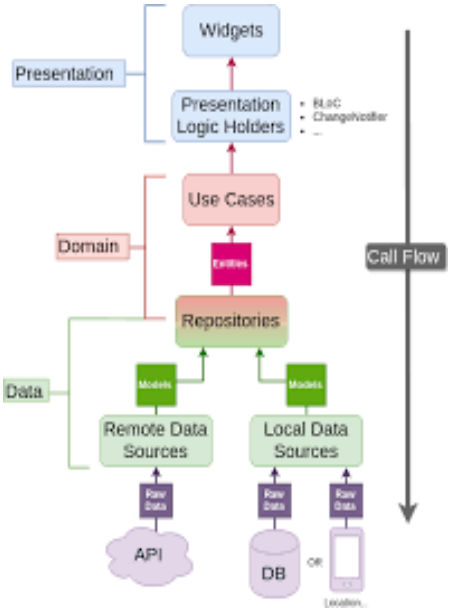
Clean Architecture

الهدف منها :

- كتابه كود مرتب ومفهوم بالإضافه إلي سهوله ايجاد الخطأ وكذالك معالجته بحيث يسهل التعديل عليه بجانب كونه راح يكون سهل اذا في شخص تاني جه شاف الكود ودا بيرجع لتنظيم الكود والملفات والناتج عن تنفيذ ال Clean Architecture .
- اللي خلي هذه البنيه Standard in Software هو اونكل بوب او Robert Martin وهو نفسه اللي عمل ال Solid Principles وال Clean Code .



- لاحظ هنا العمليه بتنفذ من فوق لتحت بدايه من Presentation مرورا ب Domain لحتي نصل لل Data ومن ثم نعود بالداتا.

خطوات تنفيذه :

- ف البدايه هنقسم التطبيق ل Features علي سبيل المثال تطبيق Social ف ممكن يكون في كذا خاصيه مثلا Auth , Posts وغيرها ف هي بترجع لتطبيق المشروع اللي شغال عليه ف الفائده هنا ان لو في خطأ مثلا ف تسجيل الدخول هتكون عارف مكان الخطأ واللي تبع feature اللي اسمها Auth ف هنا يعتبر كل حاجه معزوله عن التانيه.

- كل feature هتحتوي علي 3 layers واللي هم Presentation , Domain , Data .
- ف البدايه هنتكلم عن Presentation Layer وده اللي بيكون فيها UI تبع التطبيق بجانب الكود تبع state management ف بيكون فيها 3 folders واللي هم screens وده هحط UI تبع الاسكرينات التابعه للخاصيه ده فيها ، Widgets وده مثلا لو عندي ويدجت هستعمله ف اكتر م اسكرينه داخل الخاصيه ف هحطها بالفولدر ده ، Controller وده بقي اللي هيحتوي علي الكود تبع state management وليكن مثلا ال Cubit ف هيكون فيه ملف خاص بالكيوبيت واخر ب states .
- أما بالنسبه لل Domain layer ف هي الوسيطه بين ال Data , Presentation وبتحمل 3 folders واللي هم UseCases , Entities , Contract Repositories هنتكلم عن كل واحده بالتفصيل.

- **: UseCases**

  ده هحط بداخلها فايل خاص بكل داله موجوده داخل ال Feature اللي انا فيها ع سبيل المثال لو بنتكلم عن البوستات ف هعمل مثلا فايل خاص ب createPost واخر updatePost واخر deletePost واخر getAllPosts وهتوضح اكتر مع الكود.

- **: Entities**

  هي حاله مصغره من ال Model ازاي يعني هيكون بداخله فقط تعريف المتغيرات اللي بداخل الكلاس عشان انا اللي هتعامل معاه ف UI هم فقط المتغيرات اللي بداخل الكلاس مش مثلا الداله تبع fromJson or toJson طب دول هيكونوا فين ؟ هيتحطوا في Models اللي هتكون موجوده ف Data layer لان هناك هيكون مسؤله عن عمل refactor للداتا اللي جايه م API .

- **: Contract Repository**

  ده هيكون فيها abstract class هحط فيها كل الدوال اللي راح تكون بال feature اللي شغال عليها وليكن ال posts بس طبعا هتكون مجرده بمعني هكتب فقط ال prototype تبع الداله فقط بدون ال body لان الخطوه ده هتتم في Data layer اما اعمل كلاس ي implement للكلاس اللي ف contract repo .

- بالنسبه لل Data layer ف ده بيكون فيها التعامل مع الداتا بيز سواء ان كانت local or remote ، بتحتوي علي 3 folders واللي هم Imply Repositories , Models , data_source هنتكلم ع كل واحد بالتفصيل.

- **: Models**

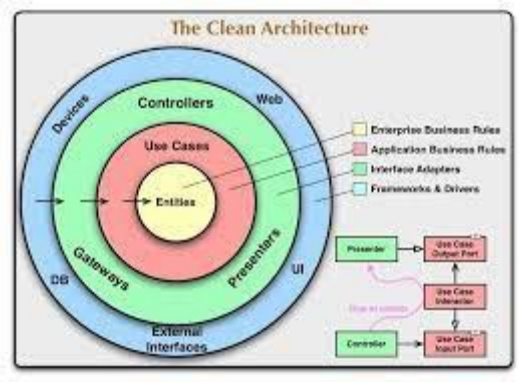
ده بعمل فيها كلاس بيعمل extend من ال entity المقابل له بجانب انه هيتوي بقي toJson , fromJson واللي خاصين بالتعامل مع ال API بمعني كنت عامل class PostEntity هاجي داخل Models هعمل class PostModel extends PostEntity ف ده كده ب اختصار الفكره.

- **: DataSource**

ده بقي بيكون فيها 2 files واحد خاص ب local , remote network عشان مثلا لو شغال علي تطبيق فيه API وعاوز تعمل Caching للداتا وممكن نضيف فايل تالت ف الحاله ده يكون عباره ع abstract class واحط فيها كل الدوال اللي هنفذ فيها في local , remote وكل اللي عليا داخل كل فايل هبقي اعمل implement لل abstract class ، بس طبعا ف بعض الحالات ممكن اعمل فايل واحد خاص ب local لو شغال مثلا ع حاجه زي ال Notes App بس الافضل ان اعمل abstract class عشان مثلا لو تطبيق زي ده حاولت استعمل فيه الفايربيز مثلا مكان ال local storage تبقي الدنيا سهله لان كل اللي هعمله هعمل implement for abstract class وهعمل override for all methods .

- **: Imply Repositories**

ده بعمل فيه implement لل contract repo اللي ب Domain layer طب ال body تبع الدوال هجيبه منين راح يكون من Data Source بس طبعا لو عامل Cache للداتا هبقي اشيك لو ف نت مثلا يجيب الداتا اللي ب LocalDataSource ولو ف نت هتكون م RemoteDataSource .



الصوره ده بتوضح اي طبقه المفروض ابلش فيها عند كتابه الكود :

- Entities
- Contract Repo
- Use Cases
- Models
- Data Source
- Imply Repo
- من ثم بقي Presentation layer

ده packages اللي هنستعملها ف التطبيق تبع البوستات

- ***dartz :***

بتوفر امكانيه ان الداله ترجع واحده من القيمتين بمعني ممكن يحصل Failure او حاجه اثناء تنفيذ الداله او ان الداتا تجي بشكل صحيح ف لو كده هخليها ممكن ترجع instance from Failure او انها ترجع مثلا response اللي جاي م API واللي بيكون ع هيئه json or map .

مثال =>

```
Future<Either<Failure,List<ClubEntity>>> getClubs();
```

ف انا هكون عارف لو اللي راجع ال left هيكون instance from Failure ولو اليمين يبقي List<ClubEntity>.

- Equatable :

بتساعدني في عمليه المقارنه بين ال objects وبعض عشان اعرض اذا كانوا بيحملوا نفس الداتا ولا اي

```
// PostModel

// PostModel post1 = PostModel(title:"some title", body : "some text")
// PostModel post2 = PostModel(title:"some title", body : "some text")
// print( post1 == post2 )
```

hashCode function will be like this :

```
@override
bool operator ==(Object other) {
  if (identical(this, other)) return true;
  return other is Post &&
    other.title == title &&
    other.body == body;

@override
int get hashCode => title.hashCode ^ body.hashCode;
```

لان هنا ف الطبيعي ان post1 = post2 بس لاء هنا هيكون الناتج false طب عشان اقدر اطلع ناتج المقارنه صح لازم اعمل override for hashCode Function مثل ما موضح ع اليمين ف الفكره بقي ان ال Equatable بتعمل override لها بشكل تلقائي مثل الكود التالي :

```
import 'package:equatable/equatable.dart';

class MemberEntity extends Equatable {
  final String memberID;
  final String memberName;
  final String membershipDate;

  const MemberEntity({required this.memberName,required this.memberID, required this.membershipDate});

  @override
  // TODO: implement props
  List<Object?> get props => [memberName,membershipDate,memberID];
}
```

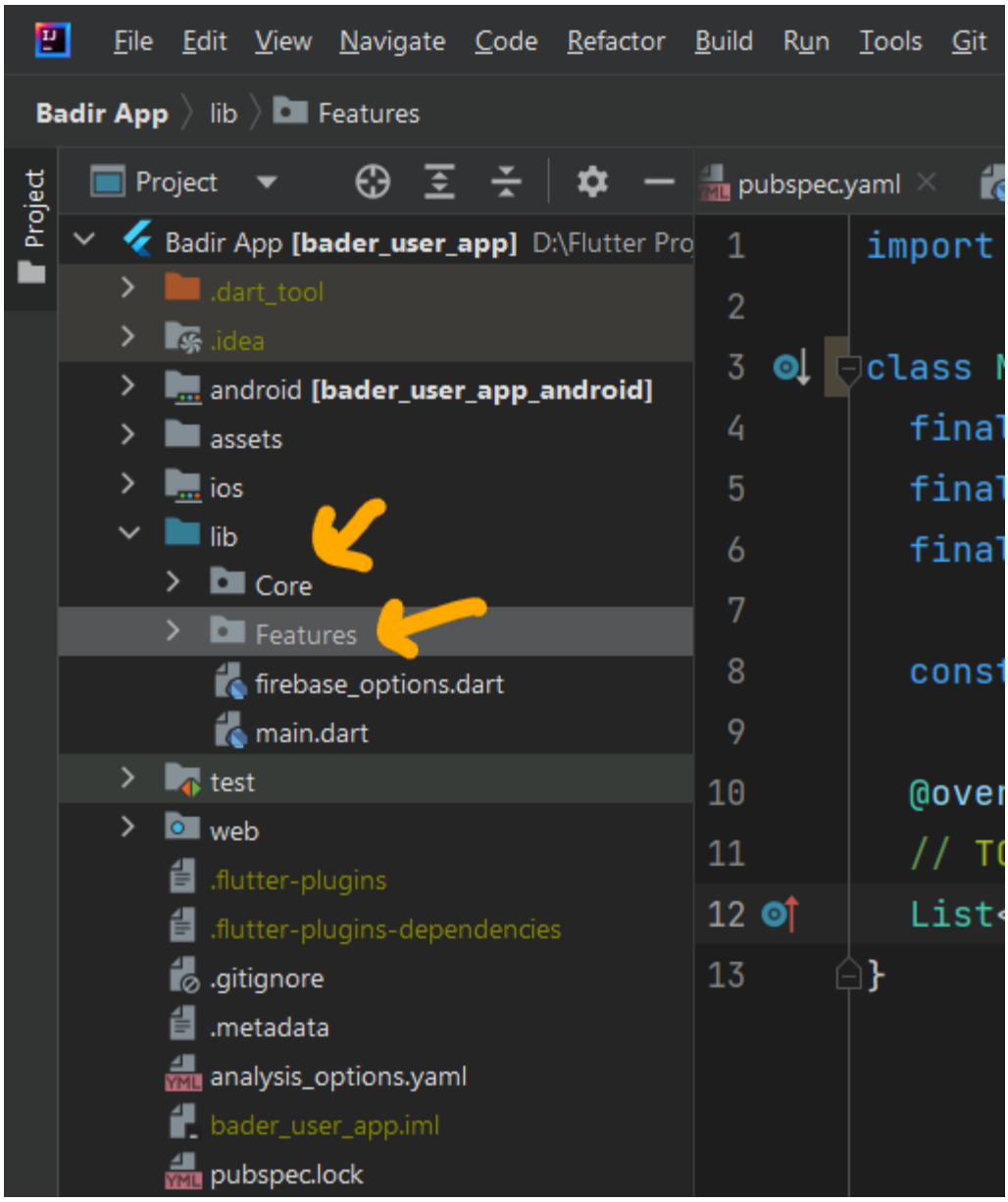
بس خلي بالك لازم اعمل override لل props method وداخل List اللي هترجع همرر فيها attributes or variables on this class وال Equatable هيعمل هو override لل hashCode Function بشكل تلقائي.

الفائده من استعمال Equatable :

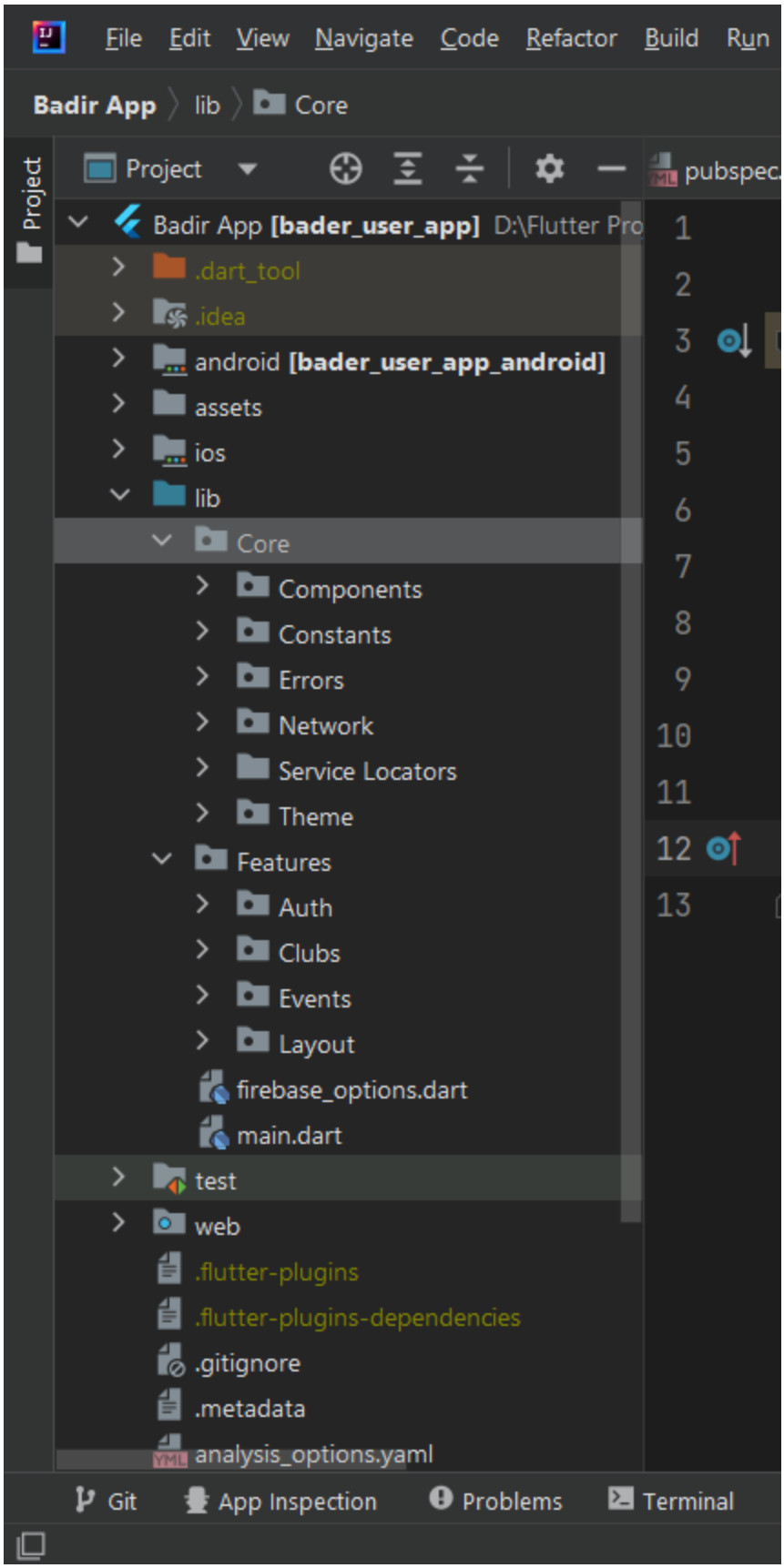
- هي ان اقارن الداتا اللي معايا مثلا وليكن List<ClubEntity> باللي جاي من ال API واشوفها لو هي نفسها مش هعمل build لل UI م خلال buildWhen اللي بتكون خاصه ب BlocBuilder عشان اقلل عمليه build UI بالتالي مش هيحصل build لو الداتا جديده بالفعل.

- get_it :

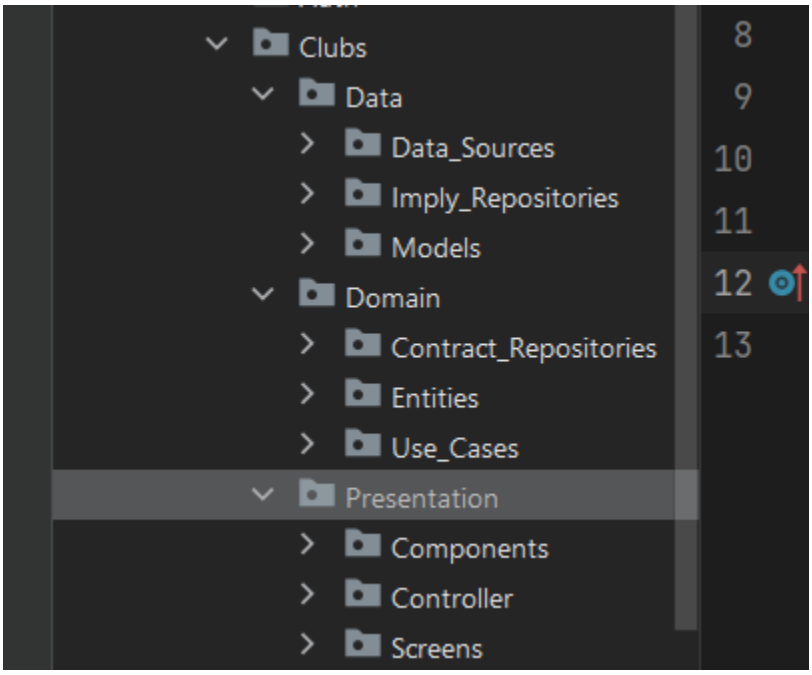
هستعملها ف عمليه Dependency Injection هنشرحها بالتفصيل ف فيديو جاي.



- هنا هنقسم المشروع كالتالي فولدر اسمه Core هيكون فيه الحاجات العامه بالتطبيق زي constants , strings , theme , failure , components والتاني Features بيكون مثل هيك :


File
Edit
View
Navigate
Code
Refactor
Build
Run
Badir App
lib
Core
Project
pubspec.
Badir App [bader_user_app] D:\Flutter Pro
.dart_tool
.idea
android [bader_user_app_android]
assets
ios
lib
Core
Components
Constants
Errors
Network
Service Locators
Theme
Features
Auth
Clubs
Events
Layout
firebase_options.dart
main.dart
test
web
.flutter-plugins
.flutter-plugins-dependencies
.gitignore
.metadata
analysis_options.yaml
Git
App Inspection
Problems
Terminal

وداخل كل Feature هيكون فيه ال 3 layers مثل هيك :



Coding

اولا كيفيه كتابه ال Entity :

```dart
import 'package:equatable/equatable.dart';

class MemberEntity extends Equatable {
  final String memberID;
  final String memberName;
  final String membershipDate;

  const MemberEntity({required this.memberName,required this.memberID, required this.membershipDate});

  @override
  // TODO: implement props
  List<Object?> get props => [memberName,membershipDate,memberID];
}
```

- بعمل كلاس واخليه يرث من Equatable عشان المقارنه بين objects بعدين بس عشان تنفيذ ال hashCode راح اعمل implement لل props method واللي بترجع list م نوع اوبجيكت واللي هنا هرجع attributes داخل ال List مثل ما موضح بالأعلي.
- بعرف ال attributes تبعي وبعدين بعمل Constructor عشان اسند لهم قيم.
- لاحظ هنا انا مش بتعامل مع json او اي حاجه بتخص ال API لان Entity هي اللي بتتعامل معاها Presentation.
- ف الميزه نا ان بعزل كل حاجه عن بعض.

ثانيا كتابه ال Contract Repo :

- ف البدايه كنا قلنا لو عاوز الداله ترجع قيمتين مختلفتين هستعمل dartz م خلال Either<leftValue,rightValue> بالنسبه للي ع اليسار هتنفذ ف حاله الخطأ واليمين اذا كان صح .

```dart
import ...

abstract class ClubsContractRepository{

  Future<Either<Failure,List<ClubEntity>>> getClubs();

  // TODO: CLUBS -- LEADER ROLE
  Future<Either<Failure,String>> uploadClubImageToStorage({required File imgFile});
  Future<Either<Failure,Unit>> removeMemberFromClubILead({required String memberID,required String clubID});

  // Unit replace void role
  Future<Either<Failure,Unit>> updateClubData({required String clubID,required String image,required String name,required int member
  Future<bool> deleteClub({required String clubID});
  Future<Either<Failure,Unit>> updateClubAvailability({required String clubID,required bool isAvailable,required List availableOnlyF

    // TODO: ACCEPT OR REFUSE MEMBERSHIP REQUEST -- LEADER ROLE
  Future<Either<Failure,Unit>> acceptOrRejectMembershipRequest({required String committeeNameForRequestSender,required String request
  Future<bool> deleteMemberFromClub({required String memberID});
  Future<Either<Failure,Set<String>>> getMembersOnMyClub({required String idForClubILead});

  // TODO: Invitation to be a member on Specific Club ( Public Member, Leader on this club will be accept or refuse )
  Future<bool> requestAMembershipOnSpecificClub({String? senderFirebaseFCMToken,required String clubID,required String requestUserNa
  Future<Either<Failure,List<RequestMembershipEntity>>> getMembershipRequests({required String clubID});
```

- بعمل abstract class وحط فيه الدوال كلها تبع ال feature اللي انا فيها بس prototype فقط لان قلنا هعمل implementation ف imply repo اللي ب Data source .
- لاحظ هنا مستعمل Either ف اغلب الدوال عشان اعرف اذا كانت نفذت الداله بشكل صحيح و لاء.
- بالنسبه لل Data type اللي راجعه ف بعض الدوال واللي اسمها Unit ده حاجه بديله كده لل void معناها ان مش هيرجع حاجه لان زي ما احنا عارفين لو عملت void مكان Unit ازاي هقدر ارجع void ف الحل هنا Unit بتقوم بنفس الشئ بس بقدر ارجعها عادي طب ازاي هكتب كده return unit; ف unit ده زي instance from Unit كده ف هنا اكني برجع void اللي هو ولا شئ .
- بالنسبه لل Failure واللي بيرجع ف اغلب الدوال برده هنشوف ازاي بنكتبه :

```dart
import 'package:equatable/equatable.dart';

abstract class Failure extends Equatable {}

class OfflineFailure extends Failure {
  @override
  List<Object?> get props => [];
}

class ServerFailure extends Failure {
  @override
  List<Object?> get props => [];
}

class EmptyCacheFailure extends Failure {
  @override
  List<Object?> get props => [];
}
```

- هعمل abstract class Failure بعدين هعمل بقي كلاس لكل خطأ ع سبيل المثال واحد ف حاله ف خطأ ف server وانا بجيب حاجه منه واخر بيعبر ان مفيش نت واخر بيعبر ان الكاش فاضيه دا ف حاله لو بعمل caching for data وطبعا بالنسبه لل abstract كلاس خلته يرث من Equatable .
- خلي بالك بقي ازاي ال Failure ده هيحصل المفروض عن طريق ال Exception (اما اجي استقبل ال Exception عشان اقدر return ال Failure الصح) عشان كده هعمل Exception مقابل لكل Failure كالتالي :

```dart
class ServerException implements Exception {}

class EmptyCacheException implements Exception {}

class OfflineException implements Exception {}
```

- عملت فايل اسمه exceptions زي ما في failure وحطت فيه 3 exceptions واللي هم مقابلين لل 3 failures .
- طبعا الملفين بداخل فولدر اسمه errors واللي موجود بال Core لان خاص بالتطبيق عامه.

ثالثا كتابه ال Use Cases :

- هتعمل use_case file لكل داله موجوده بال Contract Repo لان هطلبها من ال Contract Repo .

```
main.dart | pubspec.yaml | posts_repository.dart M | get_all_posts.dart U | failures.dart U | exception.dart U
lib > features > posts > domain > repositories > posts_repository.dart > ...
1  import 'package:clean_architecture_posts_app/features/posts/domain/entities/post.dart'
2  import 'package:dartz/dartz.dart';
3
4  import '../../../../core/error/failures.dart';
5
6  abstract class PostsRepository {
7    Future<Either<Failure, List<Post>>> getAllPosts();
8    Future<Either<Failure, Unit>> deletePost(int id);
9    Future<Either<Failure, Unit>> updatePost(Post post);
10   Future<Either<Failure, Unit>> addPost(Post post);
11 }
12
```

```
main.dart | pubspec.yaml | posts_repository.dart M | get_all_posts.dart U | failures.dart U | exception.dart U
lib > features > posts > domain > usecases > get_all_posts.dart > ...
1  import 'package:clean_architecture_posts_app/features/posts/domain/repositories/posts_
2  import 'package:dartz/dartz.dart';
3
4  import '../../../../core/error/failures.dart';
5  import '../entities/post.dart';
6
7  class GetAllPostsUsercase {
8    final PostsRepository repository;
9
10   GetAllPostsUsercase(this.repository);
11
12   Future<Either<Failure, List<Post>>> call() async {
13     return await repository.getAllPosts();
14   }
15 }
16
```

- لاحظ ف الصوره الأولي شكل Contract Repo والتانيه هي Use Case الخاصه ب getAllPosts هي عباره ع كلاس عادي بأخد م خلال constructor ال instance from contract repo عشان م خلاله اقدر اجيب الداله اللي بتجيب البوستات وعامل داله اسمها call بتنادي ع الداله اللي ب contract repo لان presentation layer هتتعامل مع use cases مش ال contract Repo .

- ***اي الفائده م ان خلت الداله اسها call ؟***

- هنوضح حاجه الأول ان اي كلاس بيكون بداخله داله اسمها call بس من غير body والداله ده بتستدعي مع الكلاس اي عند عمل instance م الكلاس بالتالي ممكن نقول ان الكلاس بقي callable .
- بالتالي ف المثال السابق ان لو عملت object من ال GetAllPostsUseCase مباشره هيروح يستدعي call method وبما ان هنا انا عاملها implement ف بالتالي هينفذ ال body تبعها واللي فيه هيروح ينفذ الداله اللي ب contract repo .
- طبعا ممكن اسمي داله ال call ب اي حاجه تانيه وليكن execute بس الفكره كده هضطر انادي عليها عكس call method ، مع العلم ان حتي call method ممكن استدعيها عادي بعد اما اعمل object م الكلاس لانها ف الاخير داله.

```
main.dart   pubspec.yaml   posts_repository.dart M   get_all_posts.dart U   temp.dart U ×   failures.dart U   exception.dar
lib > temp.dart > ...
1  class Hello{
2   String call(String name, int age){
3       return "hello $name, your age $age";
4    }
5
6  }
7
8  Hello hello = Hello();
9  // print(hello.call("Rabee", 23));
10 //callable class
```

- هنا مثال اخر ع callable function ف حاله لو بتحتوي ع parameters هباصيها عادي لل constructor تبع الكلاس مثل ما موضح بالأعلي.
- مثال اخر ع Use Case :

```dart
import 'package:clean_architecture_posts_app/features/posts/domain/repositories/posts_
import 'package:dartz/dartz.dart';

import '../../../../core/error/failures.dart';

class DeletePostUsecase {
  final PostsRepository repository;

  DeletePostUsecase(this.repository);

  Future<Either<Failure, Unit>> call(int postId) async {
    return await repository.deletePost(postId);
  }
}
```

- كنا قلنا ان void مش data type عشان ترجع مثلا من داله انما ال Unit تعتبر data type عشان كده بقدر ارجعها من الداله وقلنا كل اللي هعمله ان اكتب return unit وده معناها ولا شئ.
- بالنسبه لل call method عشان بس الكلاس يصير callable اي عند عمل object منه يستدعي الداله call بشكل تلقائي.

رابعا كتابه ال Model :

- هنا هعمل كلاس هي extend من ال entity المقابله له وبداخله هعمل fromJson , toJson والمسؤلين عن ال contact with API .

```dart
import 'package:equatable/equatable.dart';

class MemberEntity extends Equatable {
  final String memberID;
  final String memberName;
  final String membershipDate;

  const MemberEntity({required this.memberName,required this.memberID, required this.membershipDate});

  @override
  // TODO: implement props
  List<Object?> get props => [memberName,membershipDate,memberID];
}
```

- ده ال Entity

```dart
import 'package:bader_user_app/Features/Clubs/Domain/Entities/member_entity.dart';

class MemberModel extends MemberEntity{
  // Todo: Extend From Achievement Entity
  const MemberModel({
    required super.memberName,
    required super.memberID,
    required super.membershipDate,
  });

  // Todo: From JSON to Refactor Data That come from Firestore
  factory MemberModel.fromJson({required Map<String,dynamic> json})
  {
    return MemberModel(
        memberName: json['memberName'],
        memberID: json['memberID'],
        membershipDate:json['membershipDate'],
    );
  }

  // Todo: Send Data to Firebase
  Map<String,dynamic> toJson(){
    return {
```

- وده ال Model بيحتوي ع ثلاثه اشياء الأولي ال Constructor لحتي اسند قيم لل attributes اللي ب super class واللي هنا MemberEntity تاني شئ ال fromJson وهنا استعملت ال factory named constructor لحتي ارجع instance from Model وتالت شئ هو toJson عشان لو هبعت الداتا لل API or Firebase مثلا تكون ف صيغه json or map .

```dart
  factory MemberModel.fromJson({required Map<String,dynamic> json})
  {
    return MemberModel(
        memberName: json['memberName'],
        memberID: json['memberID'],
        membershipDate:json['membershipDate'],
    );
  }

  // Todo: Send Data to Firebase
  Map<String,dynamic> toJson(){
    return {
      'memberName' : super.memberName,
      'memberID' : super.memberID,
      'membershipDate' : super.membershipDate,
    };
  }
}
```

- طبعا super قبل اسم المتغير لانه موجود ب super class بس طبعا ممكن مكتبش كلمه super لان دلوقتي اصبح كل ال attributes ف super بقت ف MemberModel ف عادي لو كتبت memberID , memberName .

خامسا كتابه Imply Repositories :

```
main.dart   post_model.dart U   post_repository_impl.dart U ×
lib > features > posts > data > repositories > post_repository_impl.dart > PostsRepositoryImpl > addPost
1  import 'package:clean_architecture_posts_app/features/posts/domain/entities/post.dart';
2  import 'package:clean_architecture_posts_app/core/error/failures.dart';
3  import 'package:clean_architecture_posts_app/features/posts/domain/repositories/posts_repository.dart';
4  import 'package:dartz/dartz.dart';
5
6  class PostsRepositoryImpl implements PostsRepository {
7    @override
8    Future<Either<Failure, Unit>> addPost(Post post) {
9      // TODO: implement addPost
10     throw UnimplementedError();
11   }
12
13   @override
14   Future<Either<Failure, Unit>> deletePost(int id) {
15     // TODO: implement deletePost
16     throw UnimplementedError();
17   }
18
19   @override
20   Future<Either<Failure, List<Post>>> getAllPosts() {
21     // TODO: implement getAllPosts
22     throw UnimplementedError();
23   }
24
25   @override
26   Future<Either<Failure, Unit>> updatePost(Post post) {
27     // TODO: implement updatePost
28     throw UnimplementedError();
```

- هعمل كلاس وهخليه ي implement لل contract Repo واللي كان فيها كل الدوال بس prototype فقط من غير body .

```
post_repository_impl.dart - clean_architecture_posts_app - Visual Studio Code
main.dart   post_model.dart U   post_repository_impl.dart 1, U ×   post_remote_data_source.dart U   post_local_data_source.dart U
lib > features > posts > data > repositories > post_repository_impl.dart > PostsRepositoryImpl > getAllPosts
5  import 'package:clean_architecture_posts_app/features/posts/domain/repositories/posts_repository.dart';
6  import 'package:dartz/dartz.dart';
7
8  class PostsRepositoryImpl implements PostsRepository {
9    final PostRemoteDataSource remoteDateSource;
10   final PostLocalDataSource localDateSource;
11
12   PostsRepositoryImpl(
13       {required this.remoteDateSource, required this.localDateSource});
14   @override
15   Future<Either<Failure, List<Post>>> getAllPosts() async {
16     // await remoteDateSource.getAllPost();
17   }
18
19   @override
20   Future<Either<Failure, Unit>> addPost(Post post) {…
24
25   @override
26   Future<Either<Failure, Unit>> deletePost(int id) {…
30
31   @override
32   Future<Either<Failure, Unit>> updatePost(Post post) {…
36 }
```

- الفكره هنا ان لو هكيش الداتا هعمل remote_data_source , local_data_source وكل اللي عليا داخل كل داله هتشك علي النت لو مفيش هستدعي الداله اللي بتجيب البوستات مثلا م local_data_source والعكس هتكون م remote_data_source .

post_model.dart | post_repository_impl.dart 1, M | post_remote_data_source.dart | post_local_data_source.dart

features > posts > data > repositories > post_repository_impl.dart > PostsRepositoryImpl > getAllPosts

```dart
import 'package:clean_architecture_posts_app/features/posts/domain/repositories/posts_repository.dart';
import 'package:dartz/dartz.dart';

class PostsRepositoryImpl implements PostsRepository {
  final PostRemoteDataSource remoteDataSource;
  final PostLocalDataSource localDataSource;

  PostsRepositoryImpl(
      {required this.remoteDataSource, required this.localDataSource});
  @override
  Future<Either<Failure, List<Post>>> getAllPosts() async {
    if (isDeviceConnected) {
      try {
        final remotePosts = await remoteDataSource.getAllPosts();
        localDataSource.cachePosts(remotePosts);
        return Right(remotePosts);
      } on ServerException {
        return Left(ServerFailure());
      }
    } else {
      try {
        final localPosts = await localDataSource.getCachedPosts();
        return Right(localPosts);
      } on EmptyCacheException {
        return Left(EmptyCacheFailure());
      }
    }

    throw UnimplementedError();
  }

  @override
```

- هنا ع سبيل المثال ف getAllPosts اول حاجه هشوف هل في نت ولا لاء عشان اعرف هجيب الداله م cacheDataSource ولا RemoteDataSource طب لو في نت مثلا هراب الداله ب try catch عشان لو ف exception وانا هنا عارف لو ف exception هيكون من نوع ServerException ف بالتالي هرجع Failure المقابل له ServerFailure وبالنسبه لل ServerException انا ف Data Source هبقي اراب الكود ب try catch عشان لو ف exception هبقي اعمل throw ServerException عشان استقبله زي ما تم توضيحه فوق ف حاله ان ف نت وف exception هيكون نوعه ServerException مثل المثال التالي :

```dart
import ...

class RemoteClubsDataSource {

  Future<String> uploadClubImageToStorage({required File imgFile}) async {
    try
    {
      Reference imageRef = FirebaseStorage.instance.ref(basename(imgFile.path));
      await imageRef.putFile(imgFile);
      return imageRef.getDownloadURL();
    }
    catch(e)
    {
      throw ServerException(exceptionMessage: "Something went wrong, try again later");
    }
  }

```

- طب بالنسبه ان اشيك علي الانترنت :

```dart
abstract class NetworkInfo {
  Future<bool> get isConnected;
}

class NetworkInfoImpl implements NetworkInfo {
  final InternetConnectionChecker connectionChecker;

  NetworkInfoImpl(this.connectionChecker);
  @override
  Future<bool> get isConnected => connectionChecker.hasConnection;
}
```

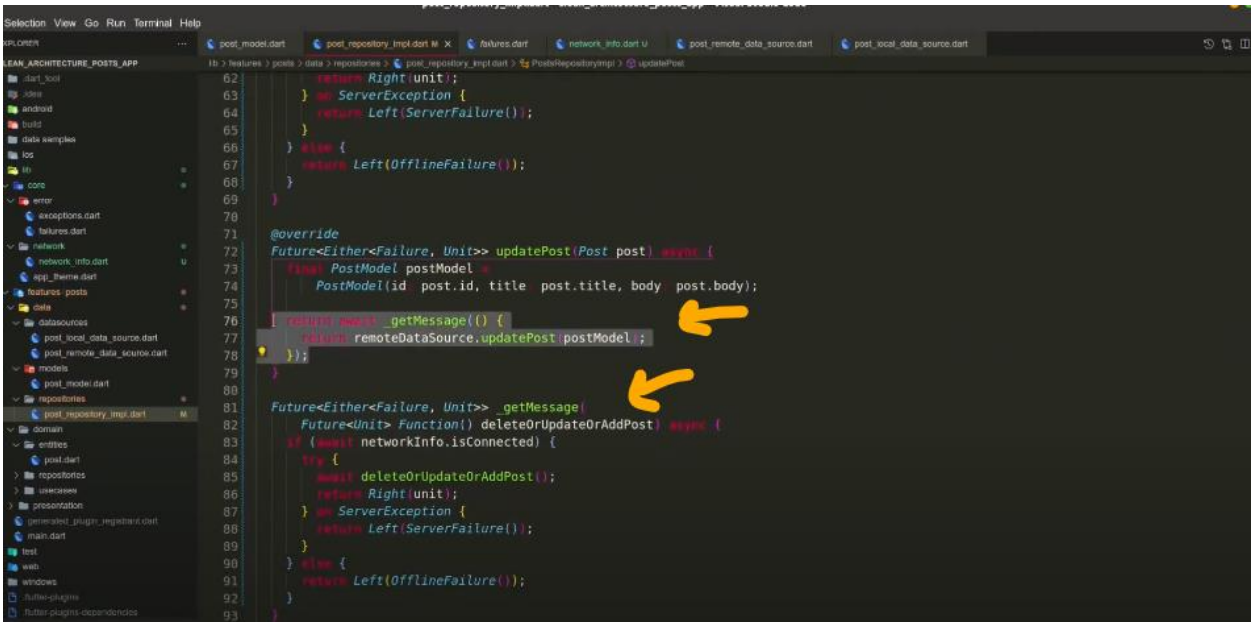
- هنا استعملت الباكدج اللي اسمها InternetConnectionChecker وجت داخل ال Core عملت فولدر لل Network وبداخله File اسمه network_info عشان احط فيه الكلاس اللي هيكون فيه الداله اللي هتشيك ع النت وهنا عملته abstract عشان ممكن اعمل كلاسين يعملوا implement له وف كل واحده استعمل باكدج مختلفه عشان لو واحده حصل فيها مشكله.

```dart
import 'package:clean_architecture_posts_app/core/network/network_info.dart';
import 'package:clean_architecture_posts_app/features/posts/data/datasources/post_local_data_source.dart';
import 'package:clean_architecture_posts_app/features/posts/data/datasources/post_remote_data_source.dart';
import 'package:clean_architecture_posts_app/features/posts/domain/entities/post.dart';
import 'package:clean_architecture_posts_app/core/error/failures.dart';
import 'package:clean_architecture_posts_app/features/posts/domain/repositories/posts_repository.dart';
import 'package:dartz/dartz.dart';

class PostsRepositoryImpl implements PostsRepository {
  final PostRemoteDataSource remoteDataSource;
  final PostLocalDataSource localDataSource;
  final NetworkInfo networkInfo;

  PostsRepositoryImpl(
      {required this.remoteDataSource, required this.localDataSource});
  @override
  Future<Either<Failure, List<Post>>> getAllPosts() async {
    if (await networkInfo.isConnected) {
      try {
        final remotePosts = await remoteDataSource.getAllPosts();
        localDataSource.cachePosts(remotePosts);
        return Right(remotePosts);
      } on ServerException {
        return Left(ServerFailure());
      }
    } else {
      try {
        final localPosts = await localDataSource.getCachedPosts();
        return Right(localPosts);
      } on EmptyCacheException {
        return Left(EmptyCacheFailure());
      }
    }
```

- هنا بقي ف imply repo هبقي اباصي object from NetworkInfo وداخل كل داله هبقي استدعي isConnected method واللي موجوده ب NetworkInfo عشان اشيك ع النت.
- بالنسبه لي ان المفروض الداله ترجع List<PostEntity> لان الداله ده تابعه لل Contract Repo واللي هو موجود ب Domain عشان كده بتعامل مع Entity وخلي بالك ان الدوال اللي ب Data Source هترجع Model مش Entity بس مفيش مشكله لان ف الاساس المودل بيعمل extends from entity .

```dart
    }
  }
}

@override
Future<Either<Failure, Unit>> addPost(Post post) async {
  final PostModel postModel =
      PostModel(id: post.id, title: post.title, body: post.body);

  if (await networkInfo.isConnected) {
    try {
      await remoteDataSource.addPost(postModel);
      return Right(unit);
    } on ServerException {
      return Left(ServerFailure());
    }
  } else {
    return Left(OfflineFailure());
  }
}

@override
Future<Either<Failure, Unit>> deletePost(int id) {...

@override
Future<Either<Failure, Unit>> updatePost(Post post) {...
}
```

- هنا ف داله اضافه بوست المفروض الداله اللي ب data_source محتاجه مودل مش entity ف عشان كده عملت object from model وبصت القيم تبع entity ، جت بعدين اشيك ع النت لو مفيش هرجع left result واللي هي OfflineFailure غير كده هنادي ع الداله اللي ب Data_Source بس هرابها ب try catch ولو ف exception هيكون ServerException ف هرجع left result واللي هي ServerFailure وهكذا بقي مع الباقي.



- ده لتحسين الكود لان ف تلت دوال تقريبا بيعملوا نفس الشئ ف عشان كده عملت داله تحت وكل اللي عليا هبقي اباصي لها بس الداله تبع كل واحده .

سادسا كتابه ال Data Source :

```dart
abstract class PostRemoteDataSource {
  Future<List<PostModel>> getAllPosts();
  Future<Unit> deletePost(int postId);
  Future<Unit> updatePost(PostModel postModel);
  Future<Unit> addPost(PostModel postModel);
}

class PostRemoteImplWithDio implements PostRemoteDataSource{}

class PostRemoteImplWithHttp implements PostRemoteDataSource {...
```

- هنا ممكن اعمل abstract class خاص ب remote data source واحط فيه الدوال تبع imply repo لان هناك هيتم استدعاؤها.
- هنا عملت abstract لان ممكن انفذ الدوال بمكتبه http مره ومره تانيه ب dio بحيث لو حصلت مشكله ف اي مكتبه التانيه تكون موجوده ب implementation .
- بنفس الفكره ممكن اعملها ف local or cache لان ممكن انفذها مثلا ب shared_pref او SQLite .
- بس طبعا ممكن اقتصر ان يكون ف كلاس واحد فيه بعمل implement لل PostRemoteDataSource

```dart
import 'package:clean_architecture_posts_app/features/posts/data/models/post_model.dart';
import 'package:dartz/dartz.dart';

abstract class PostLocalDataSource {
  Future<List<PostModel>> getCachedPosts();
  Future<Unit> cachePosts(List<PostModel> postModels);
}

class PostLocalDataSourceImpl implements PostLocalDataSource {
  @override
  Future<Unit> cachePosts(List<PostModel> postModels) {
    // TODO: implement cachePosts
    throw UnimplementedError();
  }

  @override
  Future<List<PostModel>> getCachedPosts() {
    // TODO: implement getCachedPosts
    throw UnimplementedError();
  }
}
```

- وده بالنسبه لل Cache Data Source .
- خلي بالك هنا انا في Data layer عشان كده هتعامل مع Model مش ال Entity مثل ما موضح في داله ارجاع البوستات List<PostModel> .

```dart
import 'package:dartz/dartz.dart';
import 'package:shared_preferences/shared_preferences.dart';

abstract class PostLocalDataSource {
  Future<List<PostModel>> getCachedPosts();
  Future<Unit> cachePosts(List<PostModel> postModels);
}

class PostLocalDataSourceImpl implements PostLocalDataSource {
  final SharedPreferences sharedPreferences;

  PostLocalDataSourceImpl(this.sharedPreferences);
  @override
  Future<Unit> cachePosts(List<PostModel> postModels) {
    List postModelsToJson = postModels
        .map<Map<String, dynamic>>((postModel) => postModel.toJson())
        .toList();
    sharedPreferences.setString("CACHED_POSTS", json.encode(postModelsToJson));
    return Future.value(unit);
  }

  @override
  Future<List<PostModel>> getCachedPosts() {
    // TODO: implement getCachedPosts
    throw UnimplementedError();
  }
}
```

- ❖ بالنسبه لكاش البوستات انا عاوز احول models لل List<Json> عشان احفظها ف الكاش ف ها loop on Posts وهستدعي toJson لحتي احولهم ل map وف الاخر هحفظ ال List<JsonOrMap> ده ف الكاش ع هيئه ال String وبعدين اما اجي اجيبها هبقي افكها م خلال jsonDecode .
- ❖ بالنسبه لل Future.value(unit) لو عندي حاجه مش future وعاوز اخليها future بحطها داخل القوسين مثل ال unit لان المفروض الداله بترجع Future<Unit>

features > posts > data > datasources > post_local_data_source.dart > PostLocalDataSourceImpl > getCachedPosts

```
15
16    PostLocalDataSourceImpl({required this.sharedPreferences});
17    @override
18    Future<Unit> cachePosts(List<PostModel> postModels) {
19      List postModelsToJson = postModels
20          .map<Map<String, dynamic>>((postModel) => postModel.toJson())
21          .toList();
22      sharedPreferences.setString("CACHED_POSTS", json.encode(postModelsToJson));
23      return Future.value(unit);
24    }
25
26    @override
27    Future<List<PostModel>> getCachedPosts() {
28      final jsonString = sharedPreferences.getString("CACHED_POSTS");
29      if (jsonString != null) {
30        List decodeJsonData = json.decode(jsonString);
31        List<PostModel> jsonToPostModels = decodeJsonData
32            .map<PostModel>((jsonPostModel) => PostModel.fromJson(jsonPostModel))
33            .toList();
34        return Future.value(jsonToPostModels);
35      } else {
36        throw EmptyCacheException();
37      }
38    }
39  }
40
```

- ❖ طب هنا بالنسبه لل getCachedPosts عملت jsonDecode بعدين loop on List<Json> وحولتها ل List<PostModel> .
- ❖ بما اني ف ال Data Source ف المفروض هنا برمي ال Exception عشان استقبله ف Imply Repo ف هنا مثلا لو String فاضي معناها ان مفيش داتا بالكاش اصلا ف عملت throw EmptyCacheException .

```dart
final http.Client client;

PostRemoteDataSourceImpl({required this.client});
@override
Future<List<PostModel>> getAllPosts() async {
  final response = await client.get(
    Uri.parse(BASE_URL + "/posts/"),
    headers: {"Content-Type": "application/json"},
  );

  if (response.statusCode == 200) {
    final List decodedJson = json.decode(response.body) as List;
    final List<PostModel> postModels = decodedJson
        .map<PostModel>((jsonPostModel) => PostModel.fromJson(jsonPostModel))
        .toList();

    return postModels;
  } else {
    throw ServerException();
  }
}
```

- ❖ ده بالنسبه لل Remote Data Source عملت object from httpClient لحتي اقدر استعمل الدوال اللي ب http .
- ❖ هنا المفروض قلنا بنرمي ال Exception لحتي نستقبله ب Imply Repo ونرجع ال Failure المقابل له ف هنا لو status code تبع ال response مو 200 يبقي في ServerException .
- ❖ بالنسبه لل BASE_URL + “” هنا انا بجمع ال api_base_url مع ال endpoint تبع ال request .

```dart
@override
Future<Unit> addPost(PostModel postModel) async {
  final body = {
    "title": postModel.title,
    "body": postModel.body,
  };

  final response =
      await client.post(Uri.parse(BASE_URL + "/posts/"), body: body);

  if (response.statusCode == 201) {
    return Future.value(unit);
  } else {
    throw ServerException();
  }
}
```

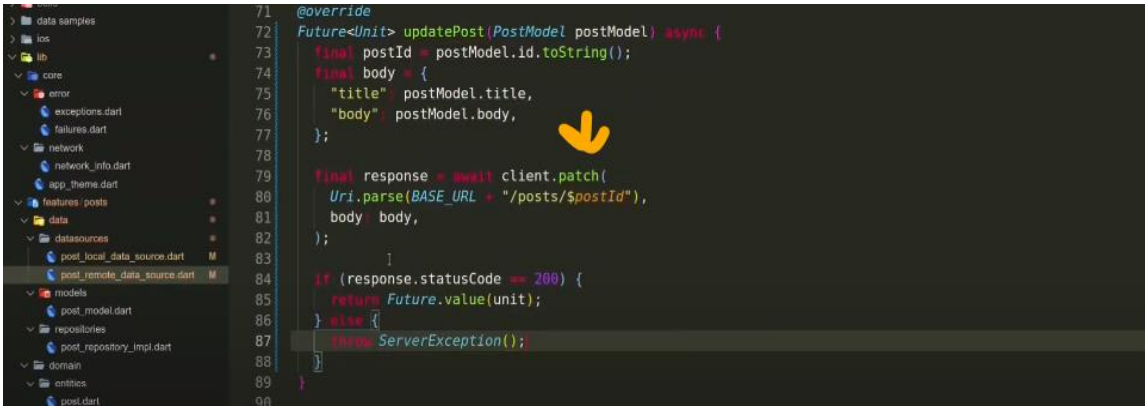
- ❖ هنا بالنسبه لإضافه بوست لو ال status code ب 201 معناها البوست انضاف غير كده يبقي ف ServerException .
- ❖ بالنسبه لل delete request او update ف ال status code لو ب 200 يبقي ال Request نفذ بشكل صحيح.
- ❖ خلي بالك ان put , patch كلاهما بيتسعمل ف حاله update داتا م خلال http بس الفرق كالتالي :

In HTTP, the PATCH and PUT methods are generally used for updating resources on the server. However, there is a subtle difference between the two methods.

The PUT method is used to completely replace an existing resource or create a new one if it doesn't exist. When you make a PUT request to the server, you must send the entire updated resource in the request body, which will replace the existing resource on the server.

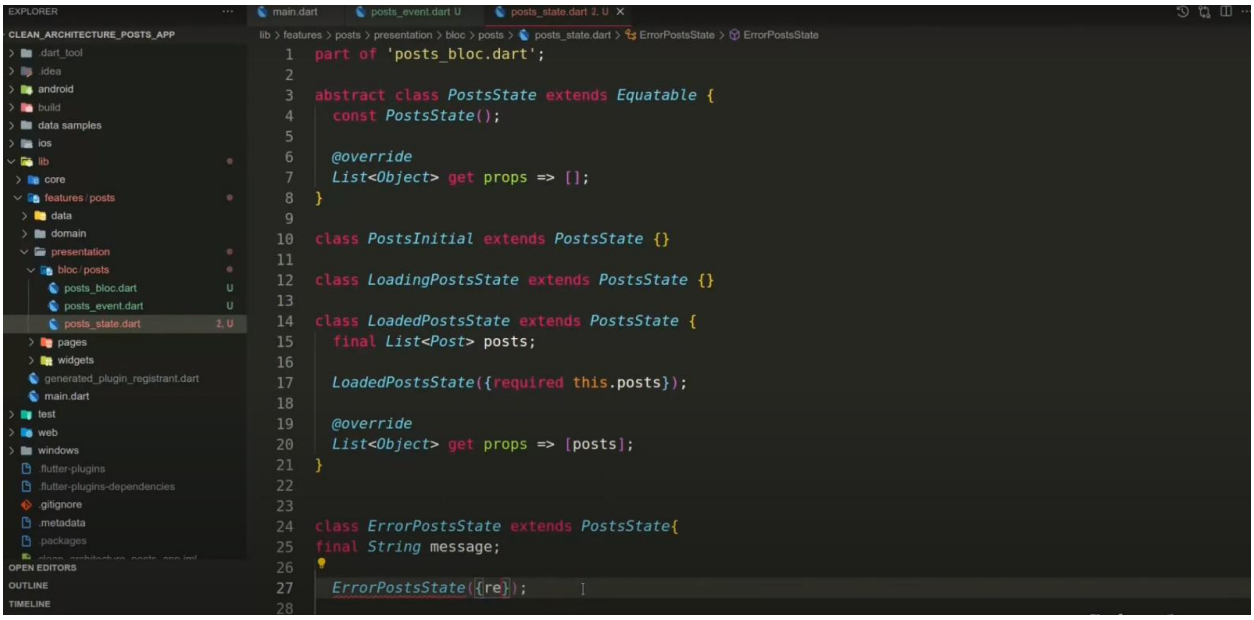
On the other hand, the PATCH method is used to partially update an existing resource. When you make a PATCH request to the server, you only need to send the changes that you want to apply to the resource. The server will then apply these changes to the existing resource and return the updated version.

In Flutter, the `http` package provides support for making HTTP requests. When you use the `http.patch` method, you are making a PATCH request to the server, and when you use the `http.put` method, you are making a PUT request.

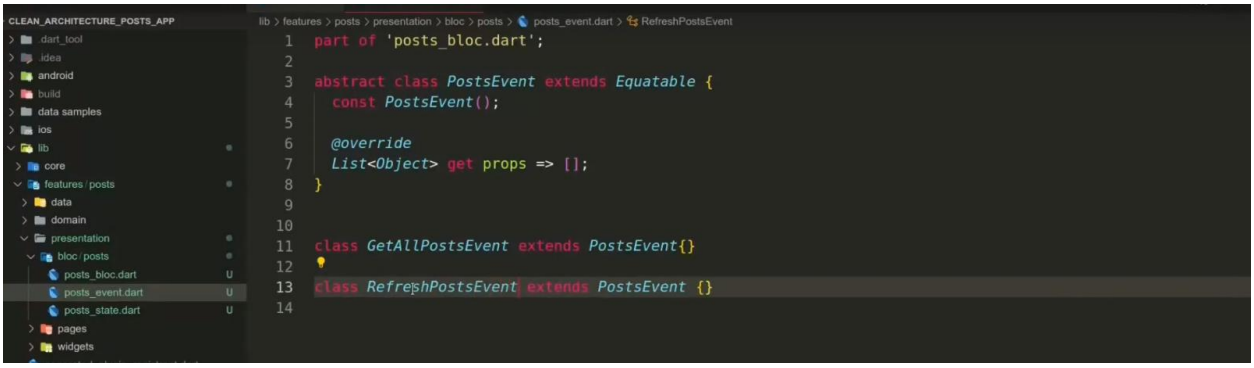


- مثل ما موضح ف المثال.
- بكده خلصنا Data Layer and Domain Layer هندخل بقي ع ال Representation .

Presentation Layer



- ده بالنسبه لملف States تبع ال Bloc لانه شغال بال Bloc ك state management .



- وده ملف ال Events لان ال Bloc بيشتغل كالتالي :
  - أما بنفذ Event بيرد بال State

```dart
import 'package:bloc/bloc.dart';
import 'package:clean_architecture_posts_app/features/posts/domain/entities/post.dart';
import 'package:equatable/equatable.dart';

part 'posts_event.dart';
part 'posts_state.dart';

class PostsBloc extends Bloc<PostsEvent, PostsState> {
  PostsBloc() : super(PostsInitial()) {
    on<PostsEvent>((event, emit) async {
      if (event is GetAllPostsEvent) {
      } else if (event is RefreshPostsEvent) {}
    });
  }
}
```

- ده شكل الكلاس تبع البلوك واللي فيه بنعمل logic انه مثلا ف حاله اذا كان ال event كذا هيروح يحكي مع UseCase واللي فيه بتنفذ الداله وليكن getPosts .

```dart
const String SERVER_FAILURE_MESSAGE = 'Please try again later .';
const String EMPTY_CACHE_FAILURE_MESSAGE = 'No Data';
const String OFFLINE_FAILURE_MESSAGE = 'Please Check your Internet Connection';
```

- دول هبقي استعملهم مع state الخاصه بال Error لحتي اعرض المسدج للمستخدم.

```dart
          (failure) {
            emit(ErrorPostsState(message: message));
          },
          (posts) {},
        );
      } else if (event is RefreshPostsEvent) {}
    });
  }

  String _mapFailureToMessage(Failure failure) {
    switch (failure.runtimeType) {
      case ServerFailure:
        return SERVER_FAILURE_MESSAGE;
      case EmptyCacheFailure:
        return EMPTY_CACHE_FAILURE_MESSAGE;

      case OfflineFailure:
        return OFFLINE_FAILURE_MESSAGE;

      default:
      return "Unexpected Error , Please try again later .";
    }
  }
}
```

- الداله ده هيكون هدفها ف حاله لو ف Failure انا هحدد بقي المسدج اللي هباصيها لل state تبع ال error بناء ع ال Failure اللي همرره للداله
- بالنسبه لل runTimeType ده داله بتعرفني ال DataType تبع المتغير.

```
class PostsBloc extends Bloc<PostsEvent, PostsState> {
  final GetAllPostsUsecase getAllPosts;
  PostsBloc({
    required this.getAllPosts,
  }) : super(PostsInitial()) {
    on<PostsEvent>((event, emit) async {
      if (event is GetAllPostsEvent) {
        emit(LoadingPostsState());

        final failureOrPosts = await getAllPosts();
        failureOrPosts.fold(
          (failure) {
            emit(ErrorPostsState(message: _mapFailureToMessage(failure)));
          },
          (posts) {
            emit(LoadedPostsState(posts: posts));
          },
        );
      } else if (event is RefreshPostsEvent) {}
    });
  }

  String _mapFailureToMessage(Failure failure) {
    switch (failure.runtimeType) {
      case ServerFailure:
        return SERVER_FAILURE_MESSAGE;
```

- هنا عشان اللي راجع م UseCase بيحمل قيمه م حاجتين في ف داله اسمها fold بتستعمل مع الحاجه اللي ممكن ترجع قيمه م اتنين مثل ما موضح بالأعلي ف بتحتوي ع two anonymous function ف الأول بيمرر ال left result واللي هنا ال Failure والتانيه ال right side واللي هنا List<Post> .
- هنا بمرر ال Posts لل LoadedPostsState عشان هبقي اعرضها من خلال ال State لان ال state بتحتوي ع attribute اسمه posts .

- ممكن عشان اخلي الكود Clean اعمل التالي :

```
lib > features > posts > presentation > bloc > posts > posts_bloc.dart > PostsBloc > PostsBloc
CLEAN_ARCHITECTURE_POSTS_APP
  failures.dart
  app_theme.dart
  features / posts
    data
    domain
      entities
      repositories
      usecases
        add_post.dart
        delete_post.dart
        get_all_posts.dart
        update_post.dart
    presentation
      bloc / posts
        posts_bloc.dart
        posts_event.dart
        posts_state.dart
      pages
      widgets
  generated_plugin_registrant.dart
  main.dart
  test
  web
  windows
  .flutter-plugins
  .flutter-plugins-dependencies
  .gitignore
  .metadata
  .packages
```

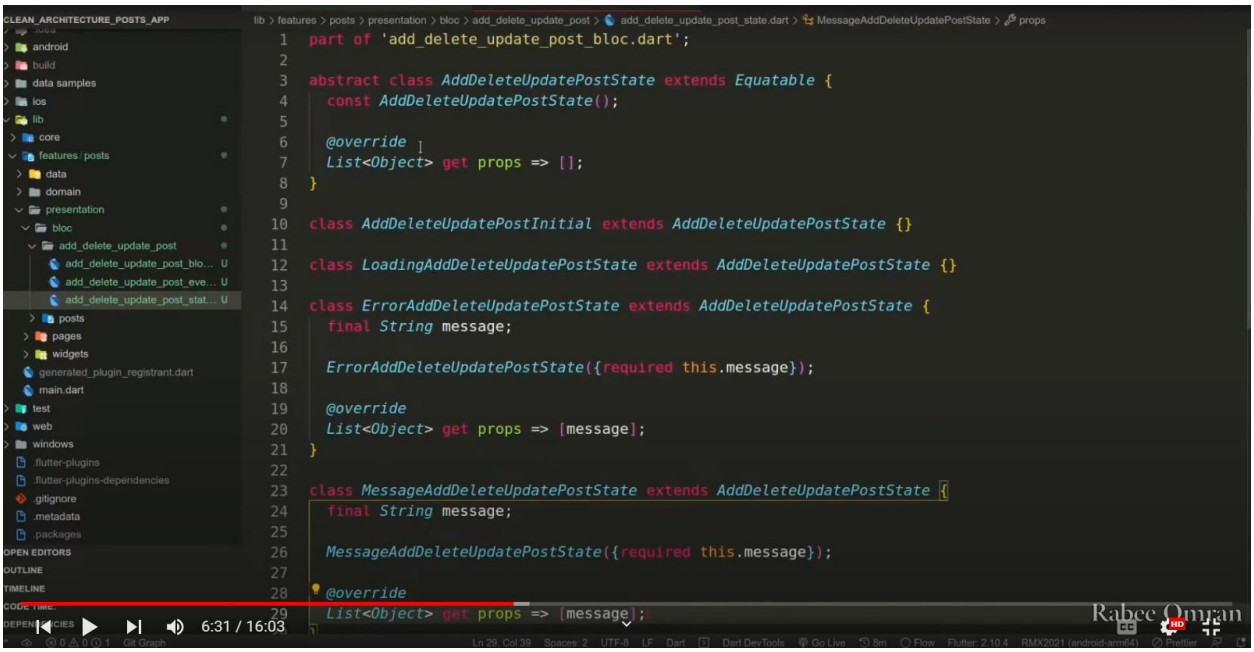
```dart
  }) : super(PostsInitial()) {
    on<PostsEvent>((event, emit) async {
      if (event is GetAllPostsEvent) {
        emit(LoadingPostsState());

        final failureOrPosts = await getAllPosts();
        emit(_mapFailureOrPostsToState(failureOrPosts));
      } else if (event is RefreshPostsEvent) {
        emit(LoadingPostsState());

        final failureOrPosts = await getAllPosts();
        emit(_mapFailureOrPostsToState(failureOrPosts));
      }
    });
  }

  PostsState _mapFailureOrPostsToState(Either<Failure, List<Post>> either) {
    return either.fold(
      (failure) => ErrorPostsState(message: _mapFailureToMessage(failure)),
      (posts) => LoadedPostsState(
        posts: posts,
      ),
    );
  }
```

- هنا عملت داله بترجع state واستدعيت الداله اللي بترجع حاجه م اتنين بداخلها وعملت return لها.
- خلي بالك ان ال Bloc , state management here بتتعامل مع Use Case

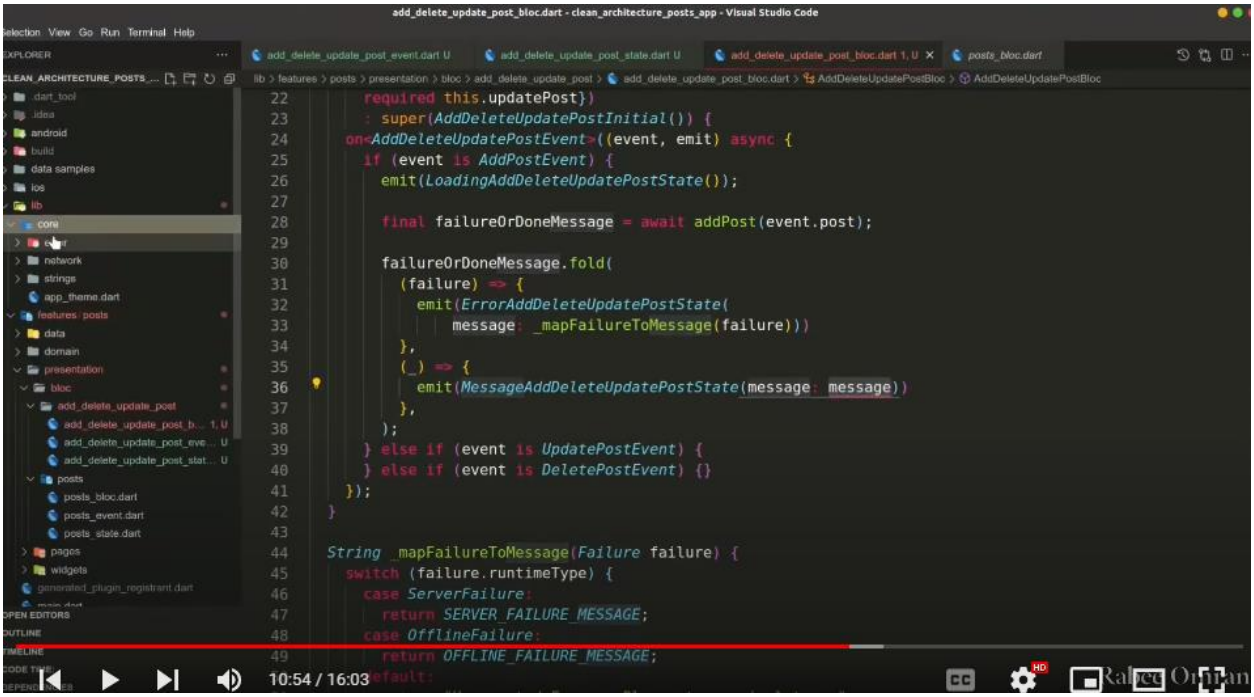
```
lib > features > posts > presentation > bloc > add_delete_update_post > add_delete_update_post_event.dart > UpdatePostEvent > AddPostEvent
CLEAN_ARCHITECTURE_POSTS_APP
  android
  build
  data samples
  ios
  lib
    core
    features / posts
      data
      domain
      presentation
        bloc
          add_delete_update_post
            add_delete_update_post_blo... U
            add_delete_update_post_eve... U
            add_delete_update_post_stat... U
          posts
        pages
        widgets
    generated_plugin_registrant.dart
    main.dart
  test
  web
  windows
  .flutter-plugins
  .flutter-plugins-dependencies
  .gitignore
  .metadata
  .packages
```

```dart
abstract class AddDeleteUpdatePostEvent extends Equatable {
  const AddDeleteUpdatePostEvent();

  @override
  List<Object> get props => [];
}

class AddPostEvent extends AddDeleteUpdatePostEvent {
  final Post post;

  AddPostEvent({required this.post});

  @override
  List<Object> get props => [post];
}

class UpdatePostEvent extends AddDeleteUpdatePostEvent {
  final Post post;

  UpdatePostEvent({required this.post});

  @override
  List<Object> get props => [post];
}
```

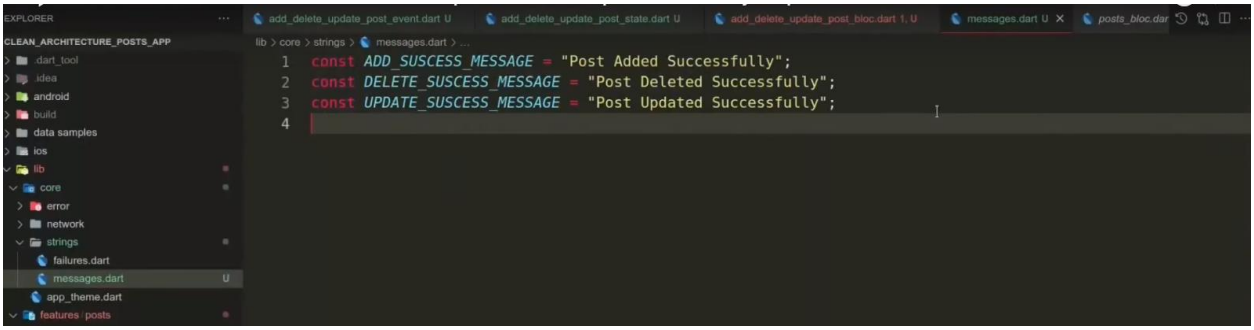
- ده شكل ال Events تبع البلوك الثاني هيكون فيه اضافه وحذف والتعديل ع البوست.



- وده شكل ال States تبع البلوك الثاني.



- وده شكل الكلاس تبع البلوك ، هنا تقريبا ال 3 use cases بترجع اما Failure or unit شان كده ف ال right side اللي راجع م الداله حطت underscore لان ما بيهمني اللي راجع.



- الأفضل ابقي اعمل مثلا file خاص ب Strings اللي زي كده ف Core .

هنتكلم بقي ع استعمال get_it ف Dependancy Injection :

- اولا كده عمليه ال Dependancy Injection معناها ان بحقن كلاس ب object م كلاس اخر كما بالمثال التالي :

```
10 import '../../../../../core/error/failures.dart';
11 import '../../../../../core/strings/failures.dart';
12
13 part 'add_delete_update_post_event.dart';
14 part 'add_delete_update_post_state.dart';
15
16 class AddDeleteUpdatePostBloc
17     extends Bloc<AddDeleteUpdatePostEvent, AddDeleteUpdatePostState> {
18   final AddPostUsecase addPost;
19   final DeletePostUsecase deletePost;
20   final UpdatePostUsecase updatePost;
21   AddDeleteUpdatePostBloc(
22       {required this.addPost,
23       required this.deletePost,
24       required this.updatePost})
25       : super(AddDeleteUpdatePostInitial()) {
26     on<AddDeleteUpdatePostEvent>((event, emit) async {
27       if (event is AddPostEvent) {
28         emit(LoadingAddDeleteUpdatePostState());
29
30         final failureOrDoneMessage = await addPost(event.post);
```

- يعني هنا مثلا ال Class ده معتمد علي ال objects اللي هحقنها له عند عمل instance منه واللي هم addPost,deletePost,updatePost .

The get_it package is a popular service locator library for Dart and Flutter applications. It provides a way to register and retrieve objects (or services) across the application, without having to manually pass them around.

In get_it, there are three main methods for registering objects: registerSingleton, registerLazySingleton, and registerFactory. Each of these methods has a different purpose and should be used based on the requirements of your application. Here's a brief overview of each method:

- ف هنستعمل الباكد ده عشان تساعدنا ف عمليه انشاء instance م الكلاسات عشان نبقي نباصيها عند الحق ع سبيل المثال ف المثال السابق كده هنبقي نباصي object from addPost مثلا.
- Get_it بتحتوي علي 3 methods مهمه ف انشاء ال objects .

- ## اولا registerSingleton :

1. `registerSingleton`: This method registers a single instance of an object that will be shared across the entire application, and will be created when the object is first requested. This is useful for objects that are expensive to create and should only be created once. For example, you might use `registerSingleton` to register a database connection object that can be shared across multiple screens.

- وده ب اختصار م خلاله بقدر اعمل Instance واحده بستعمله ع مدار التطبيق بالكامل بس الفكره هنا عند استدعاء ال get_it او instance م الباكدج هيروح يكريت instance م الحاجه ع سبيل المثال لو عاوز اعمل instance من ال SharedPreferences ف الحاله ده هكريت اوبجيكت واحد هستعمله ع التطبيق بالكامل.

- ## ثانيا registerLazySingleton :

2. `registerLazySingleton`: This method is similar to `registerSingleton`, but the object is only created when it is first requested. This is useful for objects that are not needed immediately, or that might not be needed at all. For example, you might use `registerLazySingleton` to register a logging service that is only used in certain situations.

dart Copy

```dart
// Register a lazy singleton of a logging service.
GetIt.instance.registerLazySingleton(() => Logger());

// Retrieve the lazy singleton instance.
final logger = GetIt.instance<Logger>();
```

- نفس فكره اللي قبله بس الفرق ان ال object هيتكريت فقط ف حاله ان طلبته بمعني مثلا عاوز اعمل اوبجيكت واحد من ال SharedPref بس ميتكريتش الا اما احتاج ال instance .

• ثالثا registerFactory :

3. `registerFactory`: This method registers a factory function that will be called each time the object is requested. This is useful for objects that need to be created on demand, or that might need to be recreated multiple times. For example, you might use `registerFactory` to register a service that needs to connect to a new API endpoint each time it is used.

dart Copy

```dart
// Register a factory for a service that connects to an API endpoin
t.
GetIt.instance.registerFactory(() => ApiService());

// Retrieve a new instance of the service each time it is needed.
final apiService = GetIt.instance<ApiService>();
```

In general, you should use `registerSingleton` for objects that should only be created once and shared across the application, `registerLazySingleton` for objects that are not needed immediately or might not be needed at all, and `registerFactory` for objects that need to be created on demand or recreated multiple times.

- ده بقي بستعمله مع ال objects اللي بتحتاج ان يحصل لها اعاده انشاء اكثر من مره او ع حسب الطلب علي سبيل المثال ف الشكل التالي :

Here's an example of how you might register a `CounterBloc` instance with `get_it` using `registerFactory`:

dart Copy

```dart
// Register a factory for a CounterBloc instance.
GetIt.instance.registerFactory<CounterBloc>(() => CounterBloc());

// Retrieve a new instance of the CounterBloc each time it is neede
d.
final counterBloc = GetIt.instance<CounterBloc>();

// Use the BlocProvider widget to provide the CounterBloc to its de
scendants.
BlocProvider<CounterBloc>(
  create: (context) => counterBloc,
  child: CounterScreen(),
);
```

In this example, `registerFactory` is used to register a factory function that creates a new `CounterBloc` instance each time it is requested. The `BlocProvider` widget is then used to provide the `CounterBloc` instance to its descendants, ensuring that each `Bloc` instance has its own independent state.

- هنا مثلا ال Instance اللي هيكريت من ال Bloc اكيد مش هيكون واحد ع مدار التطبيق لان في كل مره ال States مثلا بتتغير علي حسب الاسكرينه ف الفكره من ال factory انه بيكريت object جديد مع كل request عليه بس عند الحاجه.

```dart
import ...

final GetIt sl = GetIt.instance;

class ServiceLocators{
  // TODO: Make Object from Classes That I want be created
  static Future<void> serviceLocatorInitialization() async {

    // TODO: Data Source Instance ......

    // LAYOUT DATA SOURCE
    sl.registerLazySingleton<LayoutRemoteDataSource>(() => LayoutRemoteDataSource());

    // CLUBS DATA SOURCE
    sl.registerLazySingleton<RemoteClubsDataSource>(() => RemoteClubsDataSource());
    sl.registerLazySingleton<LocalClubsDataSource>(() => LocalClubsDataSource());

    // EVENTS DATA SOURCE
    sl.registerLazySingleton<RemoteEventsDataSource>(() => RemoteEventsDataSource());
    sl.registerLazySingleton<LocalEventsDataSource>(() => LocalEventsDataSource());
```

- هنا انا عملت instance من ال get_it بعدين عملت داله هيكون فيها انشاء ال objects اللي هعمل لها injection بعدين وع حسب بقي زي ما قلنا هستعمل الداله سواء registerSingleton او الاتنين الباقيين.
- طبعا التلت دوال بيقبلوا anonymous function واللي المفروض بترجع اوبجيكت من الكلاس.
- بعدين بقي لو عاوز object م حاجه انا مكريتها م خلال get_it كل اللي عليا هكتب ال instance اللي عامله من الباكدج واللي هنا مثلا ال sl وهحدد نوع الاوبجيكت وليكن LayoutRemoteDataSource مثل الشكل التالي :

```dart
// TODO: USE CASES ......

// Auth USE CASES
sl.registerLazySingleton<UpdateMyFirebaseMessagingTokenUseCase>(() => UpdateMyFirebaseMessagingTokenUseCase(authBaseRepo
sl.registerLazySingleton<RegisterUseCase>(() => RegisterUseCase(authBaseRepository: sl<AuthImplyRepository>()));
sl.registerLazySingleton<LoginUseCase>(() => LoginUseCase(authBaseRepository: sl<AuthImplyRepository>()));
```

- هنا انا داخل الداله اللي بتعمل initialize لل objects م خلال ال get_it كل اللي عليا ان كتبت instance وحددت ال type تبع ال object وممكن ف بعض الحالات محددش لان هو هيرو يأخد ال object اللي بيدور عليه زي التالي :

```dart
sl.registerLazySingleton<AuthRemoteDataSource>(() => AuthRemoteDataSource());


// TODO: Implementation Repository Instance ......
sl.registerLazySingleton<LayoutImplyRepository>(() => LayoutImplyRepository(layoutRemoteDataSource: sl()));

sl.registerLazySingleton<ClubsImplyRepository>(() => ClubsImplyRepository(remoteClubsDataSource: sl(), localClubsDataSou

sl.registerLazySingleton<EventsImplyRepository>(() => EventsImplyRepository(remoteEventsDataSource: sl(), localEventsDat

sl.registerLazySingleton<AuthImplyRepository>(() => AuthImplyRepository(authRemoteDataSource: sl()));


// TODO: USE CASES ......

// Auth USE CASES
sl.registerLazySingleton<UpdateMyFirebaseMessagingTokenUseCase>(() => UpdateMyFirebaseMessagingTokenUseCase(authBaseRepo
sl.registerLazySingleton<RegisterUseCase>(() => RegisterUseCase(authBaseRepository: sl<AuthImplyRepository>()));
sl.registerLazySingleton<LoginUseCase>(() => LoginUseCase(authBaseRepository: sl<AuthImplyRepository>()));

// CLUBS USE CASES
sl.registerLazySingleton<RequestAMembershipUseCase>(() => RequestAMembershipUseCase(clubsContractRepository: sl<ClubsImp
```

```dart
import ...

Future<void> firebaseBackgroundMessageHandler(RemoteMessage message) async {...}

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
  await FirebaseMessaging.instance.requestPermission(...);
  await ServiceLocators.serviceLocatorInitialization();    // TODO: GetIT ( Initialize for Objects from Classes )
  Bloc.observer = MyBlocObserver();
  await SharedPref.cacheInitialization();
  Constants.userID = SharedPref.getString(key: 'userID');
  debugPrint("User ID is : ${Constants.userID}");
  // Todo: Receive message on Background as app is closed
  FirebaseMessaging.onBackgroundMessage(firebaseBackgroundMessageHandler);
  runApp(Phoenix(child: const MyApp()));
}
```

- هنا بال main هستدعي بقي الداله اللي بتعمل initialize لل objects م خلال get_it .

```dart
import ...

class ClubsCubit extends Cubit<ClubsStates> {
  ClubsCubit() : super(ClubsInitialState());

  static ClubsCubit getInstance(BuildContext context) => BlocProvider.of(context);

  // TODO: Get Notifications
  List<ClubEntity> clubs = [];
  Future<void> getAllClubs({UserEntity? userEntity}) async {
    emit(GetClubsLoadingState());
    final result = await sl<GetAllClubsUseCase>().execute();
    result.fold(
          (serverFailure){
            emit(FailedToGetClubsDataState(message: serverFailure.errorMessage));
          },
          (clubsData) async
          {
            clubs = clubsData;
            if( userEntity != null && userEntity.idForClubLead == null ) await getIDForClubsIAskedForMembership(userID: us
            emit(GetClubsDataSuccessState());
```

- هنا بقي اما اعوز استدعي object انا مكريته م خلال get_it عشان اقدر اكسس علي ال كل اللي بداخل الكلاس.

```dart
///! Core

  sl.registerLazySingleton<NetworkInfo>(() => NetworkInfoImpl(sl()));

///! External
  final sharedPreferences = await SharedPreferences.getInstance();

  sl.registerLazySingleton(() => sharedPreferences);
  sl.registerLazySingleton(() => http.Client());
  sl.registerLazySingleton(() => InternetConnectionChecker());
}
```

- ده مثال اخر ع عمل object من ال SharedPreferences و http.

هندخل بقي ع UI :

```
EXPLORER
injection_container.dart   main.dart M   posts_page.dart 4, U   posts_state.dart
CLEAN_ARCHITECTURE_POSTS_APP
lib > features > posts > presentation > pages > posts_page.dart > PostsPage > _buildBody
  android
  build
  data samples
  ios
  lib
    core
    features / posts
      data
      domain
      presentation
        bloc
          add_delete_update_post
          posts
            posts_bloc.dart
            posts_event.dart
            posts_state.dart
        pages
          posts_page.dart   4, U
        widgets
      generated_plugin_registrant.dart
      injection_container.dart
      main.dart   M
  test
  web
  windows
```

```
11      return Scaffold(appBar: _buildAppbar(), body: _buildBody());
12    }
13
14    AppBar _buildAppbar() => AppBar(title: Text('Posts'));
15
16    Widget _buildBody() {
17      return Padding(
18        padding: const EdgeInsets.all(10),
19        child: BlocBuilder<PostsBloc, PostsState>(
20          builder: (context, state) {
21            if (state is LoadingPostsState) {
22              return LoadingWidget();
23            } else if (state is LoadedPostsState) {
24              return PostListWidget(posts: state.posts);
25            } else if (state is ErrorPostsState) {
26              return MessageDisplayWidget(message: state.message);
27            }
28            return LoadingWidget();
29          },
30        ), // BlocBuilder
31      ); // Padding
```

- هنا استعملت ال BlocBuilder عشان اعمل build لل UI مع ال States تبع ال Bloc .
- بالنسبه للويدجت اللي هتستعمل فقط ف UI تبع ال Feature اللي انا فيها زي PostsListWidget هحطها داخل الفولدر اللي اسمه widgets اللي داخل Presentation انما لو حاجه public زي ال LoadingWidget هحطها ف ال Widgets اللي ب Core .

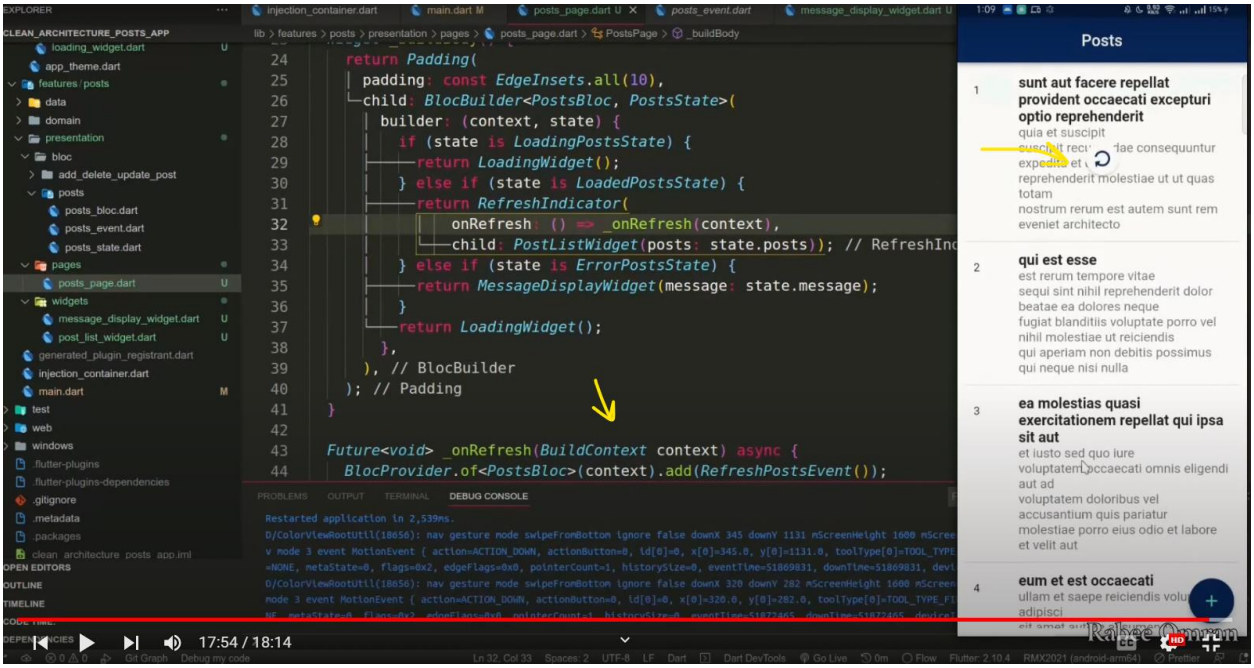
```
AN_ARCHITECTURE_POSTS_APP
lib > main.dart > MyApp > build
  loading_widget.dart   U
  app_theme.dart
  features / posts
    data
    domain
    presentation
      bloc
        add_delete_update_post
        posts
          posts_bloc.dart
          posts_event.dart
          posts_state.dart
      pages
        posts_page.dart   U
      widgets
        message_display_widget.dart   U
        post_list_widget.dart   U
  generated_plugin_registrant.dart
  injection_container.dart
  main.dart   M
  test
  web
  windows
  .flutter-plugins
  .flutter-plugins-dependencies
```

```
12    await di.init();
13    runApp(MyApp());
14  }
15
16  class MyApp extends StatelessWidget {
17    @override
18    Widget build(BuildContext context) {
19      return MultiBlocProvider(
20        providers: [
21          BlocProvider(                                   GetAllPostsEvent()
22            create: (_) => di.sl<PostsBloc>()..add(GetAllPostsEvent())),
23          BlocProvider(create: (_) => di.sl<AddDeleteUpdatePostBloc>()),
24        ],
25        child: MaterialApp(
26          debugShowCheckedModeBanner: false,
27          theme: appTheme,
28          title: 'Posts App',
29          home: PostsPage())); // MaterialApp // MultiBlocProvider
30    }
31  }
32
```

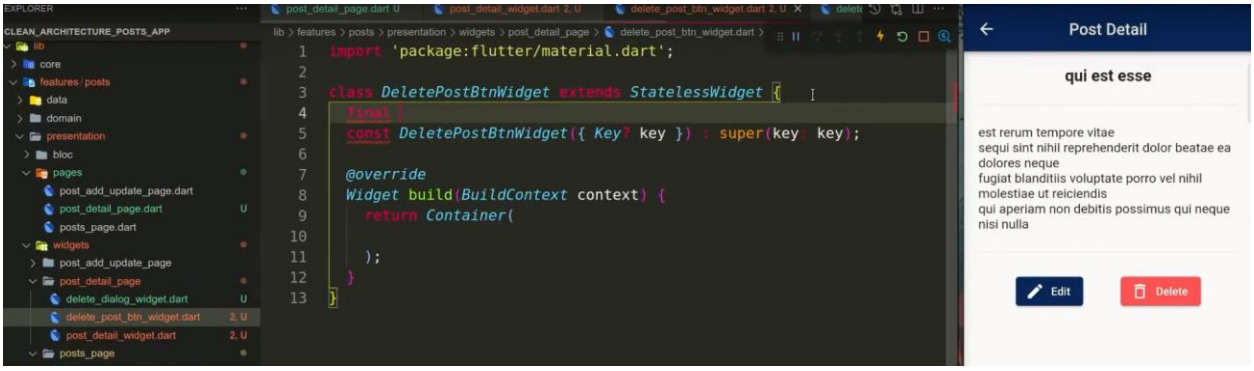
- هنا انا نديت علي event تبع getAllPosts ف الوقت اللي هيعمل فيه create instance من الكلاس لان انا شغال بالبلوك مش الكيوبيت.
- خلي بالك ان بتشك ع النت في Imply Repo لو في هيرجع الداتا من ال remote data source لو مفيش هيجيبها من local data source ع سبيل المثال هنا مثلا ف جلب البوستات.

```
AppBar _buildAppbar() => AppBar(title: Text('Posts'));

Widget _buildBody() {
  return Padding(
    padding: const EdgeInsets.all(10),
    child: BlocBuilder<PostsBloc, PostsState>(
      builder: (context, state) {
        if (state is LoadingPostsState) {
          return LoadingWidget();
        } else if (state is LoadedPostsState) {
          return RefreshIndicator(
            onRefresh: () => _onRefresh(context),
            child: PostListWidget(posts: state.posts)); // RefreshIndicator
        } else if (state is ErrorPostsState) {
          return MessageDisplayWidget(message: state.message);
        }
        return LoadingWidget();
      },
    ), // BlocBuilder
  ); // Padding
```

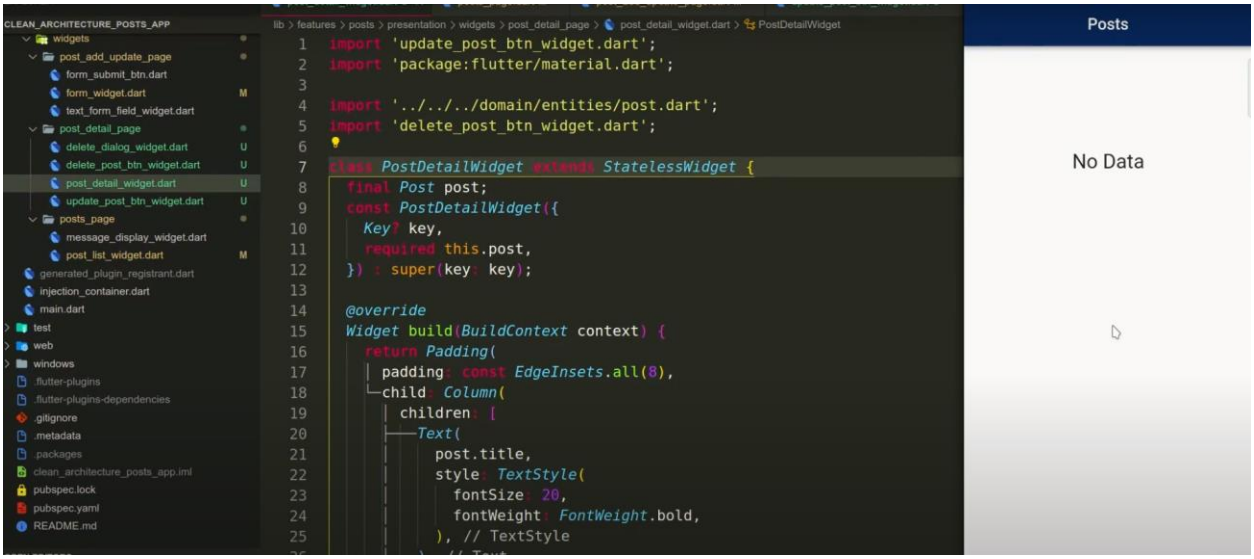
- هنا عشان اعمل refresh للبوستات مثلا ف ويدجت اسمها refreshIndicator بمرر لها ال body عاي وداخل onRefresh هستدعي الداله اللي هتعمل refresh للبوستات واللي هنا مثلا هتستدعي الداله تبع getPosts مثلا بحيث لو نزل بوستات جديده او حاجه وطبعا من مميزات ال BlocBuilder ان فيها property اسمها buildWhen بحيث مش هخليه يعمل build ع الفاضي بمعني لو البوستات اللي جايه مش نفسها اللي كانت موجوده.



- وده شكل الداله وفيها عملت instance م البلوك واستدعيت ال event المسؤل عن عمليه ال refreshing .

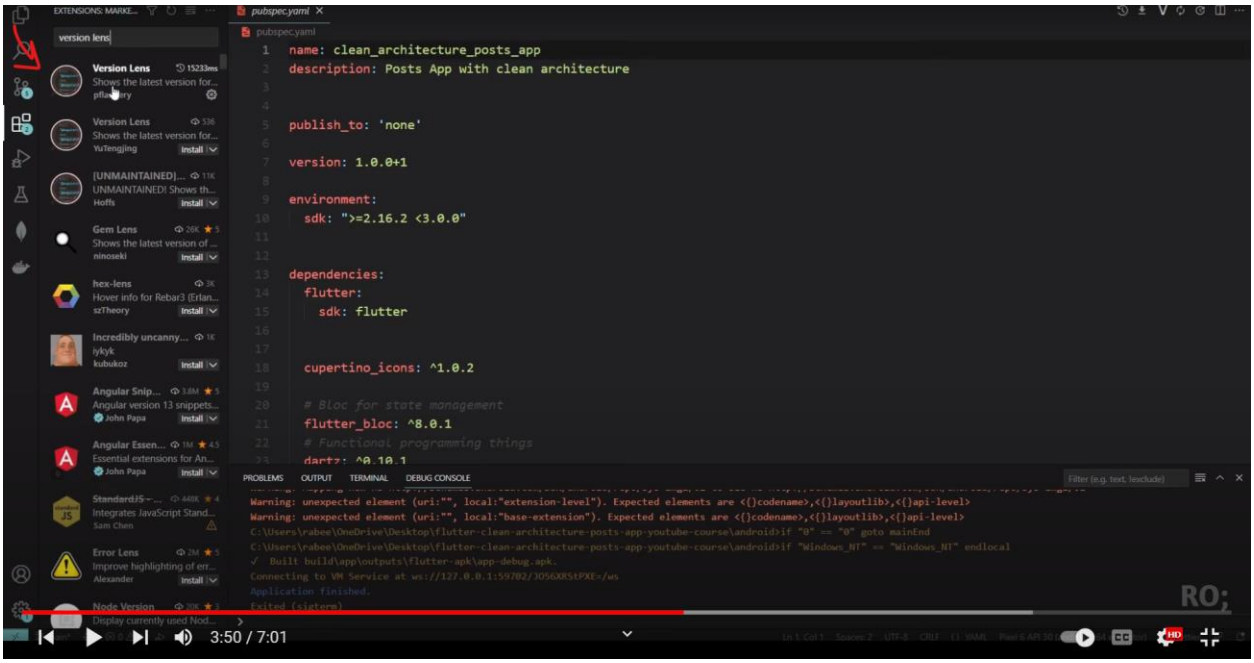


- هنا عشان الكود ميكنش كبير ف صفه معينه هأخد مثلا widget تبع deleteButton وهحطها داخل ال Widgets اللي موجوده بال Presentation بس الافضل قلنا اعمل فولدر جوا ال Widgets ب اسم الاسكرينه واحط فيها كل widgets تبعها فيه.
- ف extension in VS Code بتضبط ال import ف كل اسكرينه.



- بالنسبه للمسدج No Data ده المفروض هتظهر ف حاله ان مفيش نت بجانب ان مفيش داتا بالكاش خاصه بالبوستات.

- لو عاوز تنزل مشروع م الجيتهب من خلال CMD بتكتب git clone ومن ثم url تبع ال project .



- Version lens ده extension بتعمل update للباكدجات اللي بالمشروع لاحدث version

- بالضغط علي ال V اللي بالأعلي هيعمل update packages .