

CatchPad Case Study Documentation

Content:

- 1- Structure
- 2- Plugins
- 3- BLoC/Cubit/Stream
- 4- Utility

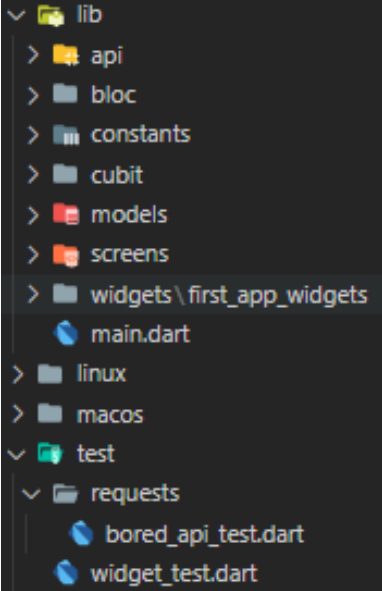
Abstract:

The app is developed with Flutter 3.0.1. Main method runs the MyApp widget which returns a MaterialApp widget. The first MaterialApp widget has 2 MaterialApps in a row as requested.

Second MaterialApp (Second App) is wrapped by a StreamBuilder widget and its controller is passed to the First MaterialApp (First App). So StreamController that passed to the First App can be used in passed widgets and able to add stream data via controller. When stream data is handled in the stream builder its being added to a list of activity then that list of activities is passed as a parameter to the second app. Once second app handles the data its elements is being shown in a list view. First App has a filter icon in the app bar. Filter icon has a provider as its parent for changing the requested url after using filter. When requested url changed it affects ActivityFetchingBloc requesting method url. As result the activities can be filtered and addable to the favorite activity list via streamcontroller.

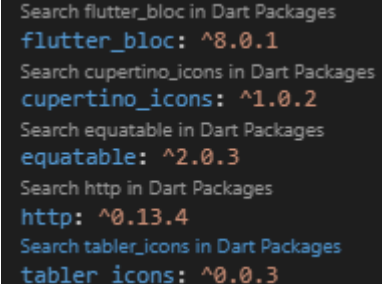
Structure

In lib Folder every type of widget and data is categorized and seperated in folders.

	<p>Api folder has only one dart file that contains needed requests. Since our app has only one endurl with optional parameters, the file has only one method that fetches an activity.</p> <p>Bloc folder has ActivityFetching BLoC files.</p> <p>Cubit folder has only one file thats for emitting the changed url after filtering. Changed url is being sended as a data to the ActivityFetchingBloc and filters the response.</p> <p>Models folder has the ActivityModel file in it.</p> <p>Screens folder has our 2 seperated material app widgets in it. First App files BlocProvider is created and its scaffold widgets body widget is declared in widgets folder.</p> <p>First App has 4 different dart files in widgets folder. AppBar FilterDialog and FirstAppBody is completely seperated from Second app. The activity list file is the only one shared in widgets folder.</p> <p>As last our main.dart file has a streambuilder that wraps second app and its streamcontroller is passed to the first app in it.</p>
---	--

In test folder a unit test is made for fetching activity data correctly.

Plugins

	<p>Plugin usage is kinda minimized while developing. Since it has only 2 material apps and all the logic for it is sharing and fetching data. The app doesn't contain any utility plugin such as AutoRoute, SharedPref or Hive etc.</p> <p>Bloc and Equatable is used for BLoC state management.</p> <p>CupertinoIcons and TablerIcons for UI development and at last http package for data fetching.</p>
---	---

BLoC/Cubit/Stream

ActivityFetchBloc:

```
class ActivityFetchBloc extends Bloc<ActivityFetchEvent, ActivityFetchState> {
  ActivityFetchBloc() : super(ActivityFetchInitial()) {
    on<FetchActivity>((event, emit) async {
      emit(ActivityFetchLoading());
      try {
        ActivityModel newActivity =
          await Requests.fetchActivity(event.currentUrl);
        emit(ActivityFetchLoaded(newActivity));
      } catch (e) {
        emit(ActivityFetchError(Strings.noNewActivity));
      }
    });
  }
}
```

Bloc has 4 states:

First State is Initial state. Initial state informs user that there is no current activity to show.

After adding FetchActivity to Bloc with BlocProvider.of... via add button Fetching starts.

Firstly event emits Loading state. Loading state is being built in BlocProvider as a column with 2 widgets. Columns first widget is a list view that lists activities that already added and as second it shows a loading indicator while fetching. Event has a String variable in it for fetching request method.

Fetching is surrounded with try catch. If any error occurs bloc emits error state and it tells user that there is no new activity. If nothing goes wrong and activity is loaded, bloc emits the new activity to the builder and its being added to the activity list.

UrlCubit:

```
class UrlCubit extends Cubit<String> {
  UrlCubit() : super(URL.activity);

  emitNewUrl(String newUrl) {
    emit(newUrl);
  }
}
```

UrlCubit has a provider as parent of the ActivityFetchBloc that provides the requested url to its event. After filtering the url is being send to the Cubit and getting emitted in it. After emitting it provides the new url for our BLoC.

```
BlocBuilder<UrlCubit, String> (
  builder: (context, state) => IconButton(
    onPressed: () {
      BlocProvider.of<ActivityFetchBloc>(context)
        .add(FetchActivity(state));
    },
    icon: const Icon(Icons.add), // IconButton
  ), // BlocBuilder
```

As it can be seen, the state from UrlCubit is being added to the FetchActivity event. Every time when CubitUrl changes the add button is being rebuilt and being provided with the new url.

Main File:

```
final StreamController<ActivityModel> streamController =  
    StreamController.broadcast();  
List<ActivityModel> favoriteActivities = [];  
@override  
Widget build(BuildContext context) {  
    return MaterialApp(  
        title: Strings.appTitle,  
        debugShowCheckedModeBanner: false,  
        theme: ThemeData.dark(),  
        home: Row( // 2 Seperated Material Apps is called in one  
            children: [  
                Expanded(  
                    child: FirstApp(  
                        streamController: streamController,  
                    )), // FirstApp // Expanded  
                StreamBuilder<ActivityModel>(  
                    stream: streamController.stream,  
                    builder: (context, snapshot) {  
                        if (snapshot.hasData && !favoriteActivities.contains(snapshot.data)) {  
                            favoriteActivities.add(snapshot.data!);  
                        }  
                        return Expanded(  
                            child: SecondApp(  
                                favoriteActivities: favoriteActivities,  
                            )); // SecondApp // Expanded  
                    }  
                ), // StreamBuilder  
            ],  
        ), // Row  
    ); // MaterialApp
```

In MyApp widget firstly we created a StreamController as broadcast.

MaterialApp has a Row widget for home parameter. Row widget has 2 widgets in it. The first one is calling the FirstApp material app widget and the stream controller is being sended as a parameter to it. The second widget is a streambuilder that returns second app everytime. Once a stream data being added from first app via streamcontroller. StreamBuilder responds it with some conditions and receives the favorite activity then adds it to the favoriteActivities list. Favorite activities list is being sended as a parameter to the second app and being listed.

Utility

```
class Strings {
  static String appTitle = "CatchPad";
  static String minPrice = "Min Price";
  static String maxPrice = "Max Price";
  static String incorrectPriceRange =
    "Incorrect price range Min=1 Max=200\$ or 4000\$";
  static String activityError =
    "An error occurred while trying to fetch activity data";
  static String noNewActivity = "Couldn't Fetch new Activity";
  static String noActivity = "There is no Activity";
  static String noFilter = "No Filter";
  static String type = "Type";
  static String initialType = "education";
  static String participants = "Participants";
  static String price = "Price";
  static String noConnection = "There is no connection";
  static List<String> typeList = [
    "education",
    "recreational",
    "social",
    "diy",
    "charity",
    "cooking",
    "relaxation",
    "music",
    "busywork"
  ];
}
```

```
class URL {
  static String baseUrl = "http://www.boredapi.com/";
  static String api = "api/";
  static String activity = "$baseUrl${api}activity/";
  static String priceRange(num min, num max) =>
    "$activity?minprice=$min&maxprice=$max";
  static String participantFilter(int amount) =>
    "$activity?participants=$amount";
  static String typeFilter(String type) =>
    "$activity?type=$type";
}
```

Strings in app is being stored as static variables in a class for easy localization. URL/End Point management is stored in a different class.

```
return Flexible(
  child: ReorderableListView(
    shrinkWrap: true,
    onReorder: (oldIndex, newIndex) {
      setState(() {
        if (newIndex > oldIndex) {
          newIndex -= 1;
        }
        final ActivityModel item = widget.activities.removeAt(oldIndex);
        widget.activities.insert(newIndex, item);
      });
    },
    children: widget.activities.map((activity) {
      return ActivityCard(key: ValueKey(activity),
        activity: activity, streamController: widget.streamController);
    }).toList(),
  ), // ReorderableListView
); // Flexible
```

For viewing activity list ReorderableListView is used. ActivityList view is a shared code between first app and second app.

```
class ActivityModel {
  String? activity;
  String? type;
  int? participants;
  num? price;
  String? link;
  String? key;
  num? accessibility;

  ActivityModel(
    {this.activity,
    this.type,
    this.participants,
    this.price,
    this.link,
    this.key,
    this.accessibility});

  ActivityModel.fromJson(Map<String, dynamic> json) {
    activity = json['activity'];
    type = json['type'];
    participants = json['participants'];
    price = json['price'];
    link = json['link'];
    key = json['key'];
    accessibility = json['accessibility'];
  }
}
```

A model is used for transforming fetched activity json file to an Object. All of its vars is stored in a ActivityModel object. After fetching data and setting the data to model. Our activity object is being emitted via BLoC to our activitylist builder.

```
} else if (state is ActivityFetchLoaded) {
  activities.add(state.activity);
  return ActivityListView(
    activities: activities, streamController: streamController);
}
```

