Creando nuestra primer App con Flutter (2da Parte)

Diseñando y Desarrollando aplicaciones con Flutter

Índice de la guía

Ésta guía está basada en el tutorial “Write Your First Flutter App, part 2 “ que ofrece el sitio oficial de Flutter.dev.

Pasos:

Aviso

Haremos uso como base lo que construimos en la primer parte de este tutorial durante la Clase 1.

Ante cualquier duda, en la Clase_1> Creando_Primer_App_Primer_Parte > Version Final , está el proyecto el cual podemos usar como base para las futuras modificaciones que le haremos.

Paso 1 - Añadiendo iconos a la lista

En este paso, añadiremos un Icono de un corazón en cada Item de la Lista. Lo que haremos será que una vez que apretemos sobre el, guardaremos esa palabra en una lista de favoritos.

Para conseguirlo, añadiremos una variable de tipo ***Set*** el cual guardará las palabras que el usuario agregue a favoritos. Elegimos ***Set*** debido a que no permite elementos repetidos.

```
class RandomWordsState extends State<RandomWords>{

  final _sugerencias = <WordPair>[];
  final _fuenteLista = const TextStyle(fontSize: 18.0);
  final _favoritos = Set<WordPair>();
```

Paso 1 - Añadiendo iconos a la lista

En la función ***_buildPalabra()*** ,añadiremos una variable ***bool*** que controla si la palabra a construir está en la lista de favoritos, porque en función de eso haremos algo con el icono.

```
Widget _buildPalabra(WordPair palabra) {

  final bool isInFavoritos = _favoritos.contains(palabra);

  return Column(                              /*5*/
    children: <Widget>[
    ListTile(
      title: Text(
        palabra.asPascalCase,                 /*6*/
        style: _fuenteLista,                  /*7*/
      ), // Text
    ), // ListTile
    Divider(),                                /*8*/
  ], // <Widget>[]
  ); // Column
}
```

Paso 1 - Añadiendo iconos a la lista

En la misma función añadiremos un ***Icon*** con forma de corazón al ***ListTile*** para habilitar los favoritos. En el paso que le sigue a este, le daremos acciones a ese ***Icon.***

```
Widget _buildPalabra(WordPair palabra) {

  final bool isInFavoritos = _favoritos.contains(palabra);          /* 1 */

  return Column(
    children: <Widget>[
    ListTile(
      title: Text(
        palabra.asPascalCase,
        style: _fuenteLista,
      ), // Text

      trailing: Icon(                                               /* 2 */
        isInFavoritos ? Icons.favorite : Icons.favorite_border,     /* 3 */
        color: isInFavoritos ? Colors.red : null,                   /* 4 */
      ), // Icon
    ), // ListTile
    Divider(),
    ], // <Widget>[]
  ); // Column
}
```

Paso 2 - Añadiendo interacción

En este paso haremos los iconos que sean touch. Cuando el usuario alguna de las palabras en la Lista, lo que pasará es que se añadirá a la lista y se mostrará el corazón relleno de color rojo. Si estaba ya en la lista de favoritos, cuando apretemos esa palabra, se removerá de la lista de favoritos y se modificará el icono al corazón sin relleno.

Para lograrlo el ***ListTile*** tiene un atributo llamado ***onTap*** el cual espera una función como valor. En el cuerpo de esa función deberemos especificar el caso mencionado anteriormente.

Paso 2 - Añadiendo interacción

Como vemos, se definió una función anónima , la cual en el cuerpo se llama al ***setState()*** el cual también como parámetro espera una función donde éste método es propio de ***Stateful Widget,*** donde lo que hace es notificar al framework que el estado ha cambiado y por lo tanto tiene que redibujar la pantalla.

```
onTap: (){
  setState(() {
    if(isInFavoritos)
      _favoritos.remove(palabra);
    else
      _favoritos.add(palabra);
  });
},
```

Dentro de la función que le pasamos como parámetro, controlamos si está en favoritos, la removemos de la lista de favoritos, y caso contrario, añadimos la palabra a la lista de favoritos.

** Para ver el funcionamiento, se sugiere probar simplemente poner la lógica sin llamar a ***setState***, y podrán visualizar que la pantalla no se redibuja.

Paso 2 - Añadiendo interacción

Ante cualquier duda en el directorio /InforSanLuis-Flutter/Clases/Clase_2/Pasos-2daParte se encuentran los códigos para validar o intentar resolver dudas.

Paso 3 - Refactorizando el build de RandomWordsState

Antes de comenzar con el paso 3, moveremos el ***Scaffold*** que está en el atributo ***home*** del ***MaterialApp***, al ***build()*** de ***RandomWordsState.***

Una vez que lo movamos, en el atributo ***home*** del ***MaterialApp***, ahora llamaremos a ***RandomWords().***

Luego, en el atributo ***body*** del ***Scaffold,*** reemplazamos lo que estaba por el método que definimos ***_buildSugerencias()*** . Esto lo estamos haciendo para ordenar un poco la UI en código.

Paso 3 - Refactorizando el build de RandomWordsState

```dart
class RandomWordsState extends State<RandomWords>{

  final _sugerencias = <WordPair>[];
  final _fuenteLista = const TextStyle(fontSize: 18.0);
  final _favoritos = Set<WordPair>();



  @override
  Widget build(BuildContext context) {

    return Scaffold(
      appBar: AppBar(
        title: Text('Taller InforSanLuis-Flutter 2019'),
      ), // AppBar
      body: _buildSugerencias()
    ); // Scaffold
  }
```

```dart
class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Bienvenido a Flutter',
      home: RandomWords()
    ); // MaterialApp
  }
}
```

Paso 3 - Navegando a otra pantalla

En este paso, añadiremos una nueva pantalla, (***Route*** en Flutter) que muestre los favoritos.

En Flutter, el ***Navigator*** administra un stack conteniendo las rutas de la app. Si hacemos un *push* de una ruta dentro del stack del ***Navigator***, actualiza la pantalla mostrando la nueva ruta. Si hacemos un *pop*, retornará la ruta anterior.

En las siguientes diapositivas, paso a paso lograremos mostrar un icono de una lista, donde si el usuario aprieta en ese icono, lo llevara a la pantalla que mostrará sólo las palabras agregadas a favoritos.

Paso 3 - Navegando a otra pantalla

Al ***AppBar*** de la pantalla, le agregaremos un ***IconButton*** al atributo ***actions*** que tiene el ***AppBar*** . Este atributo espera una Lista de Widgets, por eso nos puede extrañar la notación. Lo que nos permite, es agregar más de una acción posible, de una manera muy simple.

En el código a continuación se ve claramente cómo lograr agregar el botón con el icono de la lista.

En ***onPressed*** ***(***atributo del ***IconButton)***, espera una función, la cual definiremos dentro de **RandomWordsState,** la cual llamaremos ***_pushView()*** que retorna tipo ***void*** que por ahora tendrá el cuerpo vacío, pero en los pasos siguientes, ahí definiremos como movernos a la próxima pantalla.

Paso 3 - Navegando a otra pantalla

Haciendo un ***hot reload***, veremos el icono de la Lista en el AppBar. Todavía no le hemos dado funcionalidad, solo poder visualizarlo.

```
appBar: AppBar(
  title: Text('Taller InforSanLuis-Flutter 2019'),

  actions: <Widget>[ // Add 3 lines from here...
    IconButton(icon: Icon(Icons.list), onPressed: _pushView),
  ], // <Widget>[]
), // AppBar
```

```
void _pushView(){

}
```

Paso 3 - Navegando a otra pantalla

Ahora dentro de ***_pushView()*** construiremos una ruta y la “pushearemos” al stack del ***Navigator***. Ésta acción nos mueve desde la vista actual a la nueva vista.

El contenido para la nueva vista, lo haremos dentro de un ***constructor de MaterialPageRoute***, en una función anónima. Es una manera de hacerlo, existen otras de lograr lo mismo.

Paso 3 - Navegando a otra pantalla

Llamaremos a ***Navigator.push*** como se muestra en el código, que hace un push al stack del ***Navigator.*** La IDE nos tirara error, pero lo arreglaremos en las próximas diapositivas.

Lo próximo es añadir la ***MaterialPageRoute*** y su constructor. Por ahora añadiremos el código que genera los **ListTiles** con las palabras de favoritos, y entre cada **ListTile** poner un divisor horizontal.

Paso 3 - Navegando a otra pantalla

Lograremos este paso haciendo uso de una variable ***tiles*** donde a este le pasaremos el resultado de *"mapear"* o aplicarle a cada palabra en ***_favoritos***, el encerrarla en un ***ListTile*** y poniendole la fuente correspondiente.

Luego de ello, en la variable ***divided*** a la lista de **ListTile** que formamos en ***tiles***, le añadimos un separador con el método ***divideTiles()*** que provee la clase ***ListTile***, el objeto resultante de ese método, lo transformamos en una Lista de **Widget** mediante el método de las colecciones ***toList()***.

Paso 3 - Navegando a otra pantalla

Hasta aquí hemos añadido la lógica mencionada en la diapositiva anterior. Ahora resta devolver

Ahor, vamos a retornar un ***Scaffold***, que contenga una ***ListView*** con la lista de Widget ***divided*** en su ***body***. Como título a la pantalla le daremos “Lista de Favoritos”.

```
void _pushView() {
    Navigator.of(context)
        .push(MaterialPageRoute(builder: (BuildContext context) {

            final Iterable<ListTile> tiles = _favoritos.map(

                // 1° Mapeando a cada palabra, el encerrarla en un ListTile
                    (WordPair palabra) {
                    return ListTile(
                    └── title: Text(
                            palabra.asPascalCase,
                            style: _fuenteLista,
                        ), // Text
                    ); // ListTile
                });

                // 2° Armamos la lista de widget con los separadores
            final List<Widget> divided = ListTile.divideTiles(
                context: context ,
                tiles: tiles
            ).toList();
```

Paso 3 - Navegando a otra pantalla

En este paso en particular está el código de como quedaría finalmente la función.

Vemos el ***Scaffold*** donde en su ***body*** tiene como hijos a ***divided*** y como título “Lista de Favoritos “

```
void _pushView() {
    Navigator.of(context)
        .push(MaterialPageRoute(builder: (BuildContext context) {

            final Iterable<ListTile> tiles = _favoritos.map(

                // 1° Mapeando a cada palabra, el encerrarla en un ListTile
                (WordPair palabra) {
                return ListTile(
                    title: Text(
                            palabra.asPascalCase,
                            style: _fuenteLista,
                        ), // Text
                ); // ListTile
                });

                // 2° Armamos la lista de widget con los separadores
            final List<Widget> divided = ListTile.divideTiles(
                context: context ,
                tiles: tiles
            ).toList();


            return Scaffold(
                appBar: AppBar(
                    title: Text("Lista de Favoritos"),
                ), // AppBar
                body: ListView(children: divided,),
            ); // Scaffold
        }) // MaterialPageRoute
    );
}
```

Paso 3 - Navegando a otra pantalla

- Notar que cuando hagamos un **hot reload** veremos que cuando ingresemos a la nueva pantalla, automáticamente nos va a salir un “Back button” para volver a la pantalla anterior. Lo que hace este “Back Button” por detrás es hacer un llamado al método ***pop()*** de **Navigator.**

**Ante cualquier duda en el directorio /InforSanLuis-Flutter/Clases/Clase_2/Pasos-2daParte se encuentran los códigos para validar o intentar resolver dudas.

Paso 4 - Cambiando la UI con Theme

En este paso, modificaremos el tema de la aplicación. El tema controla el “look and feel” de la app. Puede usar el tema predeterminado, que depende del dispositivo físico o el emulador, o personalizar el tema para reflejar su marca.

Puede cambiar fácilmente el tema de una aplicación configurando la clase ***ThemeData***. Actualmente, ésta aplicación utiliza el tema predeterminado, pero cambiará el color primario de la aplicación a blanco.

Paso 4 - Cambiando la UI con Theme

```
class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Bienvenido a Flutter',
            home: RandomWords(),

            //Agregando finalmente el ThemeData
            theme: ThemeData(
                primaryColor: Colors.white,
            ), // ThemeData
        ); // MaterialApp
    }
}
```

Hemos terminado !

Acá concluye el tutorial, este mismo se basó del tutorial provisto por Google “Write Your First Flutter App, part 2” , del siguiente link https://codelabs.developers.google.com/codelabs/first-flutter-app-pt2/#6.

Cualquier inquietud o duda, consulten a cualquiera de los colaboradores.

Esperamos que les haya sido útil !