UNIVERSIDAD DE GRANADA

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Programación de dispositivos móviles: App 2

FileChange

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

2020-2021

Índice

1 Explicación de la app

FileChange es una app orientada a que los usuarios puedan compartir los archivos entre ellos, existiendo diferentes categorías, ya sean libros, poemas, revistas, apuntes o cualquier otra cosa. Los usuarios pueden subir sus propios archivos y descargas los que han subido otros usuarios, pudiendo valorar los archivos que se descargan una vez se vuelve a entrar a la app.

Además, se ha incluido un inicio de sesión necesario para que el usuario pueda recopilar sus propios archivos, los archivos que ha descargado y una valoración de estos últimos. Comentar también que la app se ha orientado a documentos de texto, por lo que actualmente solo se permite la subida de archivo de tipo pdf.

En cuanto a las búsquedas disponibles, el usuario puede hacer una búsqueda donde seleccione el tipo y la cadena que debe contener el título, puede hacer una búsqueda directamente por tipo y también existe un sistema de recomendación en la página principal donde se le recomiendan al usuario los archivos que puede ser que le gusten. Esta recomendación la he basado en la valoración del archivo y el número de veces que el usuario ha descargado un archivo del mismo tipo.

También comentar que se ha generado un diagrama de clases con la ayuda de la herramienta obtenida en el siguiente **enlace**. Esta me ha generado automáticamente el diagrama de clases permitiendo hacer los cambios que se requieran. El diagrama de clases se puede encontrar en la carpeta classDiagram.

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 dos:

- Una estructura para el archivo, que va a contener el id, el título, descripción, la dirección donde se va a encontrar la portada en el Storage, la dirección donde se va a encontrar el archivo en el Storage, la valoración media del archivo, el número de valoraciones que tiene el archivo, el id del usuario que lo ha creado, el tipo de archivo que es y el nombre del autor del archivo.
- Una estructura que representa un usuario, la cual contiene el id del usuario, su nombre, sus apellidos, su correo, las valoraciones que tiene pendientes, las

descargas que ha hecho y un diccionario para saber el número de descargas que ha hecho de cada tipo.

Además, tenemos un enumerado en el que recogemos los tipos posibles que pueden tener los archivos. Estos son:

- Libro
- Comic
- Poema
- Revista
- Apuntes
- Otro

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 registro, la vista de carga, el menú inferior donde se selecciona entre diferentes vistas, la vista de búsqueda, etc.

La segunda es **Extras**. 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, el selector de tipo, la vista de valoración de un archivo, etc.

La tercera es **File**. Esta contiene todo lo relacionado con los archivos, como la vista de un archivo, la vista de creación y edición de un archivo, la vista en la que podemos ver varios archivos, etc.

Por último, comentar que hay un archivo de vista en el cual me he basado en código encontrado 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. Este archivo es NavigationBarColor y se puede encontrar en la carpeta Extras. Básicamente consiste en un modificador de vista, permitiéndome modificar el color de la barra de navegación, la cual SwiftUI por defecto solo nos deja poner en blanco.

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 **FilePicker** y este se encarga de proporcionar la selección del archivo. 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 ayuda de internet, con algún pequeño cambio para adaptarla a mi código, pues la clase en sí no puede modificarse mucho ya que es muy fija.

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 al igual que el archivo anterior, por lo que se ha obtenido con la ayuda de internet al ser siempre igual.

Posteriormente, volviendo a la biblioteca SwiftUI, nos encontramos con la clase **FirebaseAuth**, la cual se encarga de hacer la conexión con Firebare para permitir la autenticación del usuario. Esta dispone de un atributo Published (para que se pueda compartir entre varias vistas) que representa el estado del usuario. Este es un enumerado que puede valer login si se encuentra por loguearse, register si se encuentra por registrarse, logueado si ya se ha logueado, valoraciones si esta logueado pero tiene valoraciones pendientes e inicio si no se encuentra en ninguna de los anteriores (en función de variables de usuario se actualizará con un valor u otro en cada inicio).

Esta clase FirebaseAuth tiene dos métodos. El primero permite el login del usuario y recibe el email y la contraseña como parámetros. El segundo es para el registro del usuario y también recibe el email y la contraseña.

La clase **DocumentViewModel** se encarga de realizar la conexión con el Storage de Firebase para obtener los documentos, es decir, tanto para obtener las imágenes como los archivos. Esta contiene dos atributos, el primero para almacenar el documento en si y el segundo para almacenar la dirección al documento en el Storage y el cual se recibe como parámetro en el constructor. Esta clase tiene un método load, el cual se encarga de hacer la petición al Storage del documento almacenado en la dirección que nos indica el atributo y se encarga de guardarlo en nuestra variable data para que pueda ser accedido por otras vistas.

2.3.1 FileViewModel

Por último, nos encontramos con la clase más importante, **FileViewModel**. Esta clase se encarga de realizar la conexión con la base de datos, ya sea para obtener información de usuarios u obtener los archivos en sí.

Los atributos de esta clase son:

- files: Para almacenar los archivos en función de ningún criterio.

- misArchivos: Para almacenar los archivos del usuario.
- misDescargas: Para almacenar las descargas del usuario.
- filesValorar: Para almacenar los archivos que están pendientes de valoración por parte del usuario.
- filesRecomendados: Para almacenar los archivos que se recomiendan al usuario.
- user: Para almacenar la información del usuario.
- file: Para almacenar un archivo.
- filesTipo: diccionario para almacenar los archivos en función de cada tipo.

El primer método de la clase es **addFile**. Este método recibe el archivo a añadir y se encarga de subir el archivo a Firestore.

El segundo método es addUserInfo y recibe el usuario a añadir y se encarga de añadirlo a Firestore.

El tercer método es docEnUser, el cual recibe un diccionario y un id y se encarga de transformar estos parámetros en un objeto del struct UserModel.

El siguiente método es getUserInfo y este se encarga de obtener la información que existe en Firestore sobre el usuario actual.

Después nos encontramos dos métodos llamados docEnFile que buscar crear un objeto del struct FileModel, el primero a partir de un QueryDocumentSnapshot y el segundo a partir de un diccionario y de un id.

Ahora tenemos el método getUserFiles. Este recibe como parámetro una cadena, la cual puede ser nil también, para obtener los archivos del usuario cuya cadena esta contenida en su título en caso de que la cadena no sea nil. En otro caso, obtiene todos los archivos del usuario.

El siguiente método es getUserDownloads y tiene el mismo funcionamiento que el método anterior, es decir, recibe una cadena que puede valer nil, pero en este caso obtiene los archivos descargados por el usuario.

Después tenemos el método editFile, el cual recibe un archivo y se encarga de modificar la información almacenada de dicho archivo. También existe otro método de edición, editFileWithImage, el cual edita además de la información del archivo en sí, la imagen de su portada almacenada en el Storage.

Ahora tenemos un método para eliminar un archivo, deleteFile. Este recibe el archivo a eliminar, eliminando su información en Firestore, la imagen de su portada y el archivo en sí.

También existe en esta clase un método para obtener un archivo desde el Storage, getFile, el cual recibe la url del archivo.

El siguiente método es busquedaFiles, el cual recibe el nombre buscado y el tipo buscado y obtiene todos los archivos de dicho tipo quedándose con aquellos cuyo nombre buscado este contenido en su título. También existe el método busquedaFilesTipo, el cual solamente obtiene los archivos del tipo que se recibe como parámetro.

Después está el método aniadeDescarga, cuyos parámetros son el id del archivo y el tipo. Este método primero obtiene las descargas del usuario y ve si ya está almacenado el archivo en esas descargas. Si ya está almacenado no hace nada, pero en otro caso añade el archivo a las descargas del usuario y a las valoraciones pendientes y aumenta el contador del número de descargas del usuario del tipo que recibimos. Esto último va a servirnos después cuando queramos obtener los archivos recomendados para un usuario.

También existe un método para obtener los archivos a valorar, getFilesAValorar, el cual se encarga de obtener todos los archivos que el usuario necesita valorar. Como se almacenan los ids de los archivos a valorar cuando añadimos una descarga simplemente tenemos que hacer la petición de dichos archivos.

Ahora nos encontramos con el método deleteFileAValorar y se encarga de eliminar de la lista de valoraciones pendientes el id del archivo que recibimos como parámetro.

El siguiente método es aniadeValoración y recibe el id del archivo a valorar y la valoración. Básicamente añade esta valoración a la valoración media que ya tiene.

Después tenemos un método actualizaFilesTipo que lo único que hace es almacenar en la variable files los archivos del tipo que recibimos como parámetro almacenado en el atributo filesTipo, el cual era un diccionario que almacenaba los archivos en función del tipo.

Por último, tenemos al método getFilesRecomendados, el cual recibe una cadena que puede valer nil. Para obtener los archivos recomendados al usuario, primero se ordenan por valoración media, desempatando en caso de misma valoración por el número de descargas que ha hecho el usuario de archivos del mismo tipo. Si la cadena es distinto de nil solo nos quedamos con aquellos archivos cuya cadena está contenida en el título.