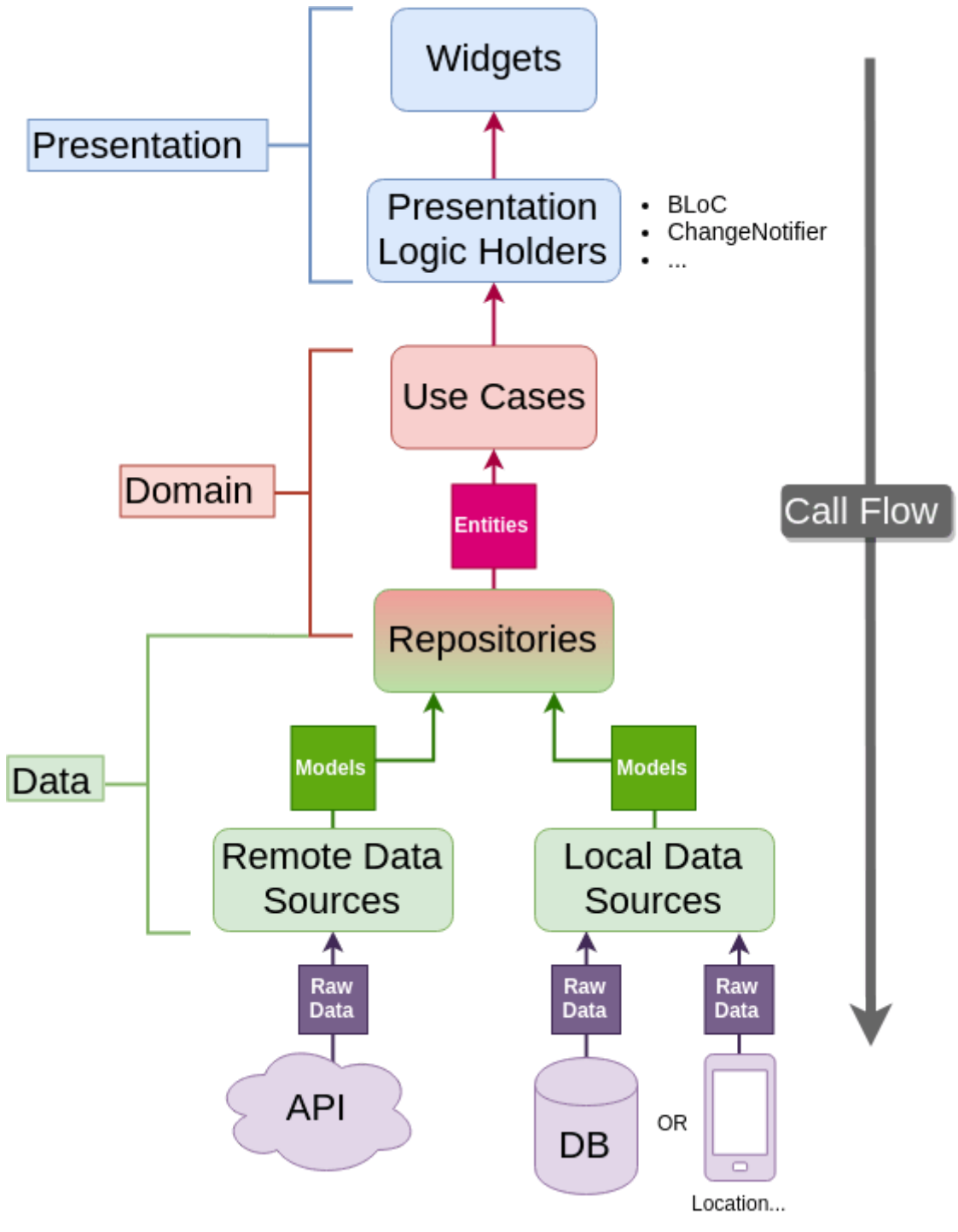
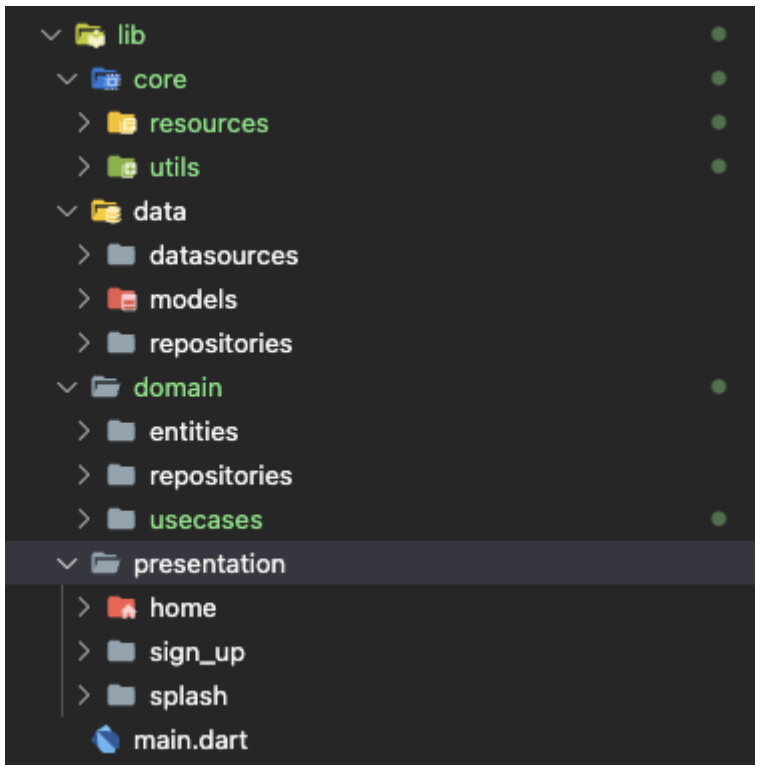
개발명세서

Design Pattern



Directory Structure



Core

> ***프로젝트 전역적으로 활용되는 요소들***
>
> - resources: Component, color, style 등 전반적인 UI 작업에 필요한 클래스들을 가짐
> - utils: Utility for the project
> - Core/resources/components는 프로젝트 내에 다른 화면들에서 범용적으로 사용되는 UI를 제외하고는 왠만하면 presentation/widget을 사용할 것

data

> ***Data layer 관련 요소들***
>
> - datasources: api 호출
> - models: DTO 클래스
> - repositories: api 클래스를 주입받아 사용하는 구현체(domain에 있는 api Interface의 implement)

domain

> ***Domain layer 관련 요소들***
>
> - entities: data 폴더의 model과 다르게 데이터 파싱, 변환에 영향을 받지않는 근본적인 데이터 구조를 나타내는 클래스 → Data layer의 models의 부모 클래스
> - repositories: api 호출 인터페이스
> - usecases: repository를 주입받아서 하나의 액션 수행하는 클래스

presentation

> ***화면 담당(View + Widget)***
>
> - 하위 디렉토리들은 기능별로 분류

Naming Convention

> Dart에서는 UppderCamelCase, lowerCamelCase, lowercase_with_underscores 네이밍 규칙을 사용합니다.
>
> - `UppderCamelCase` : 대문자로 시작하며, 각 단어의 시작 문자를 대문자로 합니다.
>   (예: UpperCamelCase)
> - `lowerCamelCase` : 소문자로 시작하며, 각 단어의 시작 문자를 대문자로 합니다. (예: lowerCamelCase)
> - `lowercase_with_underscores` : 소문자만 사용하며 단어의 구분은 언더스코어(_)를 사용합니다. 대문자나 다른 구분자는 사용하지 않습니다.

1. Folder ⇒ 모두 소문자로 표기

- core
  - resources
    - components
    - 화면을 구성할 때 필요한 것이 있다면 추가 → ex) theme
  - utils
    - constant
    - network
    - preference
    - 기능 및 설정과 관련된 명사
- presentation
  - 화면의 주요 기능 → ex) home, sign_up, splash
- data, domain package는 생략

2. File ⇒ lowercase_with_underscores

- core
  - resource
    - component
      - [특징]_[UI].dart → ex) round_button.dart
    - [element].dart → ex) colors.dart
  - util
    - constant
      - [element].dart → ex) assets.dart, string.dart
      - network, preference는 생략
- data
  - data_source
    - local
      - [table 이름]_dao.dart ⇒ ex) search_dao.dart
    - remote
      - [기능]_api.dart ⇒ ex) search_api.dart
  - model
    - [기능]_model.dart ⇒ ex) search_model.dart
  - repository
    - [기능]_repository_impl.dart ⇒ ex) search_repository_impl.dart

- domain
  - entity
    - [기능].dart ⇒ search.dart
  - repository

        - [기능]_repository.dart ⇒ ex) search_repository.dart
    - usecase
        - [기능]_use_case.dart ⇒ ex) search_use_case.dart

- presentation
    - [기능]_view.dart ⇒ search_view.dart
    - [기능]_view_model.dart ⇒ search_view_model.dart
    - [기능]_provider.dart ⇒ search_provider.dart
    - [기능]_[UI]_widget.dart ⇒ search_widget.dart

3. Class, Enum, Typedef, Type, extensions ⇒ UpperCamelCase

```
/**
      공통(file 이름 => class identifiers)

      ex)search_view.dart => class SearchView
         round_button.dart => class RoundButton
**/

class SliderMenu { ... }
class HttpRequest { ... }

typedef Predicate<T> = bool Function(T value);

extension MyFancyList<T> on List<T> { ... }
extension SmartIterable<T> on Iterable<T> { ... }


// Metadata Annotation으로 사용되는 Class의 경우에도 동일
class Foo {
  const Foo([arg]);
}

@Foo(anArg)
class A { ... }

@Foo()
class B { ... }
```

4. Function ⇒ lowerCamelCase

- 동사+명사 형태로 구성

```
/**
*    Description: [함수에 대한 설명 요약]
*
*   @param: [파라미터]
*   @return: [return type]
*/
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
);
```

5. Variable ⇒ lowerCamelCase

- enum, 상수

```
const pi = 3.14;
const defaultTimeout = 1000;

class Dice {
    static final numberGenerator = Random();
}

enum ResultState {
  idle,
  loading,
  success,
  failure
}
```

- 한 단계 상위 범주 용어(ex.Error)와 하위 용어들(ex.Error 종류)

```
[잘못된 예]
SERVER_TIMEOUT
NO_RESULT
BAD_REQUEST
SERVER_ALLOWED_REQUESTS_EXCESS


[좋은 예]
ERROR_SERVER_TIMEOUT
ERROR_NO_RESULT
ERROR_BAD_REQUEST
ERROR_SERVER_ALLOWED_REQUESTS_EXCESS
```

- private 변수일 때 ⇒ _ prefix

```
private final String _apiKey = "59b7552335bc1a8efb72bd1da8351e2d";
private String _baseUrl = "http://www.creadto.com";
```

- 변수이름은 해당 변수가 어떤 기능에 사용되거나 쓰이는지 자세하게 명시

```
[잘못된 예]
// 스크린 최대 높이를 480으로 지정함
int h = 480


[좋은 예]
int screenHeightMax = 480
```