UNIVERSIDAD DE GRANADA

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Programación de dispositivos móviles: App 1

Bar2go

José Alberto García Collado
26513007-X
joseegc10@correo.ugr.es

2020-2021

Índice

1 Explicación de la app

Bar2go es una app orientada a la búsqueda y reserva de bares. En particular, actualmente se permite la búsqueda por nombre del bar, por tipo de comida y por cercanía. En cuanto a la reserva, se permite observar como está el bar de lleno en el día de hoy y se permite la reserva para el día siguiente en 4 turnos, dos al mediodía y dos por la noche.

Además, se ha incluido un inicio de sesión necesario para poder observar las reservas y los bares del usuario. Una vez iniciamos sesión nos encontramos con una vista de introducción en la que se explica la funcionalidad de la app y se le pide al usuario que introduzca el número de personas del que va a constar la reserva.

Para la prueba de la App se puede crear una cuenta propia donde realizar las reservas y crear los bares que se requiera.

Por último, también se permite ver la ubicación de los bares en un mapa, para que el usuario sepa donde se encuentran.

2 Componentes de la app

La app se ha construido para el sistema operativo ios, a partir de la biblioteca de Apple SwiftUI, siguiendo el paradigma recomendado por ellos, el MVVM (Model - View - ViewModel). La parte servidora se ha simulado con la base de datos Cloud Firestore de Firebase.

2.1 Model

En el modelo encontramos las estructuras de datos en las que se va a pasar la aplicación. En concreto, vamos a encontrar tres:

- Un enumerado para los tipos del bar (carne, pescado, pasta...)
- Una estructura para el bar, que va a contener el id, el nombre, descripción, capacidad total, capacidad disponible para el turno, ubicación, tipo de bar, el id del usuario que creó el bar, el directorio donde se encuentra la imagen del bar en el storage de Firebase y la propia imagen del bar.
- Por último, una estructura que representa una reserva, la cual contiene el id de la reserva, el id del usuario que la hizo, el bar donde se ha hecho la reserva, el tamaño de la reserva, el turno y la fecha.

2.2 View

En cuanto a las vistas, como son demasiadas, menciono las carpetas en las que se ha dividido.

La primera es **Principales**. Esta carpeta contiene las vistas principales de la app, como es la vista de login, de introducción (inicio), la vista principal una vez elegimos el número de personas de la reserva (home) y el ContentView o vista principal desde la que se genera el resto de vistas.

La segunda es **Elementos**. Esta carpeta contiene vistas que son partes pequeñas de otras vistas y que se utilizan en varias, como puede ser la barra de navegación o el selector de imagen del bar.

La tercera es **Bar**. Esta contiene todo lo relacionado con las partes del bar, como es la carta de Bar, la selección del tipo de bar, el formulario para crear/editar el bar, etc.

La cuarta es **búsquedas**. Esta contiene las vistas de las diferentes búsquedas que puede hacer el usuario, ya sea por nombre, por distancia, etc.

La quinta y última es **Map**. Esta contiene todo lo relacionado con la selección de la ubicación del bar y mostrar la ubicación del bar y del usuario.

Por último, comentar que hay 3 vistas en las cuales me he basado en diseños encontrados por internet. El enlace a la página donde me he basado se puede encontrar en el código, al principio del todo en los comentarios iniciales de cada archivo.

2.3 ViewModel

Por último, tenemos las clases que nos conectan las estructuras de datos con las vistas, aportando la funcionalidad necesaria.

En primer lugar, nos encontramos con dos archivos en los cuales no se ha hecho uso de la biblioteca SwiftUI, sino de la biblioteca UIKit. Esto es debido a que SwiftUI es una biblioteca muy reciente y no tiene incorporadas todas las funcionalidades todavía.

El primer archivo se trata de **UserLocation** y simplemente tiene como atributos un manejador de localización, que nos permite establecer los componenentes de la localización, como la accuracy deseada, y un atributo Published locations, en el que vamos a almacenar la localización del usuario cuando se llame al metodo locationManager.

El segundo archivo se trata de **ImagePicker** y este se encarga de proporcionar la selección de la imagen para el bar. En este caso, este archivo tiene una estructura muy fija, la cual por desconocimiento de la biblioteca UIKit (ya que mi aplicación se basa en SwiftUI), se ha obtenido de la documentación de Apple practicamente en su totalidad, con algún pequeño cambio para adaptarla a mi código, pues la clase en sí no puede modificarse mucho.

Posteriormente, volviendo a la biblioteca SwiftUI, nos encontramos con la clase **ImageViewModel**, la cual se encarga de hacer la conexión con el Storage de Firebase y poder guardar la imagen. La variable data contiene la imagen en sí, la variable imageURL

contiene la dirección donde se va a guardar en el Storage y el método load se encarga de dicho cometido.

La clase **SpeechViewModel** se encarga de pasar el texto a audio. Para ello hace uso de un synthesizer, que es la clase de SwiftUI la cual se encarga de la transcripción y el Utterance, el cual digamos que contiene las propiedades del texto, es decir, el texto en sí, el idioma, el delay, etc. Por tanto, en el método speak, establecemos los atributos y llamamos al método speak del synthesizer pasándole el utterance como parámetro.

2.3.1 BarViewModel

Por último, nos encontramos con la clase más importante, **BarViewModel**. Esta clase se encarga de realizar la conexión con la base de datos, ya sea para hacer el login o para trabajar con los bares y las reservas (obteniéndolas o guardándolas).

Los atributos de esta clase son tres. El primero es **bares**, un array de bares donde guardaremos los bares que consulte el usuario. El segundo es **reservas**, un array de reservas donde guardaremos las reservas del usuario y el tercero es un enumerado **estado** el cual nos dice si el usuario tiene que loguearse o si ya está logueado.

Los dos primeros métodos de la clase son **login** y **register**. Estos reciben el email y contraseña del usuario y lo loguean o lo registran según lo que haya seleccionado.

El siguiente método es para añadir un bar, **aniadeBar**, y recibe el bar a añadir y la imagen de dicho bar.

Después, tenemos dos métodos **docEnBar**, que se encargan de pasar el json devuelto por Firebase a un objeto del Modelo BarModel. El parámetro que reciben varía en función de lo que devuelva el método de la biblioteca de Firebase, y devuelve el objeto de tipo BarModel.

Posteriomente, tenemos un método **obtenerBares**, que se conecta a la colección bares de la base de datos y los obtiene todos, y un método **obtenerBaresUsuario** que realiza lo mismo que el anterior pero estableciendo en la búsqueda el campo idUser al id del usuario que esta logueado.

Ahora, encontramos el método **obtenerReservasUsuario**, que se conecta a la colección reservas de la base de datos y obtiene las reservas cuyo idUser coincide con el id del usuario que esta logueado.

Después, está el método **eliminarReserva**, el cual recibe como parámetro la reserva a eliminar, realizando esta eliminación a través de su id.

El siguiente método es **obtenerBaresConReservasUsuario**, y este hace una petición a la colección reservas para obtener las reservas del usuario, y posteriormente hace una petición a la base de datos obteniendo los bares involucrados en dichas reservas.

Ahora encontramos el método **eliminarBar**, el cual recibe el id del bar a eliminar y la dirección de la imagen del bar en el storage. Este método elimina el bar de la colección bares a través de su id y elimina la imagen del storage a través de la dirección.

Posteriormente, encontramos dos métodos para editar un bar, **editarBar** y **editaBarConImagen**. El primero recibe solo el bar a editar, haciendo una petición a la colección bares del bar que recibimos y actualizando su información. En el segundo método recibimos además la nueva imagen, realizando entonces el cambio de imagen en la dirección de imagen almacenada en el BarModel.

El siguiente método que encontramos es el de obtener las reservas de un bar en el día de hoy, **obtenerReservasBar**, para saber su estado. Este hace una petición a la colección reservas estableciendo el campo idBar al que recibimos como paramétro y el campo fecha a la del día de hoy.

Después, encontramos un método para reservar una mesa, **reservaMesa**, el cual recibe como parámetros el tamaño de la reserva, el bar reservado y el turno del día. Este hace dos cosas, primero modifica el campo capacidadTurno del bar en el que se hace la reserva, reduciéndolo en la fecha de mañana y en el turno que haya introducido, en la cantidad de personas de la reserva. Más tarde, creamos una nueva entidad en la colección reservas, estableciendo los distintos campos de la reserva.

Por último, encontramos tres métodos que se corresponden con las búsquedas de bares. El primero, **obtenerBaresNombre**, recibe el nombre buscado y se encarga de obtener todos los bares y quedarse con los cuales el nombre buscado esté contenido en el nombre del bar. El siguiente método es **obtenerBaresTipo**, y este hace una petición a la colección bares en función del campo tipo que se recibe como parámetro. El último método es **obtenerBaresDistancia**, el cual recibe la posición del usuario y que se encarga de obtener todos los bares y ordenarlos en función de la cercanía al usuario por distancia euclídea.