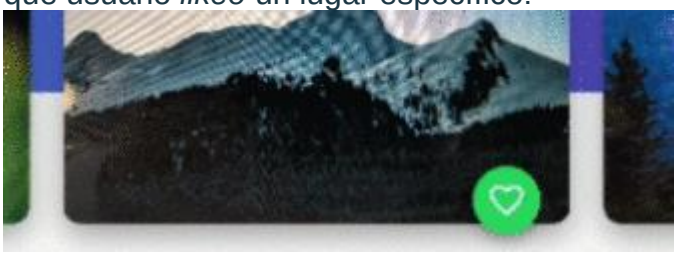
Manejando la lógica de likes, como botón toggle.

camilabs

27 de Mayo de 2019

En la clase anterior viste tan solo una probadita de la magia del Realtime que ofrece Firebase, continuemos implementando la lógica del botón like, definiendo qué usuario *likeo* un lugar específico:



Para implementar la lógica del manejo de likes necesitaremos hacer algunas modificaciones a la estructura de nuestra data, pues particularmente debes almacenar todos los usuarios que hicieron like al Place específico.

Para lograr esto, añadiremos un campo adicional que se llamará `usersLiked` su naturaleza será muy similar al campo `myPlaces` de la entidad usuario, ¿la recuerdas?, **un array de referencias justo como la vemos a continuació:**

```
/users/oFJycOj9tBS1fR4uCVierlt1T2eR
/users/OWaYU2fOIBNLPvK68rzWtAD7w23F
```

No afectaremos la data de Firebase directamente sino que más bien este campo adicional lo implementaremos con código cuando se ejecute el like. Para lograr esta implementación lo primero que debemos hacer es obtener el `uid` del usuario pues, como ves, la referencia se construye con el identificador del nodo. Si necesitamos el `uid` significa que debemos llevar a un nivel superior y con más ámbito de acceso los datos del usuario logeado.

1. Comencemos yendo al archivo `profile_trips_cupertino.dart`

Ubica la opción del tab que corresponde a la pantalla de HomeTrips(), el código que verás será algo como esto:

```
case 0:
    return CupertinoTabView(
        builder: (BuildContext context) => HomeTrips(),
    );
   break;
```

Sustituye el código del parámetro builder con la implementación del BlocProvider, así:

```
case 0:
    return CupertinoTabView(
        builder: (BuildContext context) {
            return BlocProvider(
                    bloc: UserBloc(),
                    child: HomeTrips(),
            );
        },
    );
    break;
```

Mira el código del archivo aquí

2. Ahora vamos al archivo `header_appbar.dart` para manejar el Stream que nos traerá los datos del usuario logeado.

- Crea el método **showPlacesData** a la altura de de la siguiente forma:

```
Widget showPlacesData(AsyncSnapshot snapshot){
    if(!snapshot.hasData || snapshot.hasError){
      return Stack(
        children: [
          GradientBack(height: 250.0),
          Text("Usuario no logeado. Haz Login")
        ],
      );
    } else {
      User user = User(
          uid: snapshot.data.uid,
          name: snapshot.data.displayName,
          email: snapshot.data.email,
          photoURL: snapshot.data.photoUrl
      );

      return Stack(
```

```
        children: [
          GradientBack(height: 250.0),
          CardImageList(user)
        ],
      );
    }
}
```

Notarás que te marca error en la línea donde pasamos como parámetro el objeto `user` a la Clase `CardImageList` no te preocupes lo resolveremos en el paso 3.

- En el return del método **build** implementa el Widget `StreamBuilder()` y el `switch` que nos permite monitorear el estado de la conexión:

```
@override
  Widget build(BuildContext context) {
    // TODO: implement build
    UserBloc userBloc;
    userBloc = BlocProvider.of(context);

    return StreamBuilder(
      stream: userBloc.authStatus,
      builder: (BuildContext context, AsyncSnapshot snapshot){
        switch (snapshot.connectionState){
          case ConnectionState.waiting:
            return Center(child: CircularProgressIndicator(),);
          case ConnectionState.none:
            return Center(child: CircularProgressIndicator(),);
          case ConnectionState.active:
            return showPlacesData(snapshot);
          case ConnectionState.done:
            return showPlacesData(snapshot);
          default:
            return showPlacesData(snapshot);
        }
      },
);
```

Mira el código del archivo aquí

3. Ahora vamos al archivo `card_image_list.dart`

- Este archivo tendrá varios cambios de raíz.
  Empezaremos por modificar la clase de la cual estamos heredando *StatelessWidget* por *StatefulWidget* esto porque para poder controlar correctamente el estado del **botón de like** necesitamos tener acceso al método `setState()`.

Este método está diseñado para ser utilizado cuando la interfaz de usuario sufrirá un cambio a partir de el cambio de estado de algún Widget.
Como es el caso de nuestro botón like, desencadena una serie de efectos en toda la pantalla, nombremoslos a continuación:

1. Incrementar o Decrementar la cantidad de likes en el Place
2. Añadir la referencia del usuario que dió like al Place en el campo qu emencionamos al principio: `usersLiked`
3. Cambiar el ícono del Floating Action Button

En la clase anterior ya hicimos una primera parte del paso 1., ahora hagamos la parte 2. y dejemos la 3 para la siguiente clase.

La estructura del archivo `CardImageList` debe quedar de esta forma:

```
class CardImageList extends StatefulWidget {
    ...

    @override
    State<StatefulWidget> createState() {
        // TODO: implement createState
        return _CardImageList();
    }
}

class _CardImageList extends State {
    @override
    Widget build(BuildContext context) {
        ...
    }
}
```

Como estaremos controlando el estado *liked* de un `Place` es importante que en el archivo `place.dart` añadamos el atributo `bool liked;` y lo añadamos en el constructor junto con el parámetro `id` de esta forma:

```
class Place {

  String id;
  String name;
  String description;
  String urlImage;
  int likes;
  bool liked;
  //User userOwner;
```

```
  Place({
    Key key,
    @required this.name,
    @required this.description,
    @required this.urlImage,
    @required this.likes,
    this.liked,
    this.id
    //@required this.userOwner
  });
}
```

Mira el código del archivo aquí

Regresemos donde estábamos en `card_image_list.dart` pues el siguiente cambio que haremos es al método `listViewPlaces()`, en clases anteriores lo habíamos preparado para que recibiera como parámetro `List placesCard` ya que ahí se ejecutaba todo el render de la construcción de los items de la lista de Places en forma de *Cards,* ahí mismo se encontraba el comportamiento del botón like, sacaremos esto a la clase `CardImageList` para controlar su comportamiento con el método `setState()` que acabamos de ver.

La firma del método debe quedar así:

```
Widget listViewPlaces(List<Place> places){}
```

El Listiview se verá de esta forma:

```
return ListView(
      padding: EdgeInsets.all(25.0),
      scrollDirection: Axis.horizontal,
      children: places.map((place){
        return CardImageWithFabIcon(
          pathImage: place.urlImage,
          width: 300.0,
          height: 250.0,
          left: 20.0,
          iconData: place.liked?iconDataLiked:iconDataLike,
          onPressedFabIcon: (){
            setLiked(place);
          },
          internet: true,
        );
     }).toList(),
```

```
);
```

Con la instrucción `places.map` tomaremos un objeto place a la vez, lo desmembraremos para plasmar sus propiedades en el *Card* para facilitarnos obtener el objeto al que le pertenece el like.

Además, tenemos un nuevo método `setLiked(place);` que está prepardo para recibir el **place clickeado** y modificar su estado, específicamente la propiedad `liked` según sea el caso (true/false). Aquí es precisamente donde albergaremos el método `setState()` de esta forma:

```
void setLiked(Place place){
    setState(() {
        place.liked = !place.liked;
        userBloc.likePlace(place, widget.user.uid);
    });
}
```

Lo que estamos haciendo aquí es darle el comportamiento de un toggle o un switch al botón like para que se inserte el dato correcto dependiendo de si hacemos *like* o *dislike*.

Mira el código del archivo aquí

Mira el código completo en este enlace.

Vamos a la siguiente clase terminar la implementación del comportamiento del botón like y la inserción del usuario que le dio *like* como referencia.