**Desarrollar una aplicación Android que trate la cuenta de usuario de dropbox como una biblioteca de libros remota, de forma que:**

**1. Exista una pantalla inicial de login, donde el usuario pueda introducir sus credenciales para tener acceso a su cuenta Dropbox. (**[**https://www.dropbox.com/**](https://www.dropbox.com/)**)**

Para poder realizar este paso es necesario trabajar con la API que nos ofrece Dropbox. Lo primero que tendremos que hacer es:

* Desde *App Console* y tras loguearme con mi usuario y contraseña de Dropbox, elegimos crear una app de tipo Dropbox API.
* Elijo el tipo de datos que necesito almacenar. En mi caso "Files and datastores".
* En el siguiente paso indico que mi app necesita acceder a los archivos de todo Dropbox.
* A continuación escojo el tipo de archivos a los que necesito acceder. En este caso archivos específicos.
* Selecciono el tipo de archivo específico: eBooks.
* Finalmente proporciono un nombre para mi app.

Una vez completados todos los pasos Dropbox me proporciona una ***app key*** y una ***app secret***. Estas ''claves'' son las que tendré que utilizar para autenticar mi app desde Android.

Siguiendo los pasos proporcionados por Dropbox developers, en concreto siguiendo el [tutorial](https://www.dropbox.com/developers/core/start/android) que nos ofrecen para Android, implemento el código necesario en mi clase Main.java, la cual extiende de Activity.

Hecho esto, cuando ejecuto mi aplicación puedo ver por pantalla la pantalla de loguin de Dropbox, donde debo introducir mi usuario y contraseña.

**2. Una vez introducidos los credenciales, se mostrarán en pantalla todos los libros que el usuario disponga en su cuenta Dropbox en forma de biblioteca. Esto es, todos los ficheros que tengan extensión .epub . (Para representar cada libro, puede usarse un icono genérico).**

Una vez he accedido a Dropbox, y siguiendo en mi clase Main.java, a través de mi variable *mApi* (del tipo DropboxAPI<AndroidAuthSession>), obtengo los metadatos del directorio principal y los guardo en *dir*, una variable de tipo *Entry* .

Dir es mi directorio principal, el cual tengo que recorrer en busca de archivos .epub. Para ello creo un método, al que le paso este directorio (dir), y cuyo funcionamiento es el siguiente:

Para cada uno de los archivos que contiene este directorio:

* Si el archivo es un directorio:
  + Obtengo el nombre del directorio.
  + Almaceno los metadatos de ese directorio en una variable de tipo Entry.
  + Si el directorio no está vacío, entonces vuelvo a llamar al método pasándole este nuevo directorio.
* Si el archivo es un fichero:
  + Miro si tiene extensión .epub.
  + Si es así, lo almaceno en libros, un ArrayList de tipo Entry.

Una vez tengo este ArrayList con todos los libros, voy a sacar información de cada uno de ellos. Para ello voy a crear dos métodos:

* *nameEbooks*: método que recorre el ArrayList de libros y almacena en un array de tipo String el nombre de cada uno. Devuelve un array de tipo String.
* *revEbooks*: método que recorre el ArrayList de libros y guarda en un array de tipo String la fecha de creación/modificación de cada uno de los libros. Devuelve un array de tipo String.

Estos dos arrays los paso mediante un Intent a la siguiente Actividad (Biblioteca.java), desde donde voy a mostrar mis libros por pantalla.

Aquí recojo la información del Intent, nombres y fechas de cada uno de los libros, y me creo un nuevo ArrayList llamado *eBooks* de tipo String[], donde cada elemento está compuesto por el nombre y la fecha de creación del libro.

A continuación creo otro ArrayList llamdo *items* de tipo Item. Cada elemento del ArrayList contiene una imagen genérica de libro y la información de cada libro, la cual obtengo del ArrayList eBooks.

Creado este ArrayList de ítems, y mediante un adaptador, se los asigno a un ListView. Este ListView me mostrará por pantalla todos los libros que tengo almacenados en Dropbox.

**3. Dicha pantalla tendrá un menú desplegable con dos opciones, una de ellas ordenará la lista de libros por el nombre del archivo y la otra por fecha de creación. (Idealmente nos gustaría ordenar por el título del libro en vez del nombre del fichero).**

Esta parte es implementada mediante un Spinner que contendrá dos opciones:

* Ordenar por título.
* Ordenar por fecha de creación.

Si selecciono la primera opción, se ejecutará el método reordenarPorNombre(). Si selecciono el segundo, reordenarPorFecha(). Ambos métodos se encargan de hacer lo siguiente:

* Recorrer toda la lista de ebooks.
* Para el eBook ‘x’ y para el eBook ‘x+1’, comparo sus nombres/fechas para ir ordenándolos alfabéticamente.
* Una vez ordenados actualizo el ListView.

**4. Si el usuario clickea dos veces sobre el icono genérico que representa cada libro, la aplicación mostrará la portada del libro clickEado.**

***SIN IMPLEMENTAR. SÓLO SIMULACIÓN***

Actualmente hace lo que se detalla:

* Al clickear sobre un libro, nos dirigiremos a una nueva Actividad (Portada.java). A esta Actividad le vamos a pasar el nombre del libro.
* Especifico la url de la imagen que quiero descargar.
* Me descargo la imagen y la decodifico en un Bitmap.
* Asigno la imagen a un ImageView.
* Muestro mediante un Toast el nombre del libro.

***IDEA PENSADA, PERO SIN IMPLEMENTAR***

*Mediante la API Google Custom Search:*

* Conseguir una clave: <https://code.google.com/apis/console/?api=customsearch&pli=1#welcome>
* Crear un motor de búsqueda personalizado: <http://www.google.com/cse/>, del cual obtendremos un ID único.
* Realizar una petición a una URL que deberemos componer nosotros y que está compuesta por:
  + https://www.googleapis.com/customsearch/v1
  + **key**=CLAVE
  + **cx**=ID\_DEL\_MOTOR\_DE\_BÚSQUEDA
  + **q**=”lo que queramos buscar”
  + **type=**image
* Recoger los resultados devueltos mediante un objeto JSON.
* De todos ellos nos quedaríamos con el primer elemento.
* De él obtendría el link
* La ruta devuelta y almacena en link sería la ruta específica de la imagen que quiero descargar.

**Nota**: Por lo que pude observar casi siempre la primera imagen se correspondía con la portada de los libros, aunque tampoco tendría por qué cumplirse en todos los casos.