Insertando y obteniendo referencias en datos de Firestore.

camilabs

27 de Mayo de 2019

En la clase anterior te expliqué cómo estaremos manejando la lógica de los usuarios que dan like a un `Place`. Lo haremos insertándo su referencia en el Place correspondiente.

1. En el archivo `cloud_firestore_api.dart` ya teníamos listo el método `likePlace`

Lo ajustaremos un poco para manejar varias lógicas:

- Necesitamos cambiar el parámetro de entrada `String idPlace` por un objeto `Place place` además como tambien necesitamos insertar la referencia del usuario que le dio like, agregaremos el parámetro `String uid`.

La firma del método debe verse de la siguiente forma:

```
Future likePlace(Place place, String uid) async {}
```

- Ahora necesitamos manejar el incremento o decremento de likes dependiendo de la acción: Like (se agrega uno) Dislike (se resta uno). Hasta el momento en ambos casos estamos incrementando el valor, sustituyamos eso por lo siguiente:

```
_db.collection(PLACES).document(place.id)
    .updateData({
        'likes': place.liked?likes+1:likes-1,
});
```

Notarás que ahora estamos poniendo el id del objeto place.

Ahora validemos si se debe isertar el usuario que le dio like o en el otro caso eliminarlo si le dio dislike, el método completo deberá verse así:

```
Future likePlace(Place place, String uid) async {
    await _db.collection(PLACES).document(place.id).get()
        .then((DocumentSnapshot ds){
       int likes = ds.data["likes"];

       _db.collection(PLACES).document(place.id)
           .updateData({
         'likes': place.liked?likes+1:likes-1,
         'usersLiked':
         place.liked?
         FieldValue.arrayUnion([_db.document("${USERS}/${uid}")]):
         FieldValue.arrayRemove([_db.document("${USERS}/${uid}")])
       });


    });
}
```

Mira el archivo completo aquí

2. Ahora reflejemos el cambio de este método por toda la arquitectura BLoC del proyecto

Ajustaremos los siguientes archivos:

- `cloud_firestore_repository.dart`

  Pongamos la llamada al método así:

```
Future likePlace(Place place, String uid) =>
_cloudFirestoreAPI.likePlace(place,uid);
```

Mira el archivo completo aquí

- `bloc_user.dart`

  Pongamos la llamada al método así:

```
Future likePlace(Place place, String uid) =>
_cloudFirestoreRepository.likePlace(place,uid);
```

Mira el archivo completo aquí

3. Por último regresemos al archivo `card_image_list.dart`

En el método `setLiked()` dentro colocamos `setState()` el cuál maneja el estado de la UI, pongamos la llamada a hacer like y reflejar en la base de datos `userBloc.likePlace(place, widget.user.uid);`

```
void setLiked(Place place){
      setState(() {
        place.liked = !place.liked;
```

```
        userBloc.likePlace(place, widget.user.uid);
      });
}
```

Una vez que ya todo está insertado lo único que nos faltará será reflejar en la UI que el estilo del corazón cambie, para eso pensemos lo siguiente:

Como ahora la responsabilidad de renderear el `CardImageFabIcon` está en el archivo `card_image_list.dart` (justo fue lo que trabajamos en la clase anterior), por lo tanto la responsabilidad del método `buildPlaces` del archivo `cloud_firestore_api.dart` será unicamente de construir una lista de objetos `Places` a partir del snapshot entregado por el stream.

1. La firma del método `buildPlaces` ahora será así:

Se debe añadir el objeto `User` como parámetro el cuál representará los datos del **usario que está logeado**, para entonces comparar y validar con la colección de usuarios que dieron like al `Place` y si alguno coincide. marcarlo como *liked*.

```
List buildPlaces(List placesListSnapshot, User user) {}
```

Dentro quitaremos toda la lógica que correspondía a construir el *Card* en su lugar recorreremos la lista del snapshot y construiremos la lista de objetos `Place`. El método debe verse de la siguiente forma:

```
List buildPlaces(List placesListSnapshot, User user) {
    List places = List();

    placesListSnapshot.forEach((p)  {
      Place place = Place(id: p.documentID, name: p.data["name"],
description: p.data["description"],
          urlImage: p.data["urlImage"],likes: p.data["likes"]
      );
      List usersLikedRefs =  p.data["usersLiked"];
      place.liked = false;
      usersLikedRefs?.forEach((drUL){
        if(user.uid == drUL.documentID){
          place.liked = true;
        }
      });
      places.add(place);
    });
    return places;
}
```

Observa que en esta línea: `List usersLikedRefs = p.data["usersLiked"];` estamos obteniendo la lista de usuarios que dieron like como una lista de referencias, para después recorrerlos si es que existen, eso lo indicamos con el símbolo `?`.

Al recorrerlos comparamos si el `uid` de usuario logeado es igual a alguno de los que están registrados en el `place` y si lo es marcamos el objeto Place que estamos construyendo como liked con el valor de `true`.

Mira el archivo completo aquí

Por último como est método `buildPlaces` es llamado en el Stream hacemos lo siguiente

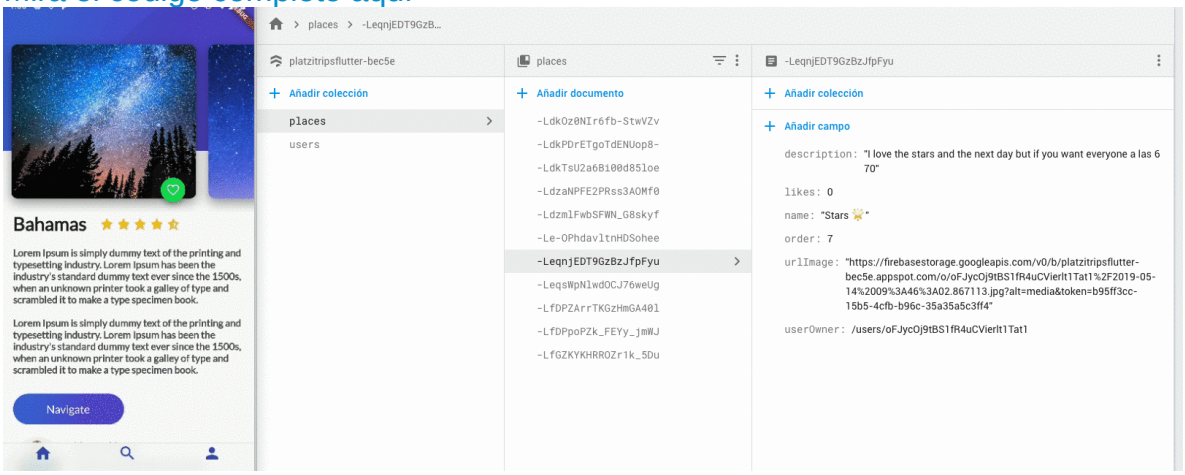
2. Regresemos al archivo `card_image_list.dart` y en los estados adecuados del Stream pongamos el llamado al método y pasemos como parámetro el objeto user.

- Lo hacemos de esta forma en el estado `active` y done:

```
case ConnectionState.active:
  print("PLACESLIST: ACTIVE");
  return
listViewPlaces(userBloc.buildPlaces(snapshot.data.documents,
widget.user));
case ConnectionState.done:
  print("PLACESLIST: DONE");
  return
listViewPlaces(userBloc.buildPlaces(snapshot.data.documents,
widget.user));
```

Mira el archivo completo aquí

Mira el código completo aquí



Ahora vamos a la siguiente clase para hacer que la carga de imágenes sea mucho más ágil.