StreamController, sink, add y StreamBuilder

camilabs

27 de Mayo de 2019

Aprovecharemos lo aprendido en nuestras primeras clases sobre Streams para crear un Stream personalizado.

Durante el curso estuvimos utilizando los Streams que nos proveía Firebase, pero es momento de crear un Stream propio. Esto surge por la necesidad de implementar la muestra de los detalles de cada Place.

En la última vista de nuestra aplicación mostrabamos un texto genérico refiriéndonos a las Bahamas.

Por la estructura que tiene nuestra aplicación en la ventana `HomeTrips()` no tenemos accesibles los datos ni de la lista de lugares para la descripción ni de la descripción para la lista, ambos se encuentran en archivos independientes.

```
class HomeTrips extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Stack(
```

```
      children: <Widget>[
        ListView(
          children: <Widget>[
            DescriptionPlace("Bahamas", 4, descriptionDummy),
            ReviewList()

          ],
        ),
        HeaderAppBar()
      ],
    );
  }

}
```

Lo que haremos con la definición del StreamController es así como con Firebase poníamos alerta el Widget StreamBuilder y este siempre estaba pendiente de reflejar los cambios en la interfaz de usuario, ahora necesitamos reflejar siempre los datos del Place seleccionado en `DescriptionPlace` de tal forma que no tengamos que pasar los datos como parámetro sino que este automáticamente los refleje.

1. Comenzaremos en el archivo `bloc_user.dart`

- Primero definiremos el tipo de elemento que será escuchado por el StreamController, para nuestro caso es un Place seleccionado, por eso escribiremos el siguiente código:

```
StreamController placeSelectedStreamController =
StreamController();
```

- Luego necesitaremos convertir este StreamController en algo que pueda leer el StreamBuilder en su parámetro `stream` por eso lo transformaremos de esta forma:

```
Stream get placeSelectedStream =>
placeSelectedStreamController.stream;
```

- Ahora dejaremos disponible el Stream para que se le pueda insertar el elemento Place, esto lo hacemos através del método `sink` que solo se puede implementar al StreamController.

```
StreamSink get placeSelectedSink =>
placeSelectedStreamController.sink;
```

Una vez que todo está declarado en el archivo Bloc que mantiene una persistencia Global en la aplicación, ya podemos ingresar el objeto Place cuando lo necesitemos.

Mira el archivo completo aquí

2. Vamos al archivo `card_image_list.dart` para añadir la funcionalidad de al dar click en la Foto podamos mostrar los detalles

- Para lograr que la *Card* sea clickeable utilizaremos un Widget nuevo `GestureDetector()`. Este widget es muy útil pues básicamente hace clickeable cualquier área y además nos provee de un método onTap que nos permitirá controlar el comportamiento.
  Nuestra lista ser verá envuelta por este widget de esta forma:

```
return ListView(
      padding: EdgeInsets.all(25.0),
      scrollDirection: Axis.horizontal,
      children: places.map((place){
        return GestureDetector(
          onTap: (){
            print("CLICK PLACE: ${place.name}");
            userBloc.placeSelectedSink.add(place);
          },
          child: CardImageWithFabIcon(
            pathImage: place.urlImage,
            width: 300.0,
            height: 250.0,
            left: 20.0,
            iconData: place.liked?iconDataLiked:iconDataLike,
            onPressedFabIcon: (){
              setLiked(place);
            },
            internet: true,
          ),
        );
      }).toList(),
);
```

- Ahora es importante no dejar independiente el evento *onPressed* del botón like, pues en la descripción también estaremos mostrando la cantidad de likes que tiene el Place, así que en el método `setLiked` implementaremos la lógica de

incremento o decremento de likes del place, para que este dato también sea persistente en la descripción.

```
void setLiked(Place place){
      setState(() {
        place.liked = !place.liked;
        userBloc.likePlace(place, widget.user.uid);
        place.likes = place.liked?place.likes+1:place.likes-1;
        userBloc.placeSelectedSink.add(place);
      });
}
```

Mira el archivo completo aquí

3. Por último en el archivo `description_place.dart` implementaremos el `StreamBuilder` que estará alerta para mostrar los datos del Place una vez que se seleccione.

- Implementemos una instancia de `UserBloc` en el método build así:

```
UserBloc userBloc = BlocProvider.of(context);
```

- Ahora agreguemos el widget StreamBuilder en el return:

```
return StreamBuilder(
      stream: userBloc.placeSelectedStream,
      builder: (BuildContext context, AsyncSnapshot snapshot){

      });
```

- Validaremos el estado de los datos:

```
if (snapshot.hasData) {
    Place place = snapshot.data;
    return Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          titleStars(place),
          descriptionWidget(place.description),
          ButtonPurple(buttonText: "Navigate", onPressed: (){})
        ],
    );
}else {
    return Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Container (
            margin: EdgeInsets.only(
                top: 400.0,
                left: 20.0,
```

```
                right: 20.0
            ),
            child: Text(
              "Selecciona un lugar",
              style: TextStyle(
                  fontFamily: "Lato",
                  fontSize: 30.0,
                  fontWeight: FontWeight.w900
              ),
              textAlign: TextAlign.left,
            ),
          )
        ],
    );
}
```

Notaste que los Widgets ahora están en forma de funciones `titleStars` y `descriptionWidget`:

```
Widget titleStars(Place place){
    return Row (
      children: [
        Container (
          margin: EdgeInsets.only(
              top: 350.0,
              left: 20.0,
              right: 20.0
          ),
          child: Text(
            place.name,
            style: TextStyle(
                fontFamily: "Lato",
                fontSize: 30.0,
                fontWeight: FontWeight.w900
            ),
            textAlign: TextAlign.left,
          ),
        ),
        Container (
          margin: EdgeInsets.only(
              top: 370.0,
          ),
          child: Text(
            "Hearts: ${place.likes}",
            style: TextStyle(
                fontFamily: "Lato",
                fontSize: 18.0,
```

```
                fontWeight: FontWeight.w900,
                color: Colors.amber
            ),
            textAlign: TextAlign.left,
          ),
        ),
      ],
    );
  }

Widget descriptionWidget(String descriptionPlace){
    return Container(
      margin: new EdgeInsets.only(
          top: 20.0,
          left: 20.0,
          right: 20.0
      ),
      child: new Text(
        descriptionPlace,
        style: const TextStyle(
            fontFamily: "Lato",
            fontSize: 16.0,
            fontWeight: FontWeight.bold,
            color: Color(0xFF56575a)
        ),
      ),
    );
  }
```

Mira el archivo completo aquí

Mira el proyecto completo aquí