DESARROLLO DE INTERFACES

PRÁCTICA 2. VISOR DE IMÁGENES PAGINADO

Vas a desarrollar una aplicación en Flutter que cargue imágenes desde assets locales y las muestre en pantalla dentro de un visor navegable usando gestos verticales sobre la pantalla. La aplicación tendrá, además, dos botones transparentes con iconos.

Tendrás que tratar los datos de un fichero que contiene una lista de mapas con información sobre la imagen: ruta, subtítulo, likes y views de la imagen. El fichero se adjunta, así como las imágenes que se deben mostrar.

Se valorará la estructura de directorios del proyecto, así que lleva un orden al crear ficheros de código.

1. Crea un nuevo proyecto de Flutter, **prac2_abc**, cambiando 'abc' por las iniciales de tu nombre.
2. Descarga de Aules los ficheros dispuestos para esta práctica.
   - Crea en la raíz del proyecto (**NO DENTRO DE lib**) los directorios **assets/images** y guarda ahí las imágenes descargadas.
   - Guarda el fichero **local_image_posts.dart** dentro de **lib/shared/data**
3. En **pubspec.yaml**, quita las almohadillas que comentan el apartado de assets e indica la ruta al directorio donde has metido las imágenes, como en la figura:

```
# To add assets to your applicatio
assets:
  - assets/images/
#   - images/a_dot_ham.jpeg
```

4. Borra el contenido de main.dart y **genera el código de una nueva aplicación MaterialApp**. Elimina el rótulo "DEBUG" rojo mostrado en la esquina superior derecha al ejecutar. Usa el atributo title para poner tu nombre y primer apellido. Asigna un Scaffold a su parámetro home. Mete un Text centrado en el cuerpo del Scaffold, con texto arbitrario.
5. Crea un fichero para configurar el tema de la aplicación. En él, **crea una clase** con un único método, que al llamarlo devuelva un ThemeData con el modo oscuro (nocturno) activado. Usa este método en main.dart para cambiar el tema de la aplicación.
6. La entidad de dominio en la que se basa la lógica de negocio de esta aplicación es una clase que contiene los siguientes cuatro atributos:
    - subt: String con un pequeño subtítulo que acompaña a la imagen
    - url: String con la ruta relativa a la imagen en assets
    - likes: el número de personas que indicaron su gusto por la imagen
    - views: el número de personas que vieron la imagen

    Crea la clase de la entidad de dominio en un fichero aparte. Llama a la clase **ImagePost**. Escribe el constructor de esta clase, con parámetros nombrados y haciendo obligatorios los dos primeros.
7. **Crea la pantalla principal de la aplicación** en un fichero aparte. Llama a la clase **DiscoverScreen**, que será un StatelessWidget que contendrá, de momento, un Container con un Text centrado. Asígnalo al body del Scaffold en main.dart.
8. Añade al proyecto la **dependencia con el paquete Provider**, para poder gestionar el estado de la aplicación. **Crea la clase DiscoverProvider**, que heredará de **ChangeNotifier**. Los atributos de esta clase-estado son:
    - Una lista de objetos de la entidad ImagePost, inicialmente vacía.
    - Un booleano isCargado = false, que indica que no hay imágenes en la lista cuando la app esté recién instalada.

    Envolver el MaterialApp con **ChangeNotifierProvider**.
9. Mapear los datos entre fuente y entidad:
    - Crear un fichero nuevo para la **clase ModeloImagenLocal**
    - Los atributos de la clase serán los mismos que en los mapas de la lista del fichero local_image_posts.dart, y con los mismos nombres.
    - Crea un **constructor básico** con parámetros nombrados, un **método factory** que devuelva una instancia a partir de un Map de los de la fuente (fichero) y un **método toImagePost()** para convertir en nuestra entidad de dominio.

- Dentro del método factory, contemplar la posibilidad de que todos los elementos del mapa de entrada sean nulos (excepto la URL), proponiendo valores alternativos en tal caso.

10. En DiscoverProvider, crea un **método cargaListaImagePosts()** que inicialice el estado con los datos de la lista de mapas local:
    - Insertar una demora de 3 segundos.
    - Utilizando los métodos de ModeloImagenLocal, mapear cada elemento de imagePosts (la lista de mapas) para obtener un objeto de la entidad ImagePost y añadirlo a la lista de estas entidades que hay en el estado (y que inicialmente está vacía).
    - Modifica el valor de la bandera booleana isCargado.
    - Notifica a los suscriptores.
11. Será necesario llamar a este método nada más iniciar la aplicación y obtener el estado. En main.dart, usa la instanciación de DiscoverProvider para llamar a cargaListaImagenPosts(). Puedes hacerlo con un bloque de instrucciones o con una única instrucción en cascada.

    En DiscoverScreen, toma del estado la longitud de la lista y muéstrala en el Text. Debería salir un 8. Aumenta el tamaño de fuente si es necesario.
12. En DiscoverScreen, elimina o comenta el Text centrado y, en su lugar, asigna al Container:
    - un **CircularProgressIndicator** si la lista de imágenes del estado todavía no está cargada.
    - un Placeholder si ya está cargada.

    Reinicia (hot restart) para comprobar. Si no se ve el indicador circular, aumenta la demora en el método del estado.
13. Crea un **widget nuevo** en un fichero aparte y llámalo **ImageScrollableView**, sin estado. Este widget reemplaza al Placeholder dentro del Container de DiscoverScreen. La clase del widget:
    - Atributo: una lista de ImagePost
    - Constructor con la lista de ImagePost obligatoria
    - Su método build() devolverá un **PageView.builder**. Se trata de un widget similar a ListView.builder, pero sus elementos se muestran página a página. Cambia su scrollDirection a vertical; itemCount estará limitado a la longitud de la lista de ImagePost (atributo). Finalmente, el método de su itemBuilder devolverá un Image.asset a partir de la URL del elemento correspondiente de la lista.

    Vincula este ImageScrollableView con DiscoverScreen, metiéndolo en el Container cuando la lista esté cargada. Necesita como entrada la lista de ImagePost del estado.
14. Expande la imagen: modifica el parámetro fit del Image.asset y envuélvela primero en un Expanded y luego en un Column.

15. Envuelve el Column con un **Stack**. Este widget permite apilar widgets en el eje Z. En su parámetro children mete, además del Column, un Text que muestre el subtítulo de la imagen correspondiente.
    Envuelve el Text con un **Positioned**. Utiliza sus parámetros left y bottom para situarlo en una posición aceptable.

16. Envuelve el Column en un Scaffold. Crea un nuevo widget ButtonColumn, stateless, que contendrá una columna con dos botones flotantes, con fondo transparente y elevación 0.0. Los iconos serán, respectivamente, favourite y remove_red_eye, con los colores modificados. Los botones deben estar alineados en la parte baja de la pantalla y tener cierta separación vertical entre ellos.
    Asigna ButtonColumn al parámetro floatingActionButton del Scaffold.

Para la entrega:

Comprimir todo el proyecto, EXCEPTO las carpetas assets y build, en un fichero .zip.

Entregar en la tarea de Aules