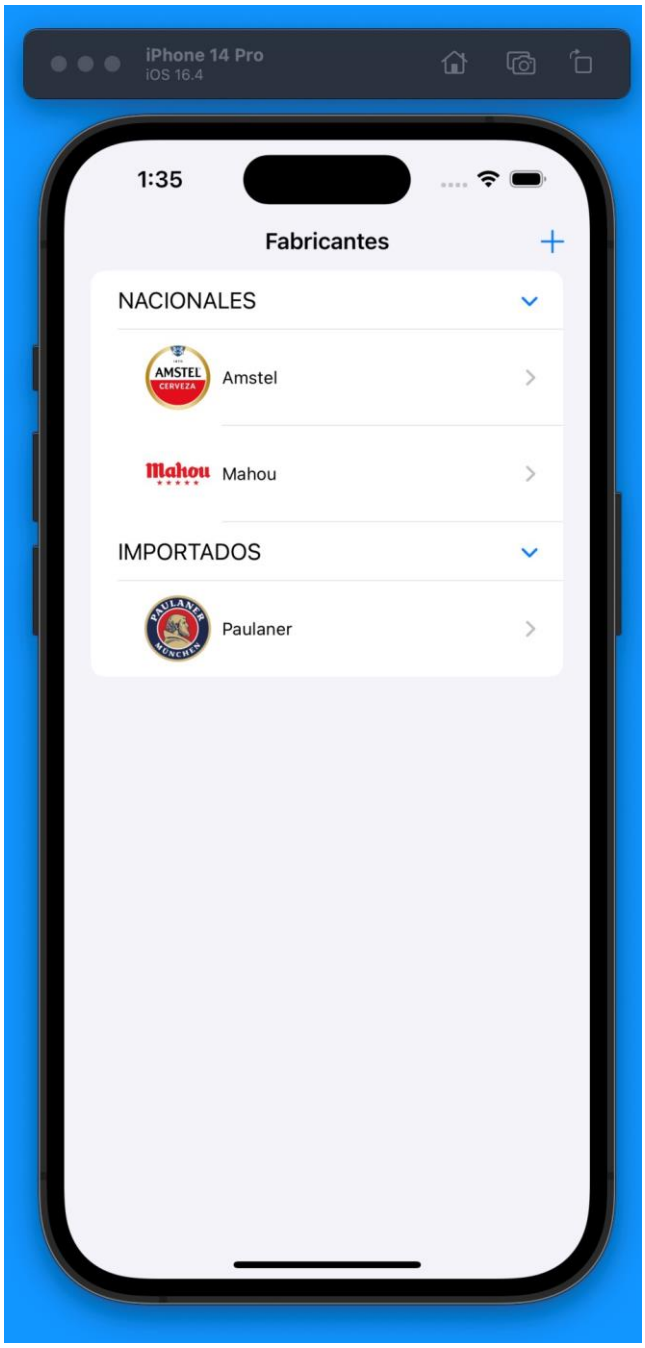


Arriba a la derecha aparece la opción de añadir un nuevo fabricante, al pulsar aparece en la pantalla un formulario para añadirlo, pudiendo elegir el origen de la cerveza y la imagen del logo, la cual se puede añadir mediante url o directamente desde la galería del teléfono:

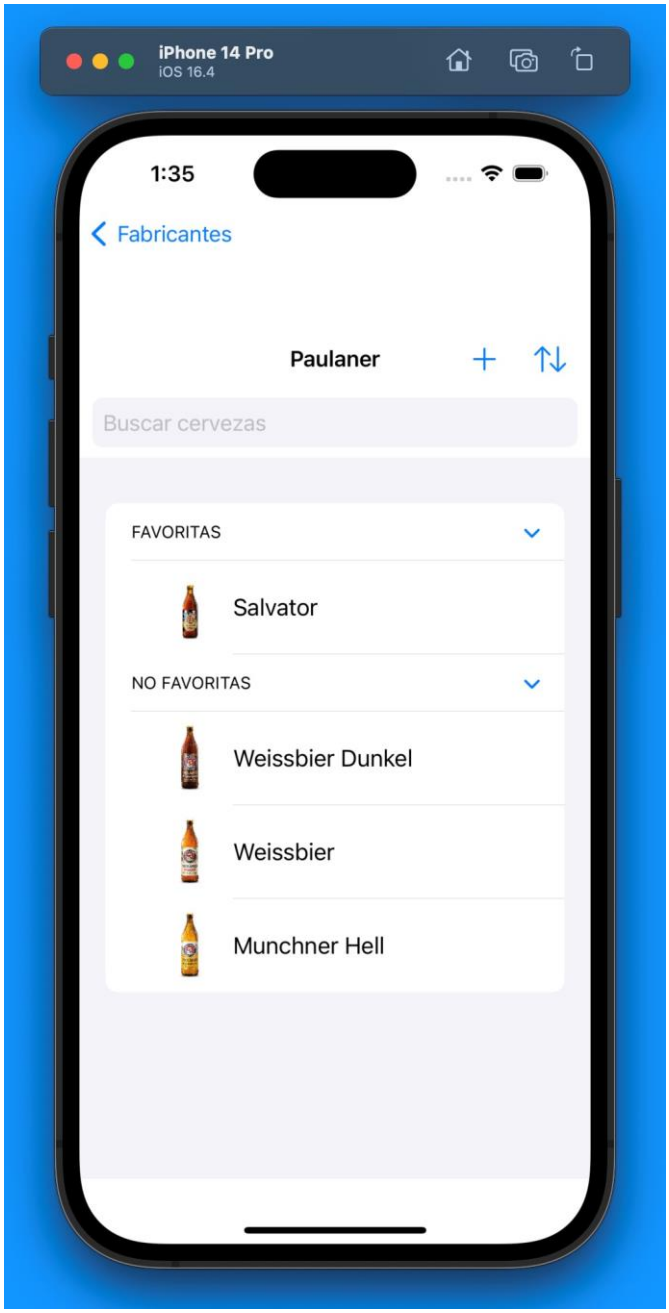




Una vez añadido el fabricante, se actualiza la vista anterior, de modo que aparece el nuevo fabricante creado:

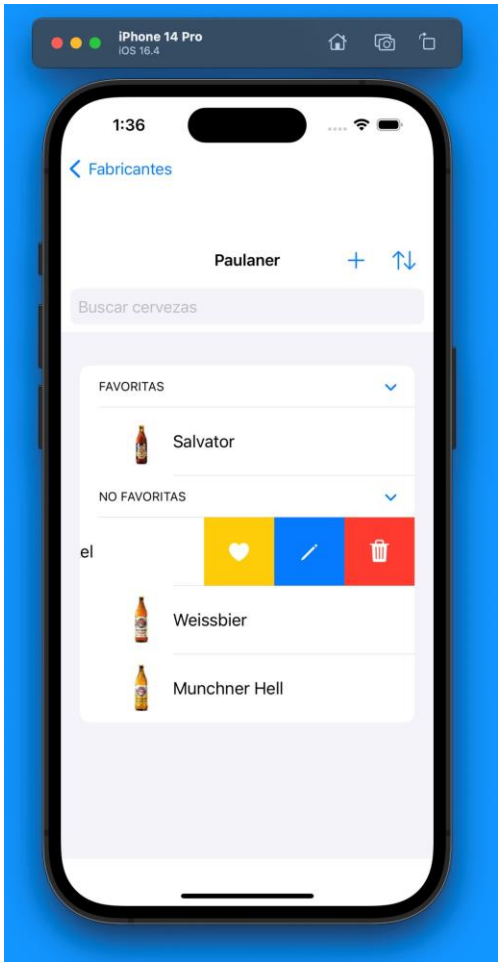


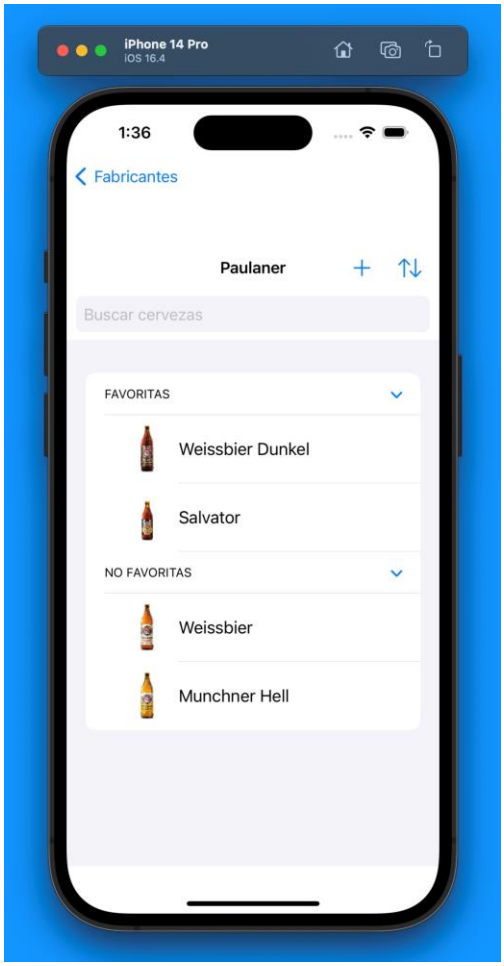
Si pulsamos encima de cualquier fabricante, nos lleva a su vista detallada, donde se encontrarán las distintas cervezas. Estas cervezas están divididas en favoritas y no favoritas. Debajo del nombre de la cerveza tenemos una barra de búsqueda, que nos permitirá filtrar las cervezas por nombre. Al lado del nombre de la cerveza tenemos dos botones, con los que podemos ordenar las cervezas o añadir una nueva:



Al deslizar hacia la izquierda en cualquier cerveza, nos aparecen tres opciones:

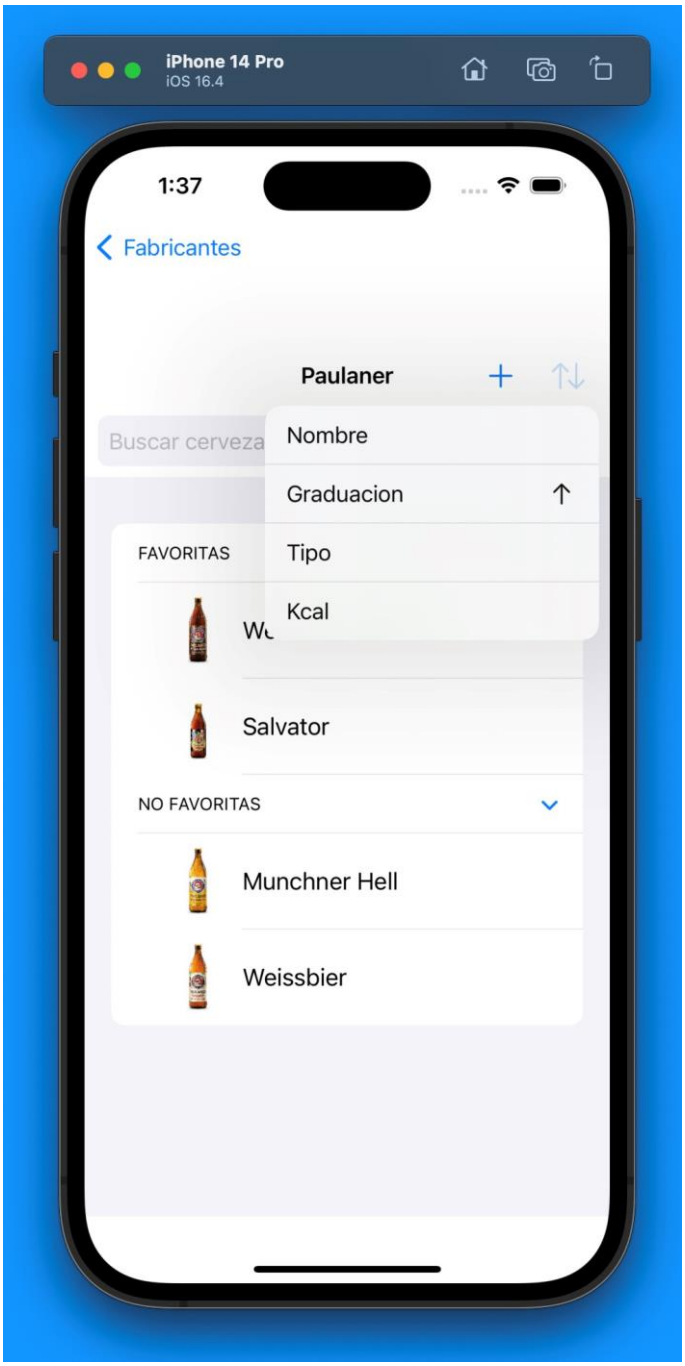
- Eliminar la cerveza (se elimina directamente de la base de datos)
- Editar la cerveza (lleva a una página donde puedes editar los datos de la cerveza)
- Añadir la cerveza a favoritos

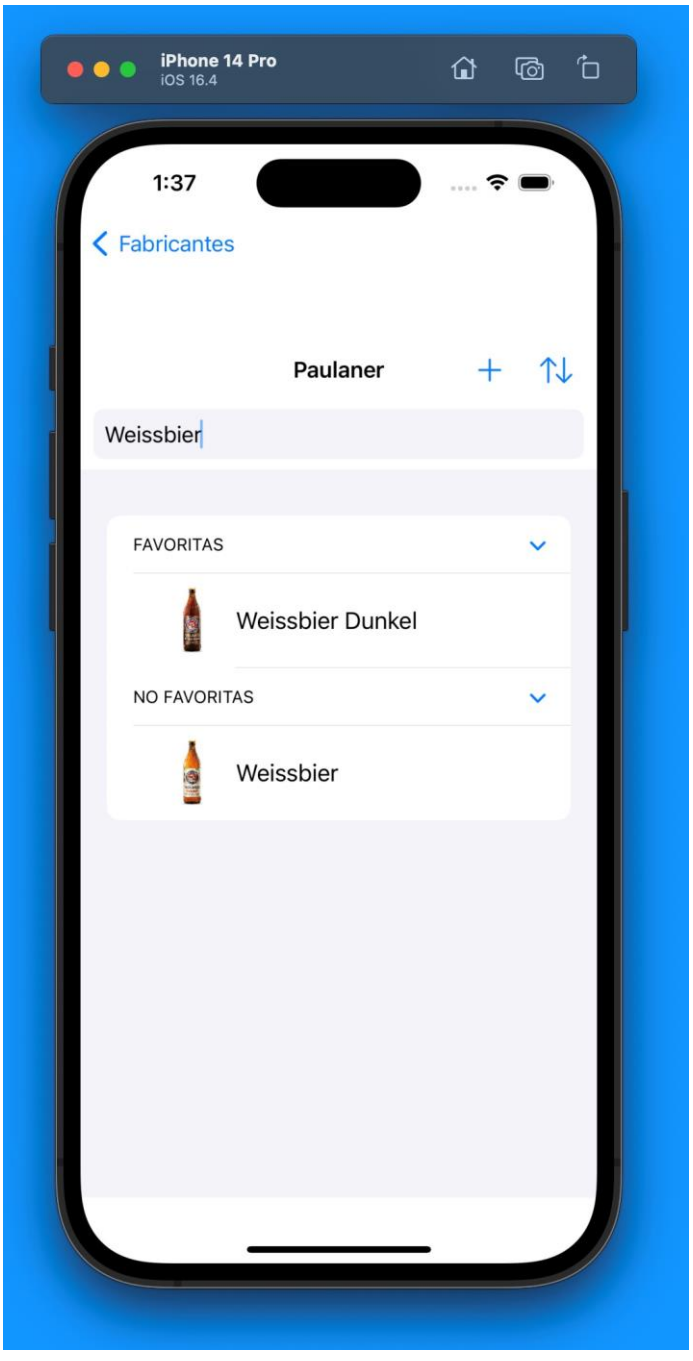




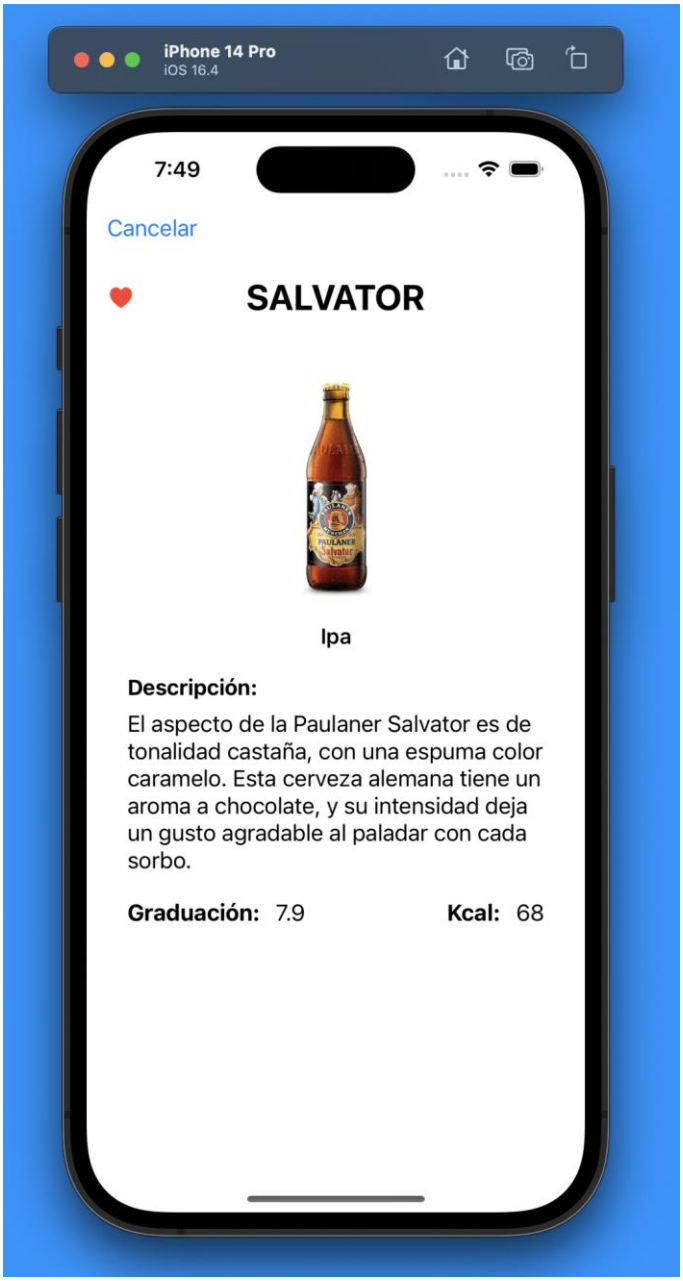
La opción de ordenar nos permite realizar la ordenación tanto de forma ascendente como de forma descendente. Por defecto, cada ordenación se realiza de una forma, pero si volvemos a pulsar en el tipo de ordenación, cambia a la contraria.

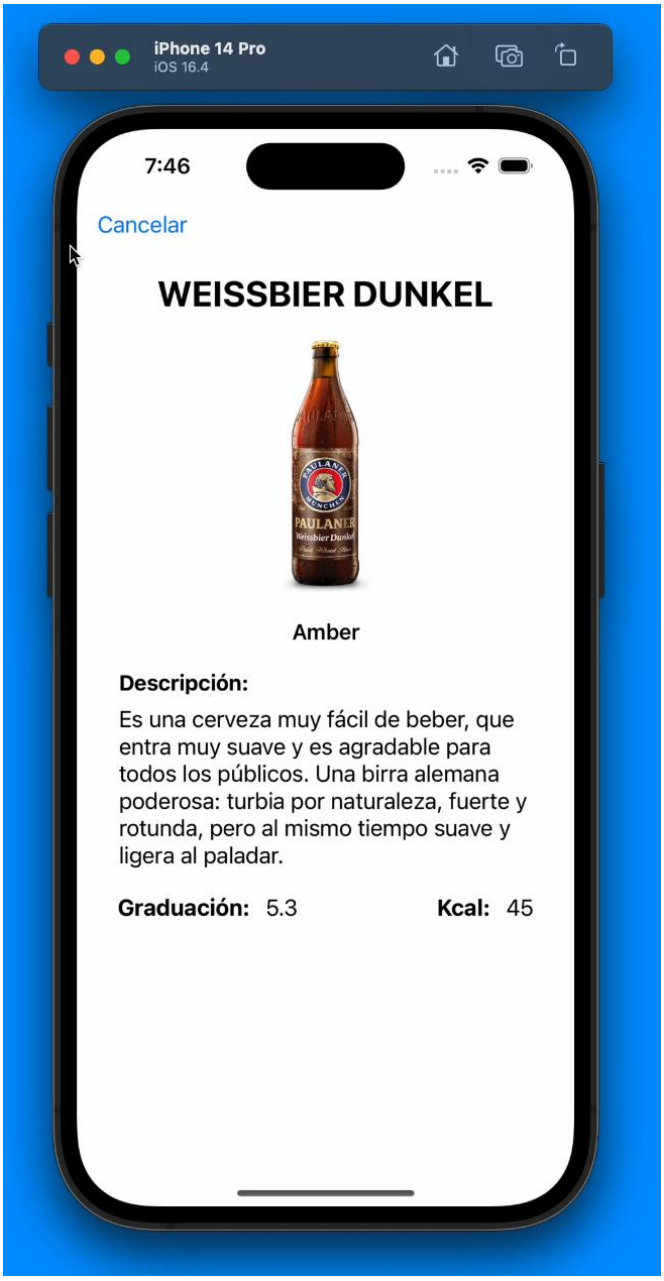
La barra de búsqueda hace que solamente se muestren las cervezas cuyo nombre coincide con el escrito. Este nuevo listado también se puede ordenar.





Por último, si pulsamos en cualquier cerveza, nos lleva a la vista detallada de la cerveza, donde aparece el nombre en forma de título y debajo, la imagen y el tipo de cerveza:





Además, también aparece la descripción, graduación y kcal, y si se trata de una cerveza añadida a favoritos, nos aparece un corazón rojo en la esquina superior izquierda.

Manual del programador

Para el manejo y persistencia de las cervezas y fabricantes se ha utilizado una API REST. Toda la información sobre esta se encuentra en el siguiente repositorio de github.

Pero, en resumidas cuentas, todo se hace mediante peticiones GET (para obtener las cervezas y los fabricantes), POST (para añadir nuevas cervezas o fabricantes), PUT (para modificar una cerveza o añadirla a favoritos) y DELTE (para eliminar alguna cerveza o fabricante de la base de datos). Toda la documentación se encuentra en el swagger de la API.

Todas estas peticiones se realizan en el fichero del código denominado APIService.swift y se relacionan con la interfaz a través de BeerCenterViewModel.swift

Un ejemplo de esto es con la petición de eliminar una cerveza:

```swift
func deleteCerveza(idCerveza: String, completion: @escaping (Result<Bool, Error>) -> Void) {
    let urlString = "\(baseURL)/cerveza/deleteCerveza"
    var components = URLComponents(string: urlString)
    components?.queryItems = [URLQueryItem(name: "id_cerveza", value: idCerveza)]

    guard let url = components?.url else {
        completion(.failure(URLError(.badURL)))
        return
    }

    var request = URLRequest(url: url)
    request.httpMethod = "DELETE"
    request.setValue(apiKey, forHTTPHeaderField: "X-API-KEY")

    URLSession.shared.dataTask(with: request) { _, response, error in
        if let error = error {
            completion(.failure(error))
            return
        }
        if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200 {
            completion(.success(true))
        } else {
            completion(.failure(URLError(.badServerResponse)))
        }
    }.resume()
}
```

Esta función del APIService recibe el id de la cerveza por parámetros, y realiza una petición DELETE a la URL base especificada anteriormente junto con /cerveza/deleteCerveza + el id de la cerveza, de esta forma realiza la petición que elimina la cerveza especifica. Si esta petición tiene éxito (recibe un 200), se comunica, y si no lo tiene, se comunica el error que ha ocurrido.

```swift
func deleteCerveza(cervezaToDelete: Cerveza) {
    APIService.shared.deleteCerveza(idCerveza: cervezaToDelete.id) { result in
        DispatchQueue.main.async {
            switch result {
            case .success:
                print("Cerveza eliminada con éxito.")
            case .failure(let error):
                print("Error al eliminar cerveza: \(error)")
            }
        }
    }
}
```

Esta imagen contiene la función de deleteCerveza del viewmodel, que simplemente llama al APIService, pasándole como parámetro el id de la cerveza que ha recibido, e imprime el resultado (imprimiendo el error, si es que ha ocurrido alguno o un mensaje de éxito si ha salido todo correctamente).

```swift
CervezaRow(cerveza: cerveza) {
    selectedCerveza = cerveza
}
.swipeActions(edge: .trailing, allowsFullSwipe: true) {
    Button(role: .destructive) {
        deleteCerveza(cerveza)
    } label: {
        Label("Eliminar", systemImage: "trash")
    }
```

```swift
private func deleteCerveza(_ cerveza: Cerveza) {
    viewModel.deleteCerveza(cervezaToDelete: cerveza)
}
```

Este código muestra como se llama desde la vista a la función correspondiente del viewmodel, pasándole como parámetro la cerveza que se quiere eliminar.

Por otra parte, para añadir una imagen como URL o como imagen desde la galería se utiliza un picker:

```
Section {
    Picker("Añadir logo desde:", selection: $logoOption) {
        Text("URL").tag(LogoOption.url)
        Text("Galería").tag(LogoOption.upload)
    }

if logoOption == .url {
    TextField("URL del logo", text: $logo)
} else {
    Button("Selecciona una imagen") {
        isImagePickerPresented = true
    }
    if let selectedImage = selectedImage {
        Image(uiImage: selectedImage)
            .resizable()
            .scaledToFit()
    }
}
```

Y se almacena la cadena como una URL o si en su defecto, se ha elegido una imagen de la galería, se almacena como una cadena en base64 (comprimiendo su calidad):

```
let logoRepresentation: String
if logoOption == .upload, let selectedImage = selectedImage {
    logoRepresentation = selectedImage.jpegData(compressionQuality: 0.1)?.base64EncodedString() ?? ""
} else {
    logoRepresentation = logo
}
```

Posteriormente, para mostrar la imagen de las cervezas, se utiliza el siguiente código:

```
private var logoView: some View {
    if let logoURL = cerveza.logoURL {
        if let url = URL(string: logoURL), logoURL.isValidURL {
            ZStack {
                Color.white
                AsyncImage(url: url) { phase in
                    switch phase {
                    case .empty:
                        ProgressView()
                    case .success(let image):
                        image.resizable().aspectRatio(contentMode: .fit)
                    case .failure:
                        fallbackImage
                    @unknown default:
                        EmptyView()
                    }
                }
            }
        } else if let imageData = Data(base64Encoded: logoURL), let uiImage = UIImage(data: imageData) {
            ZStack {
                Color.white
                Image(uiImage: uiImage)
                    .resizable()
                    .aspectRatio(contentMode: .fit)
            }
        } else {
            fallbackImage
        }
    } else {
        fallbackImage
    }
}

private var fallbackImage: some View {
    ZStack {
        Color.white
        Image(systemName: "photo").foregroundColor(.gray)
    }
}
```

Que se encarga de mostrar la imagen de una forma u otra dependiendo si es una URL o una imagen en base64. También se encarga de añadir una imagen por defecto en caso de que no se pueda acceder a la imagen.

Para ello se utilizan la siguiente extensión:

```
extension String {
    var isValidURL: Bool {
        let detector = try? NSDataDetector(types: NSTextCheckingResult.CheckingType.link.rawValue)
        if let match = detector?.firstMatch(in: self, options: [], range: NSRange(location: 0, length: self.endIndex.utf16Offset(in: self))) {
            return match.range.length == self.endIndex.utf16Offset(in: self)
        } else {
            return false
        }
    }
}
```

Que se encarga de detectar si la imagen es una URL o una cadena en base64.