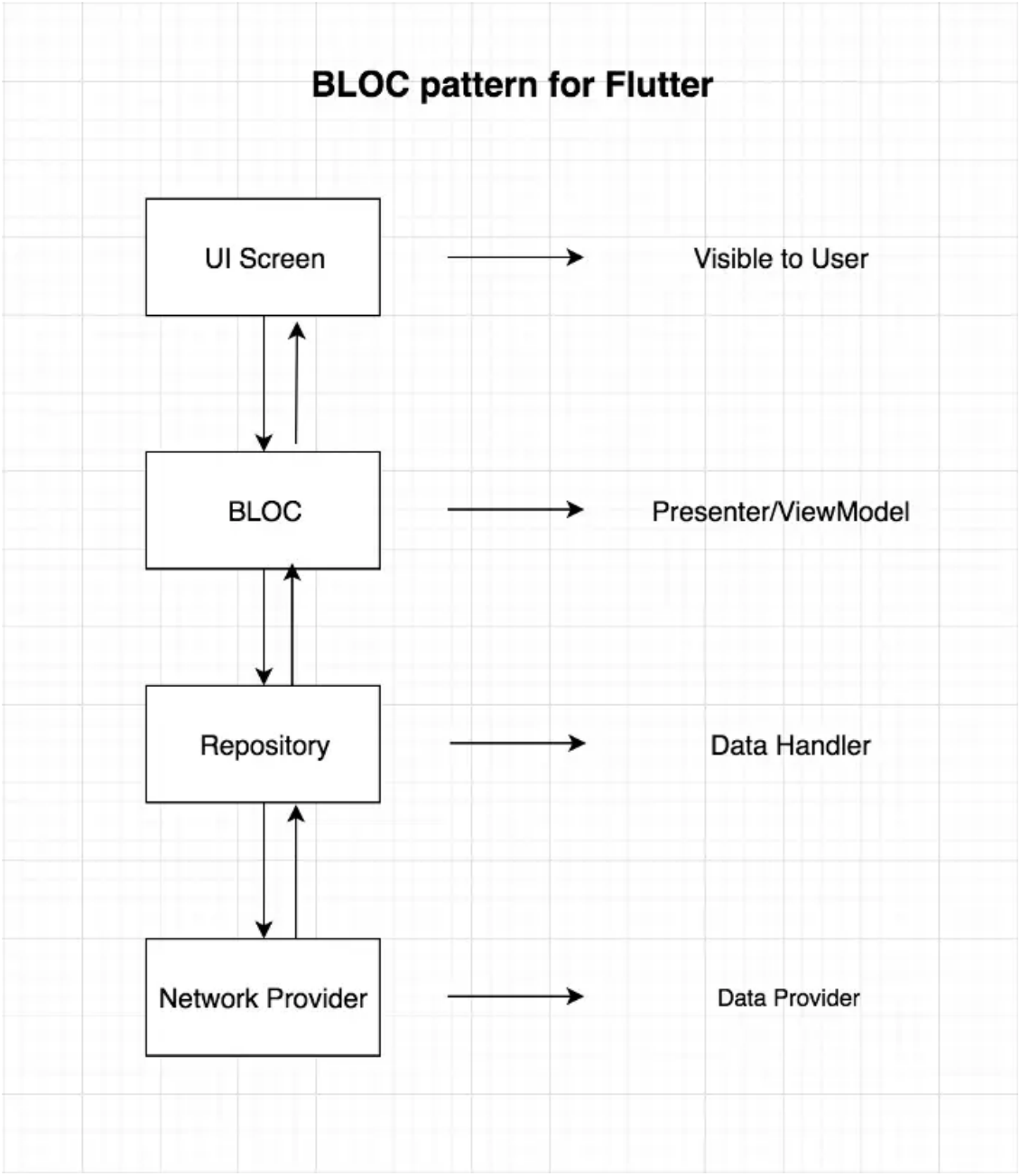
使用 BLOC 模式构建你的 Flutter 项目

Pandorox

在直接进入代码之前，让我给你展示一下 BLOC 架构的视觉体验。我们将遵循这个架构构建 app。



The BLOC pattern

上图展示了数据如何从 UI 流向数据层，反之亦然。BLOC 不会持有 UI 中 Widgets 的引用。UI 仅会监听来自 BLOC class 的变化。让我们做一个小问答来理解这个图：

1. 什么是 BLOC 模式?

它是 Google 开发人员推荐的 Flutter 状态管理系统。它从项目的中心位置访问数据，有助于管理状态。

2. 我可以将此架构与其他任何架构相关联吗?

当然可以。 MVP 和 MVVM 就是一些很好的例子。唯一会改变的是：BLOC 将被 MVVM 中的 ViewModel 所替代。

3. BLOC 的底层是什么？或者在一个地方管理状态的核心是什么?

STREAMS 或 REACTIVE 方式。一般来说，数据将以流的形式从 BLOC 流向 UI 或从 UI 流向 BLOC。如果你从未听说过流，请阅读 Stack Overflow 的回答。

希望这个小问答部分能消除你的疑虑。如果需要进一步澄清或想提出特定问题，可以在下面发表评论或直接在 LinkedIn 上与我联系。

开始使用 BLOC 模式构建项目

1.首先新建一个项目，清除 **main.dart** 文件中的所有代码。

在终端中输入以下命令：

```
flutter create myProjectName
```

2.在 **main.dart** 文件中写下以下代码：

```
import 'package:flutter/material.dart';
import 'src/app.dart'
void main() {
  void main() {
    runApp(App);
  }
}
```

3.在 **lib** 包下创建一个 **src** 包，在 **src** 包中创建一个文件并将其命名为 **app.dart**，将以下代码复制粘贴到 **app.dart** 文件中。

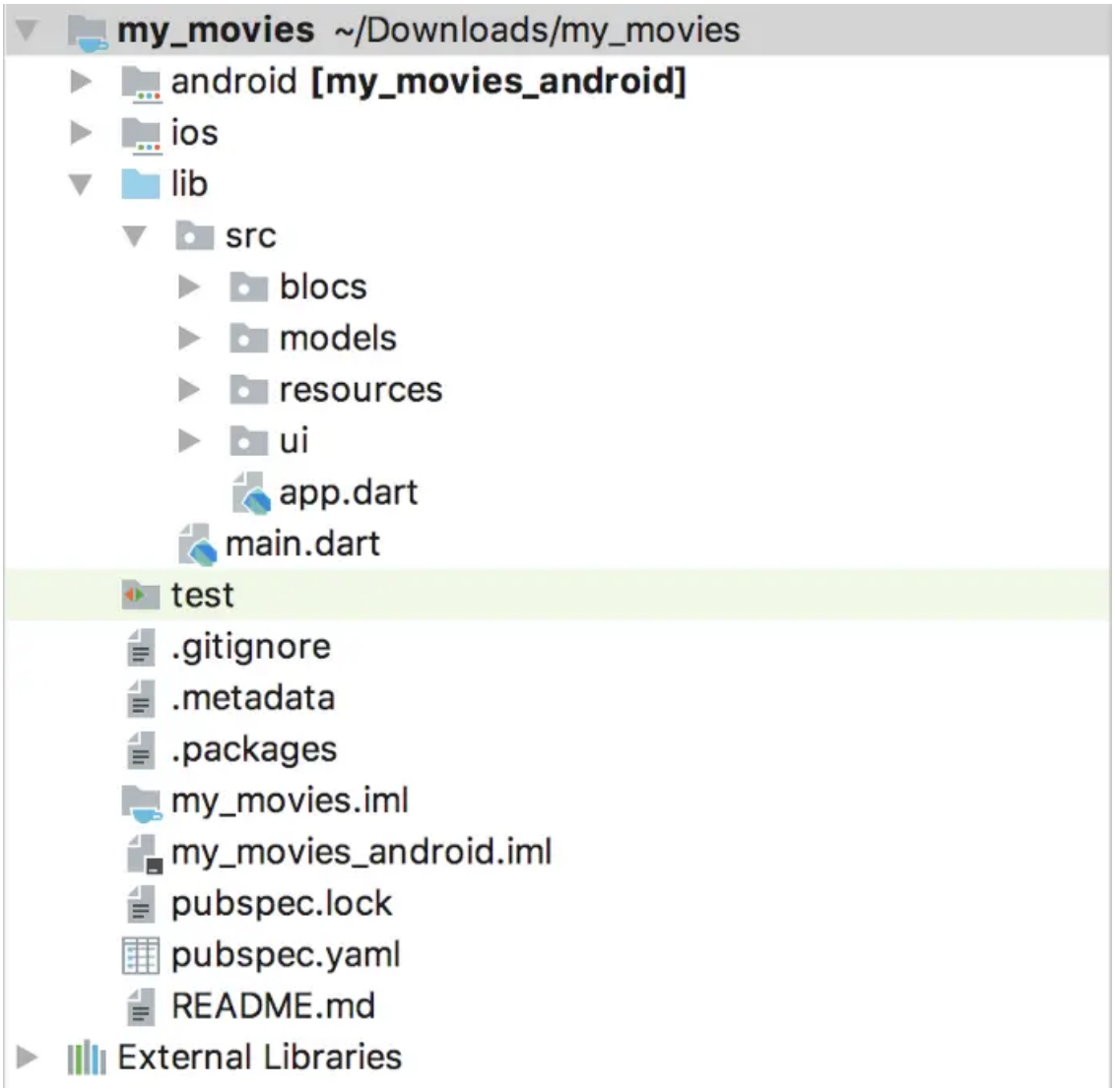
```
import 'package:flutter/material.dart';
import 'ui/movie_list.dart';

class App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return MaterialApp(
        theme: ThemeData.dark(),
        home: Scaffold(
          body: MovieList(),
        ),
      );
  }
```

```
}
```

4.在 **src** 包下创建一个新包，并将其命名为 **resources**。

现在创建几个新包，即 blocs、models、resources 和 ui，如下图所示，然后我们设置项目的骨架：



Project structure

blocs 包将存放 BLOC 实现的相关文件。**models** 包将存放 POJO 类，或从服务器获取的 JSON 的模型类。资源包将包

含存储库类和网络调用实现类。**resources** 包将存放数据存储库类和负责网络调用的实现类。 **ui** 包将存放用户可见的 UI 页面。

5.最后一件事，我们需要添加一个第三方库 RxDart 。打开 **pubspec.yaml**，添加 **rxdart: ^0.18.0**，如下所示：

```
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your
application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.2
  rxdart: ^0.18.0
  http: ^0.12.0+1
```

sync 你的项目，或在终端中键入以下命令。请确保在 project 根目录中执行此命令。

```
flutter packages get
```

6.现在我们已经完成了 project 的骨架搭建，现在开始处理项目的底层逻辑，即**网络层**。我们先了解一下我们即将使用的服务端 API 。点击 **link**，你将被带到电影网站数据库 API 页面。完成注册，并从设置页面获取你的 *API key*。我们将从下面的 url 获取数据：

http://api.themoviedb.org/3/movie/popular?api_key="your_api_key"

将你的 *API key* 放到上面的 url 中并点击这个 url (删除双引号)，你可以看到类似下面的 JSON 返回数据：


```
{
  "page": 1,
  "total_results": 19772,
  "total_pages": 989,
  "results": [
    {
      "vote_count": 6503,
      "id": 299536,
      "video": false,
      "vote_average": 8.3,
      "title": "Avengers: Infinity War",
      "popularity": 350.154,
      "poster_path": "\/7WsyChQLEftFiDOVTGkv3hFpyyt.jpg",
      "original_language": "en",
      "original_title": "Avengers: Infinity War",
      "genre_ids": [
        12,
        878,
        14,
        28
      ],
      "backdrop_path": "\/bOGkgRGdhrBYJSLpXaxhXVstddV.jpg",
      "adult": false,
      "overview": "As the Avengers and their allies have continued to protect the world from threats too large for any one hero to handle, a new danger has emerged from the
```

```
cosmic shadows: Thanos. A despot of intergalactic infamy,
his goal is to collect all six Infinity Stones, artifacts
of unimaginable power, and use them to inflict his twisted
will on all of reality. Everything the Avengers have fought
for has led up to this moment - the fate of Earth and
existence itself has never been more uncertain.",
      "release_date": "2018-04-25"
    },
```

7.为网络返回的这种数据类型创建数据模型或 POJO 类。在 **models** 包下创建一个新的文件，命名为 **item_model.dart**，并复制下面的代码到文件中：

```
class ItemModel {
  int _page;
  int _total_results;
  int _total_pages;
  List<_Result> _results = [];

  ItemModel.fromJson(Map<String, dynamic> parsedJson) {
    print(parsedJson['results'].length);
    _page = parsedJson['page'];
    _total_results = parsedJson['total_results'];
    _total_pages = parsedJson['total_pages'];
    List<_Result> temp = [];
    for (int i = 0; i < parsedJson['results'].length; i++)
{
      _Result result = _Result(parsedJson['results'][i]);
      temp.add(result);
    }
    _results = temp;
  }

  List<_Result> get results => _results;
```

```
  int get total_pages => _total_pages;

  int get total_results => _total_results;

  int get page => _page;
}

class _Result {
  int _vote_count;
  int _id;
  bool _video;
  var _vote_average;
  String _title;
  double _popularity;
  String _poster_path;
  String _original_language;
  String _original_title;
  List<int> _genre_ids = [];
  String _backdrop_path;
  bool _adult;
  String _overview;
  String _release_date;

  _Result(result) {
    _vote_count = result['vote_count'];
    _id = result['id'];
    _video = result['video'];
    _vote_average = result['vote_average'];
    _title = result['title'];
    _popularity = result['popularity'];
    _poster_path = result['poster_path'];
    _original_language = result['original_language'];
    _original_title = result['original_title'];
```

```
    for (int i = 0; i < result['genre_ids'].length; i++) {
      _genre_ids.add(result['genre_ids'][i]);
    }
    _backdrop_path = result['backdrop_path'];
    _adult = result['adult'];
    _overview = result['overview'];
    _release_date = result['release_date'];
  }

  String get release_date => _release_date;

  String get overview => _overview;

  bool get adult => _adult;

  String get backdrop_path => _backdrop_path;

  List<int> get genre_ids => _genre_ids;

  String get original_title => _original_title;

  String get original_language => _original_language;

  String get poster_path => _poster_path;

  double get popularity => _popularity;

  String get title => _title;

  double get vote_average => _vote_average;

  bool get video => _video;

  int get id => _id;
```

```
  int get vote_count => _vote_count;
}
```

我希望你可以将此文件和服务端返回的 JSON 进行影射。如果不是这样，你需要知道的是我们最关心的是 `Results` 类中的 **poster_path** ，我们将在我们的主页面中现实所有热门电影的海报（posters）。`fromJson()` 方法是用来获取解码后的 JSON ，并将数据放到正确的变量中。

8.现在处理网络请求。在 **resources** 包下新建一个文件，命名为 **movie_api_provider.dart** ，复制下面的代码到文件中，稍后我会进行解释：

```
import 'dart:async';
import 'package:http/http.dart' show Client;
import 'dart:convert';
import '../models/item_model.dart';

class MovieApiProvider {
  Client client = Client();
  final _apiKey = 'your_api_key';

  Future<ItemModel> fetchMovieList() async {
    print("entered");
    final response = await client
        .get("http://api.themoviedb.org/3/movie/popular?
api_key=$_apiKey");
    print(response.body.toString());
    if (response.statusCode == 200) {
      // If the call to the server was successful, parse
```

```
the JSON
      return
ItemModel.fromJson(json.decode(response.body));
    } else {
      // If that call was not successful, throw an error.
      throw Exception('Failed to load post');
    }
  }
}
```

Note：请将 moive_api_provider.dart 文件中的 `_apiKey` 的值替换为你的 *API key*，否则将不能请求到数据。

`fetchMovieList()` 方法用来向服务端 API 发起网络请求。一旦请求完成，如果网络请求成功，它将返回一个 Feature **ItemModel** 对象；否则，它将抛出一个异常。

9.下面我们将在 **resource** 包下创建一个新的文件，命名为 **repository.dart**。复制下面的代码到文件中：

```
import 'dart:async';
import 'movie_api_provider.dart';
import '../models/item_model.dart';

class Repository {
  final moviesApiProvider = MovieApiProvider();

  Future<ItemModel> fetchAllMovies() =>
moviesApiProvider.fetchMovieList();
}
```

文件中导入了 **movie_api_provider.dart**，并调用了

`fetchMovieList()` 方法。**Repository** 类是数据流向 **BLOC** 的中心点。

10.下面的部分稍微有点复杂，实现 bloc 逻辑。在 **blocs** 包下新建一个文件，命名为 **movies_bloc.dart** 。复制下面的代码到文件中，后面我会详细解释代码：

```
import '../resources/repository.dart';
import 'package:rxdart/rxdart.dart';
import '../models/item_model.dart';

class MoviesBloc {
  final _repository = Repository();
  final _moviesFetcher = PublishSubject<ItemModel>();

  Observable<ItemModel> get allMovies =>
_moviesFetcher.stream;

  fetchAllMovies() async {
    ItemModel itemModel = await
_repository.fetchAllMovies();
    _moviesFetcher.sink.add(itemModel);
  }

  dispose() {
    _moviesFetcher.close();
  }
}

final bloc = MoviesBloc();
```

导入 RxDart package `import 'package:rxdart/rxdart.dart';`,

这最终会将 RxDart 相关的所有方法和类导入到这个文件中。在 MoviesBloc 类中创建一个 `Repository` 对象，用来访问 `fetchAllMovies()` 方法。创建一个 PublishSubject 对象，它的职责是：以流的形式将添加到其中的 ItemModel 对象（从服务端获取的数据模型类）传递给 UI。为了将 ItemModel 对象作为流传递，需要创建另一个方法 `allMovies()`，返回类型是 Observable （如果你不了解 Observables，请观看此视频）。文件的最后一样创建了一个 bloc 对象，这样方便 UI 以单例的方式访问 MoviesBloc 类。

如果你不知道什么事响应式编程，请看这个简单的说明。简单的说，只要从服务端有新的数据返回，我们就必须更新 UI。为了简化这个更新任务，我们让 UI 保持监听来自 MoviesBloc 类的任何数据变化，并相应的更新所展示的内容。这种对数据的监听，可以通过使用 **RxDart** 完成。

11.这是最后部分了，在 UI 包下创建一个文件，命名为 **movie_list.dart** 。复制下面的代码到文件中：

```
import 'package:flutter/material.dart';
import '../models/item_model.dart';
import '../blocs/movies_bloc.dart';

class MovieList extends StatelessWidget {
```

```
  @override
  Widget build(BuildContext context) {
    bloc.fetchAllMovies();
    return Scaffold(
      appBar: AppBar(
        title: Text('Popular Movies'),
      ),
      body: StreamBuilder(
        stream: bloc.allMovies,
        builder: (context, AsyncSnapshot<ItemModel>
snapshot) {
          if (snapshot.hasData) {
            return buildList(snapshot);
          } else if (snapshot.hasError) {
            return Text(snapshot.error.toString());
          }
          return Center(child:
CircularProgressIndicator());
        },
      ),
    );
  }

  Widget buildList(AsyncSnapshot<ItemModel> snapshot) {
    return GridView.builder(
        itemCount: snapshot.data.results.length,
        gridDelegate:
            new
SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount:
2),
        itemBuilder: (BuildContext context, int index) {
          return Image.network(
            'https://image.tmdb.org/t/p/w185${snapshot.data
                .results[index].poster_path}',
```

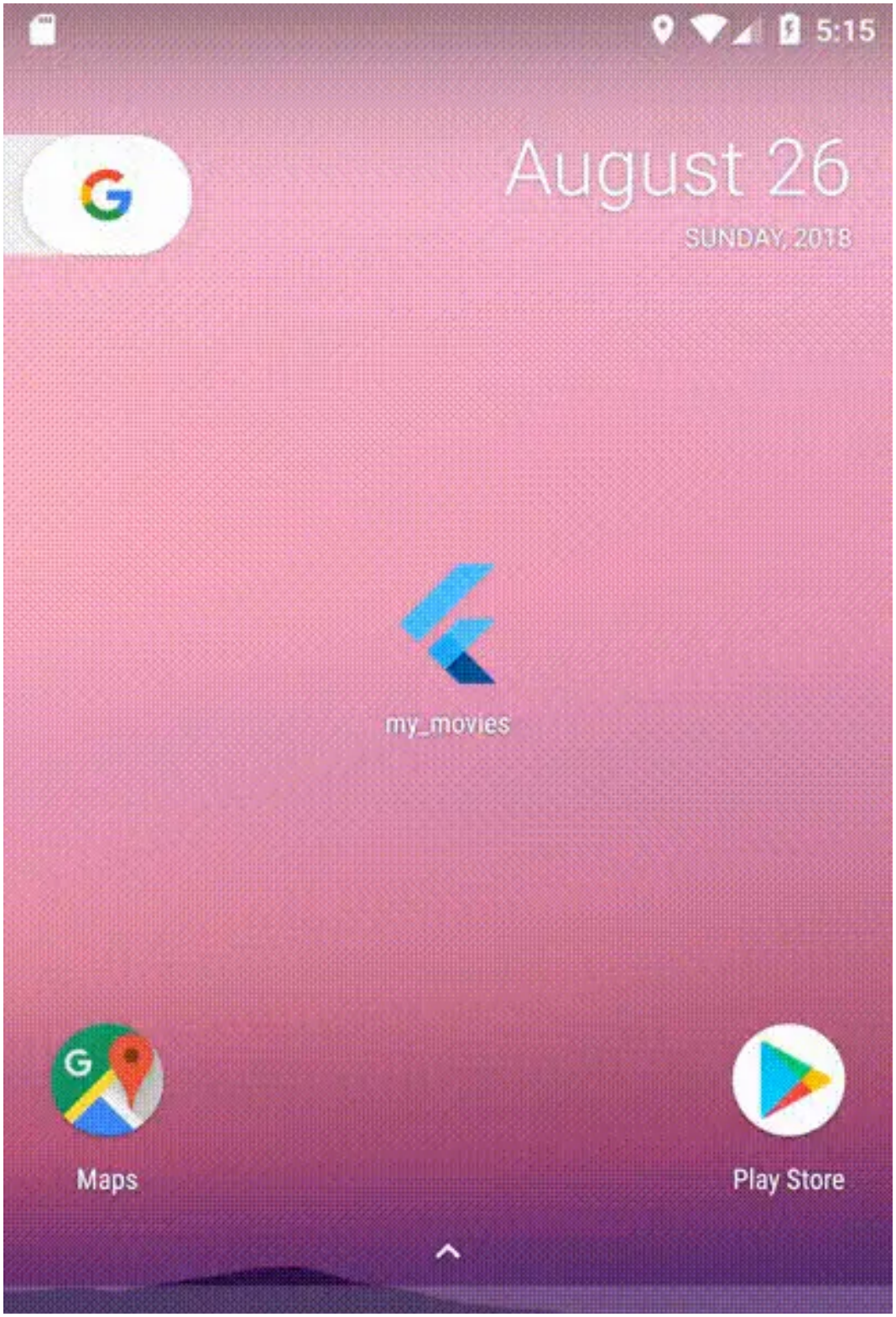
```
            fit: BoxFit.cover,
          );
        });
  }
}
```

这个类最有意思的地方是，我没有使用 **StatefulWidget**，而是使用了一个 **StreamBuilder** ，它可以像 **StatefulWidget** 一样实现更新 UI。

这里需要指出的一点是，我在 build 方法中进行了网络请求调用。这是不应该的，因为 `build(context)` 方法会被调用多次。但由于文章变得越来越长，也越来越复杂，为了保持简单，这里仍然在 `build(context)` 方法中调用网络请求。后续我会更新这篇文章，以一种更好的方式进行网络调用。

正如我所说的，MoviesBloc 类将新数据作为流传递。为了处理流，有一个很好的内置类，即 StreamBuilder，它将监听传入的流并相应地更新 UI。StreamBuilder 需要一个 `stream` 参数，这里我们传递 MovieBloc 的 `allMovies()` 方法，因为 `allMovies()` 返回一个流。当有数据流过来，StreamBuilder 将使用最新的数据重新渲染 widget ，这些数据中将包含 ItemModel 对象。你可以使用任何的 Widget 展示数据对象中的任何数据（这是你的额创造力就展现出来了）。我使用一个 GridView 来显示 ItemModel 对象结果列表中的所有海

报。这是最终产品的输出：



Small demo. The video was not capturing the complete frames I guess 😐

到了文章的末尾，伙计们，你们能坚持到最后真是太好了，希望你们喜欢这篇文章。如果你有任何疑问或问题，请通过 LinkedIn 或 Twitter 联系我。请欣赏这篇文章，不要吝惜你的掌声和评论。

如果你需要完整的源码，请访问这个工程项目的 githut repository