



Flutter 앱 개발 실전 - MVVM & Test



[수업 목표]

- 소프트웨어 아키텍처 이해하기
- MVVM 이해 및 구현
- 소프트웨어 테스트 이해하기
- 단위 & 위젯 & 통합 테스트 구현

[목차]

- [01. 소프트웨어 아키텍처](#)
- [02. MVVM](#)
- [03. MVVM 구현](#)
- [04. 소프트웨어 테스트](#)
- [05. 단위 테스트](#)
- [06. 위젯 테스트](#)
- [07. 통합 테스트](#)
- [최종코드](#)



모든 토글을 열고 닫는 단축키

Windows :

`ctrl` + `alt` + `t`

Mac :

`cmd` + `alt` + `t`

01. 소프트웨어 아키텍처

▼ 소프트웨어 아키텍처



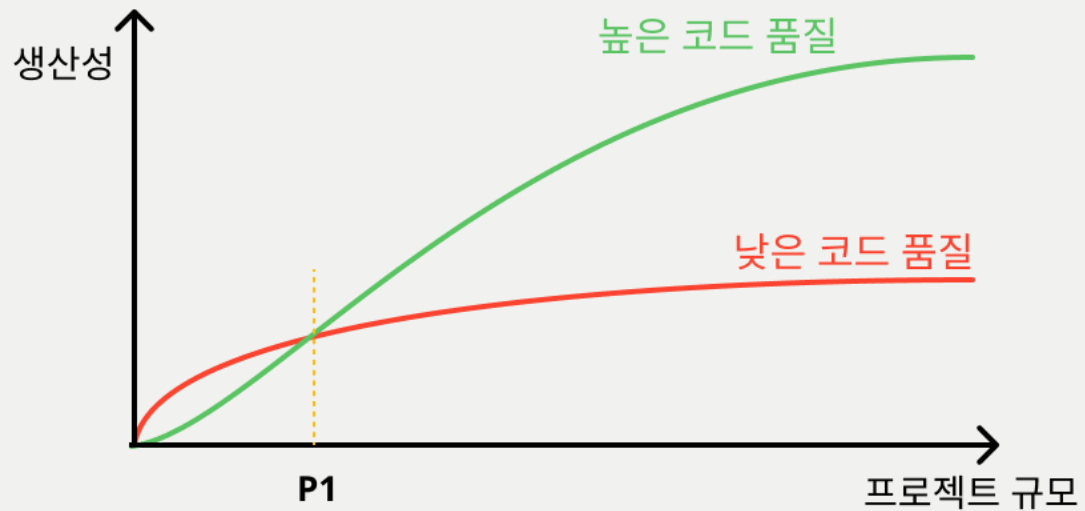
소프트웨어 아키텍처(Software Architecture)란, 구성요소들 사이에서 유기적 관계를 표현하고 소프트웨어의 설계와 업그레이드를 통제하는 지침과 원칙이다. (출처 - 위키백과)



소프트웨어를 여러 모듈로 나누고, 모듈간 규칙을 만들어 변경 사항에 대응하는 방법



소프트웨어 아키텍처 없이 마음대로 코드를 작성하는 경우 초반에 잠시 생산성이 더 높을 수 있지만, 프로젝트 규모가 커짐에 따라 생산성이 저하됩니다.



소프트웨어 아키텍처 없이 프로젝트를 진행하는 경우

- 어떤 코드가 어디에 있는지 예측이 어렵다.
- 기존에 만들어둔 코드를 재활용하기 어렵다.
- UI만 변경하고 싶지만, 관련 없는 코드도 수정해야 한다.
- DB만 변경하고 싶지만, 새로 만드는 게 빠를 것 같다.



소프트웨어 아키텍처를 적용한 프로젝트의 경우

- 어떤 코드가 있는지 예측 가능하다.
- 기존에 만들어둔 모듈을 재활용할 수 있다.
- UI를 변경시, 관련 없는 다른 로직을 수정할 필요가 없다.
- DB를 변경할 수 있다.



소프트웨어 아키텍처를 도입하면, 보다 유연한 소프트웨어를 만들어 유지보수 및 생산성 향상에 도움이 됩니다.

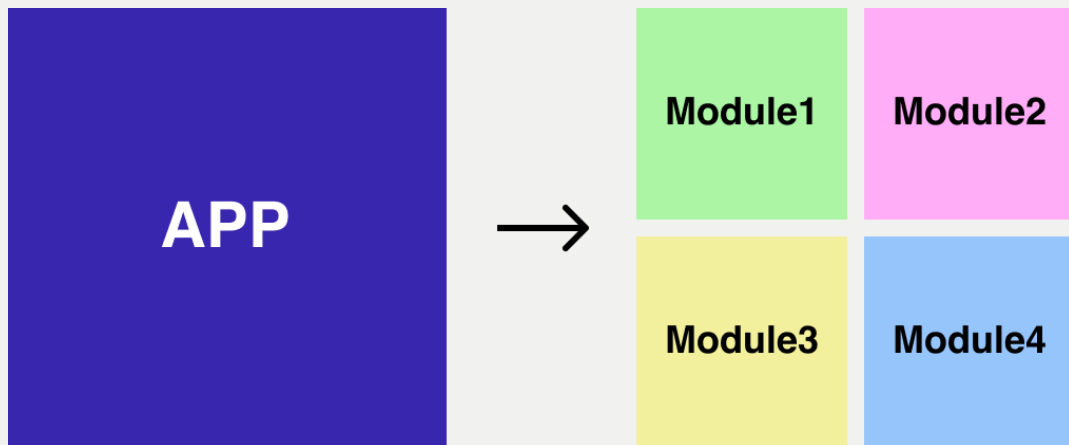


소프트웨어 아키텍처는 **관심사 분리(Separation Of Concerns, SOC)**를 통해 구현됩니다.

▼ 관심사 분리



관심사 분리(Separation Of Concerns, SOC)란, 소프트웨어를 하나의 큰 덩어리로 보지 않고 개별 모듈의 조합으로 **경계를 나누고**, **모듈간 약속을 정의하는 것**을 의미합니다.





전기 배선 을 콘센트 와 플러그 로 경계를 나누고, 표준이라는 약속을 따라 상호 호환되도록 만든 것도 관심사 분리로 볼 수 있습니다.

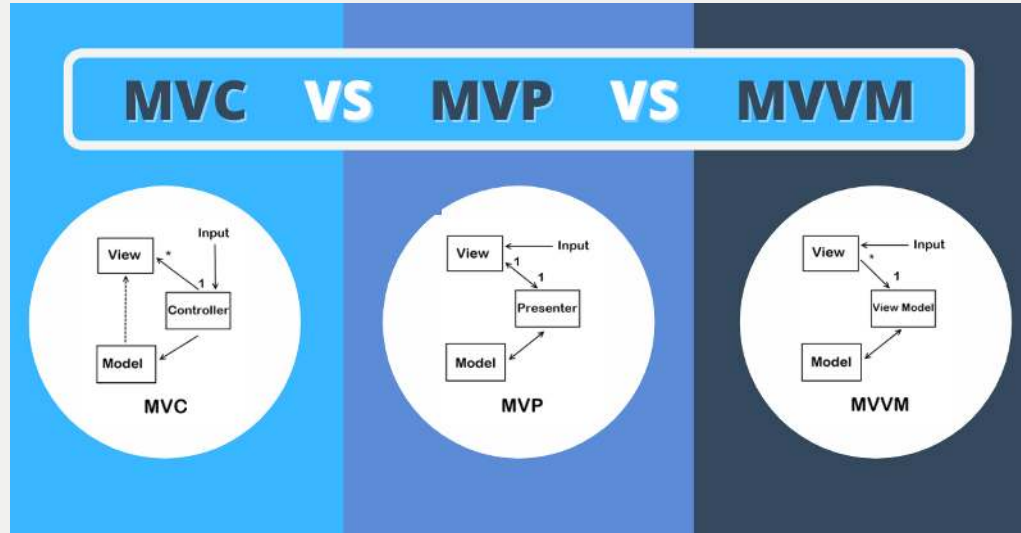


- ▼ **제품을 만들 때, 콘센트는 신경쓸 필요가 없다.**
 - 역할에 따라 분리되어 있다.
 - 협업하기 좋다.
- ▼ **콘센트를 바꿀 필요가 없이, 다양한 플러그를 끼울 수 있다.**
 - 비즈니스 로직 변경 없이, UI를 쉽게 변경할 수 있다.
 - 비즈니스 로직 변경 없이, DB를 쉽게 변경할 수 있다.
- ▼ **제품이 고장 나도, 콘센트는 정상 작동한다.**
 - 모듈의 문제가 전체 서비스로 전파되지 않는다.
 - 문제의 범위를 좁힐 수 있다.



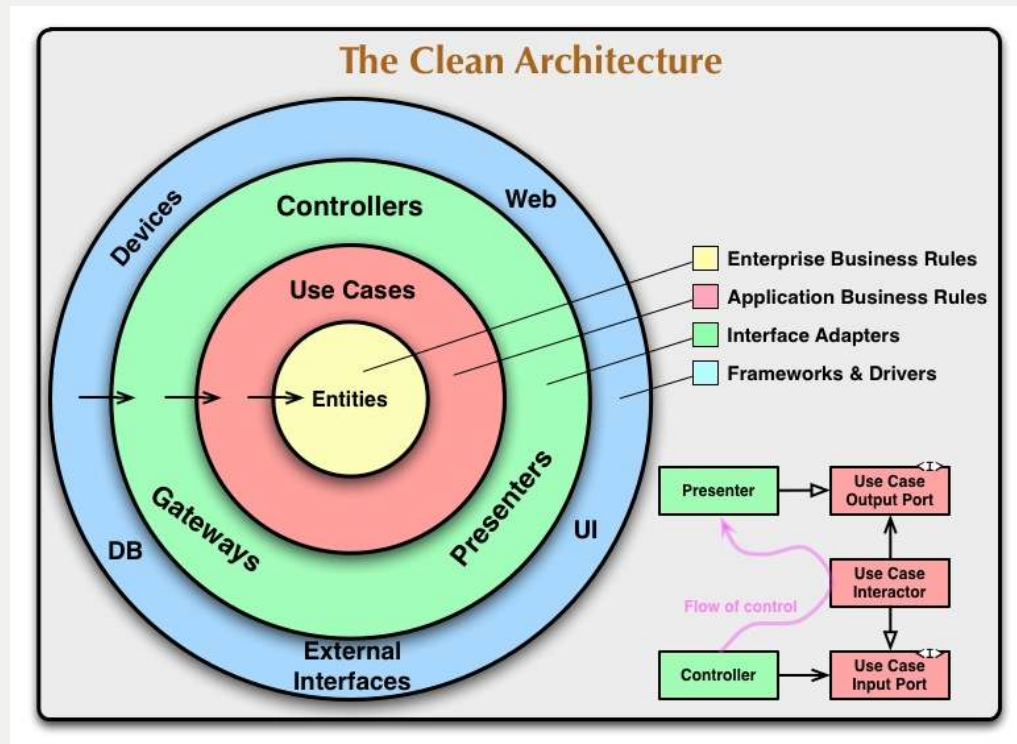
다양한 아키텍처 패턴이 존재하며, 각 패턴마다 제시하는 관심사 분리 방법이 다릅니다.

▼ MVC & MVP & MVVM



출처 - thirdrocktechkno.com

▼ Clean Architecture

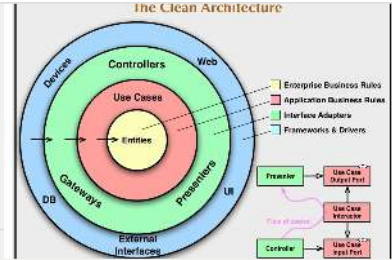


출처 - cleancoder.com

Clean Coder Blog

Over the last several years we've seen a whole range of ideas regarding the architecture of systems. These include: Though these architectures all vary somewhat in their

<https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>



▼ 추천 아키텍처



관심사 분리를 디테일하게 할수록 대규모 아키텍처입니다. **대규모 아키텍처**일수록 많은 혜택을 누릴 수 있지만, 그만큼 **지켜야 할 약속과 작성해야 하는 코드가 많아집니다.**



아키텍처 도입시 **프로젝트 규모에 맞는 아키텍처를 선택하는 것이 좋습니다.** 서비스 요구사항에 비해 시스템 구조를 복잡하게 구조화하는 것을 **오버엔지니어링(Overengineering)**이라고 부릅니다.



Flutter 프로젝트의 경우, **MVVM**으로 시작하고 대규모 프로젝트에선 **MVVM**과 **Clean Architecture**를 함께 사용하는 방법을 권장 드립니다.

02. MVVM

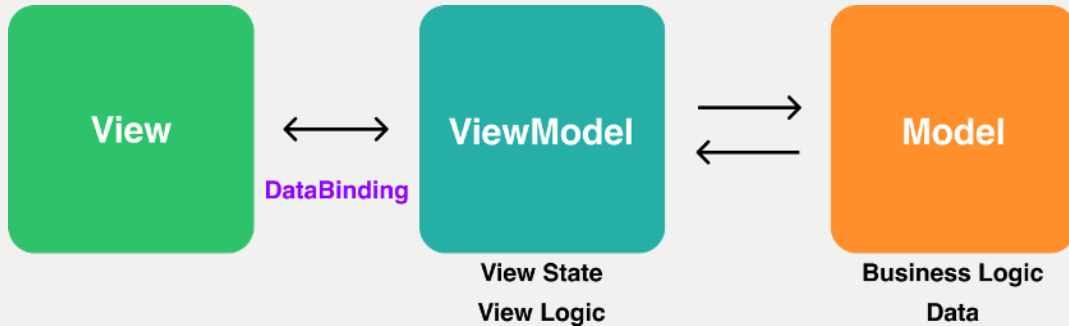
▼ MVVM 이론



아키텍처가 어떤 목적을 달성하기 위한 것인지를 이해하고 코드를 이해하는게 좋습니다.



MVVM은 소프트웨어를 Model / View / ViewModel로 관심사 분리하여 **View**를 쉽게 변경할 수 있도록 만들어줍니다.



- **View** : UI 담당
- **ViewModel** : View 상태 및 로직 담당
- **Model** : 비즈니스 로직 & 데이터 입출력 담당



데이터바인딩(DataBinding)

- **ViewModel**이 **View**를 직접 갱신하지 않고, **View**가 알아서 갱신되는 방법
- **ViewModel**이 **View**를 직접 갱신하지 않기 때문에, **ViewModel**이 **View**를 몰라도 됨




ViewModel이 **View**를 모른다.

- **ViewModel**에 **View** 관련 코드가 없다.
- **ViewModel**이 **View**에 의존성이 없다.
- **View**가 변경되어도 **ViewModel**에 영향을 끼치지 못한다.
- **View**를 변경할 때 **ViewModel**을 신경 쓸 필요가 없다.
- **View**를 쉽게 변경할 수 있다.

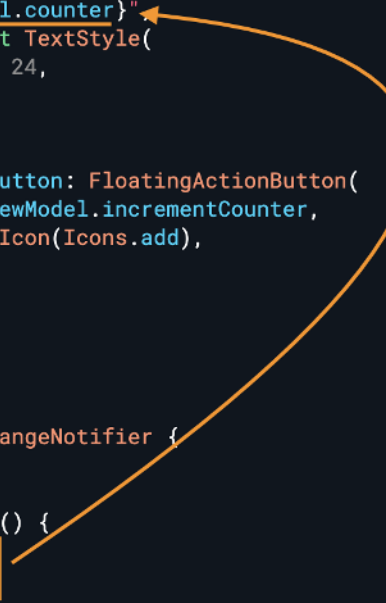


데이터바인딩은 다양한 방법으로 구현할 수 있으며, 그 중 하나로 Provider 패키지를 이용한 방법이 있습니다.

DartPad

 <https://dartpad.dev/?id=f105d93f7d5db03331fc36e634cd23>
56

```
15▼ class View extends StatelessWidget {
16   @override
17▼  Widget build(BuildContext context) {
18     ViewModel viewModel = context.watch<ViewModel>();
19     return MaterialApp(
20       debugShowCheckedModeBanner: false,
21       home: Scaffold(
22         body: Center(
23           child: Text(
24             "${viewModel.counter}"
25             style: const TextStyle(
26               fontSize: 24,
27             ),
28           ),
29         ),
30         floatingActionButton: FloatingActionButton(
31           onPressed: viewModel.incrementCounter,
32           child: const Icon(Icons.add),
33         ),
34       ),
35     );
36   }
37 }
38
39▼ class ViewModel with ChangeNotifier {
40   int counter = 0;
41
42▼  void incrementCounter() {
43    counter += 1;
44    notifyListeners();
45  }
46 }
47
```



- **ViewModel**의 `counter` 변수의 값을 변경한 뒤 `notifyListeners();`를 호출하면, **View**에서 `context.watch`로 바인딩 되어있기 때문에 **View**가 갱신됩니다.
- **ViewModel**에는 **View** 관련 코드가 없기 때문에 **View** 변경시 **ViewModel**을 수정할 필요가 없습니다.




MVVM에 대한 보다 상세한 내용은 아래 링크를 참고해 주세요.

모델-뷰-뷰모델 - 위키백과, 우리 모두의 백과사전

모델-뷰-뷰 모델(model-view-viewmodel, MVVM)은 하나의 소프트웨어 아키텍처 패턴으로- 마크업 언어 또는 GUI 코드로 구현하는-그래픽 사용자 인터페이스(뷰)의 개발을 비즈니스 로직 또는 백-엔드 로직(모델)로부터 분리시켜서 뷰가 어느 특정한 모델 플랫폼에 종속되지 않도록 해준다. MVVM의 뷰 모델은 값 변환기인데, 이는 뷰 모델이 모델
W https://ko.wikipedia.org/wiki/%EB%AA%A8%EB%8D%B8-%EB%B7%B0-%EB%B7%B0%EB%AA%A8%EB%8D%B8#cite_note-7

Introduction to Model/View/ViewModel pattern for building WPF apps

Model/View/ViewModel is a variation of Model/View/Controller (MVC) that is tailored for modern UI development platforms where the View is the responsibility of a designer rather than a classic developer. The designer is
 <https://learn.microsoft.com/ko-kr/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps>



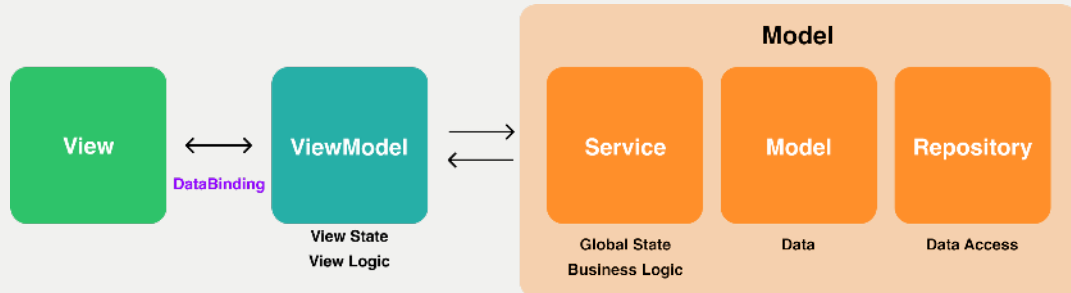
▼ MVVM 구현 방법



아키텍처는 관심사 분리 청사진을 제시할 뿐 직접적인 프로젝트 구현 방법을 제시하진 않습니다. 따라서 동일한 아키텍처 패턴을 사용하더라도 구현 방법은 조직 및 프로젝트마다 다를 수 있습니다.



MVVM 패턴을 아래와 같이 구현해 보도록 하겠습니다.



- DataBinding
 - Provider 패키지를 이용해 구현
- ViewModel
 - 각 페이지마다 하나의 ViewModel을 구현
 - View의 상태와 로직을 관리
- Model을 세분화 합니다.
 - **Service** : 앱의 전역 상태 및 비즈니스 로직 담당
 - **Model** : 데이터 클래스 담당
 - **Repository** : 데이터 요청 로직 담당

▼ MVVM with Provider



Provider를 Widget Tree 상에서 어느 위치에 주입하느냐에 따라 다르게 동작합니다. 아래 DartPad에서 예제를 확인해 봅시다.

DartPad

<https://dartpad.dev/?id=b6f92459317428a485e276bee55049de>



Root에 주입된 Provider

- App이 종료될 때 까지 사라지지 않습니다.
- 하위 위젯 어디서든 접근 가능합니다.

```
4 void main() {  
5   runApp(  
6     MultiProvider(  
7       providers: [  
8         /// MyProvider in root  
9         /// - lazy init.  
10        /// - not disposed.  
11        ChangeNotifierProvider(  
12          create: (context) => MyProvider('MyProvider in Root'),  
13        ),  
14      ],  
15      child: MaterialApp(  
16        debugShowCheckedModeBanner: false,  
17        home: Builder(builder: (context) {
```

Provider in Root

Provider



MaterialApp



View

View에 주입된 Provider

- View가 사라질 때 함께 사라집니다.
- 해당 View 하위 위젯에서만 접근 가능합니다.

```
86 class View2 extends StatelessWidget {  
87   const View2({super.key});  
88  
89   @override  
90   Widget build(BuildContext context) {  
91     /// MyProvider in view  
92     /// - disposed with the view.  
93     return ChangeNotifierProvider(  
94       create: (context) => MyProvider('MyProvider in View'),  
95       child: Consumer<MyProvider>(  
96         builder: (context, viewModel, child) {  
97           return Scaffold(  
98             appBar: AppBar(  
99               title: Text('View2'),  
100             ),  
101             body: child,  
102           ),  
103         ),  
104       ),  
105     );  
106   }  
107 }
```

Provider in View

MaterialApp



View

Provider



Service는 앱의 전역 상태를 담당하고 여러 화면에서 접근하기 때문에 **Root에 주입**하고, 특정 View에서만 접근하는 **ViewModel**은 메모리 절약을 위해 **View에 주입**하겠습니다.

03. MVVM 구현

▼ 프로젝트 준비



이전 강의에서 만든 House of Tomorrow 프로젝트를 MVVM 아키텍처 패턴으로 개선해 보도록 하겠습니다.



이전 강의의 소스 코드가 없으신 분은 아래 링크에서 코드를 다운받아주세요.

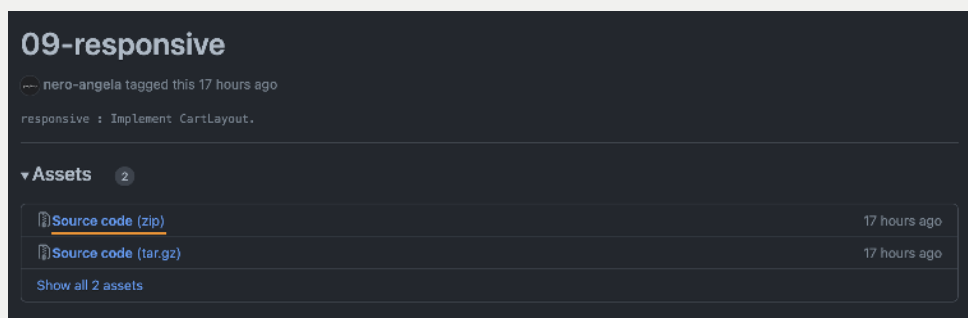
Release 09-responsive · nero-angela/flutter_house_of_tomorrow
responsive : Implement CartLayout.

https://github.com/nero-angela/flutter_house_of_tomorrow/releases/tag/09-responsive

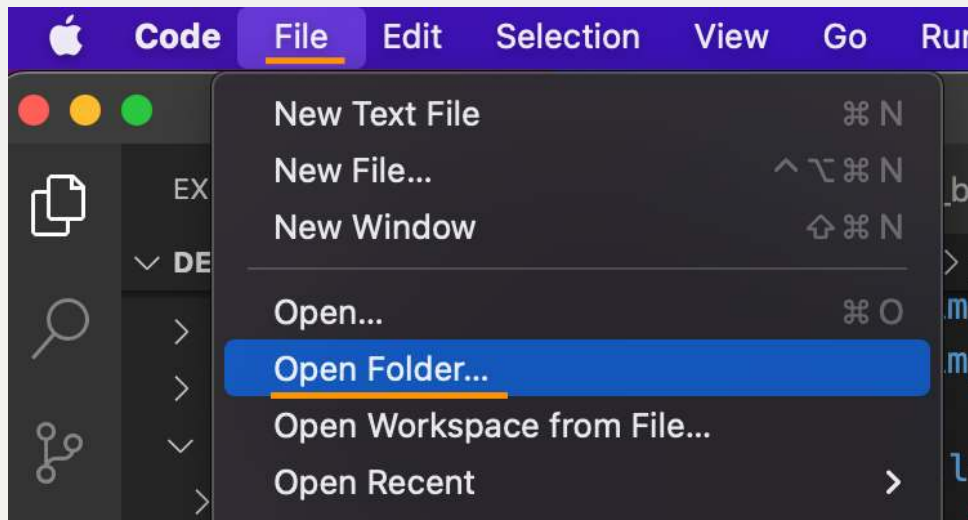


▼ 다운 받은 소스코드를 실행하는 방법

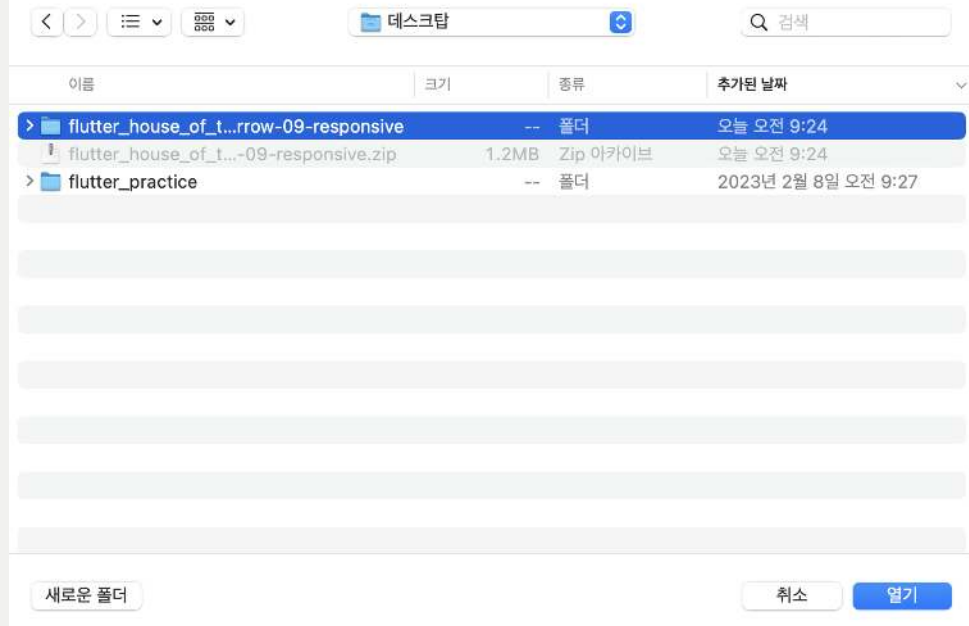
1. 링크에 접속한 뒤 **Source code (zip)** 파일을 다운받아 주세요.



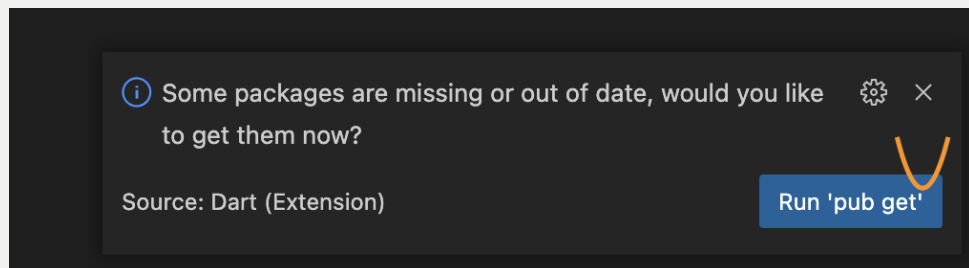
2. 다운 받은 파일의 압축을 풀어주세요. 참고로 압축 해제한 파일 이름 및 경로를 변경해도 됩니다.
3. VSCode를 실행시켜 주세요.
4. **File** → **Open Folder** 를 선택해 주세요.



5. 압축 해제한 폴더를 선택한 뒤 폴더 이름을 **house_of_tomorrow** 로 변경 **열기** 를 눌러주세요.



6. VSCode 우측 하단에 `pub get` 팝업이 뜨면 클릭해 주세요.

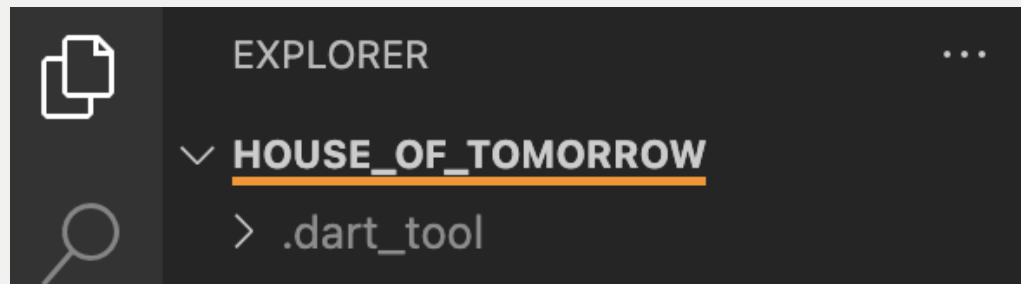


만약 팝업이 사라져 버렸다면, Terminal에 `flutter pub get` 명령어를 실행하시면 됩니다.

1. VSCode를 실행한 뒤 이전 강의에서 만든 `house_of_tomorrow` 프로젝트를 실행해 주세요.



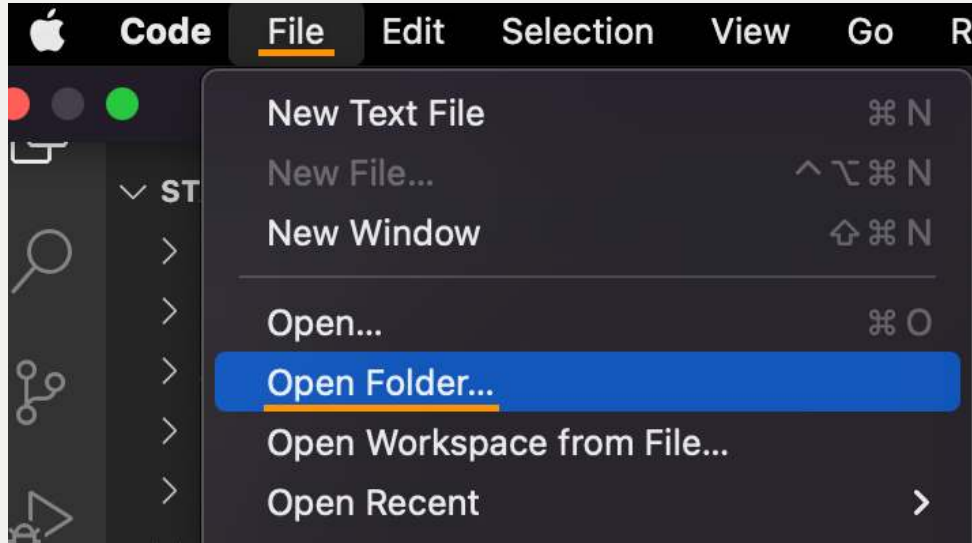
VSCode 실행시 좌측 상단에 `HOUSE_OF_TOMORROW` 라고 되어 있으면 정상적으로 실행된 것입니다.



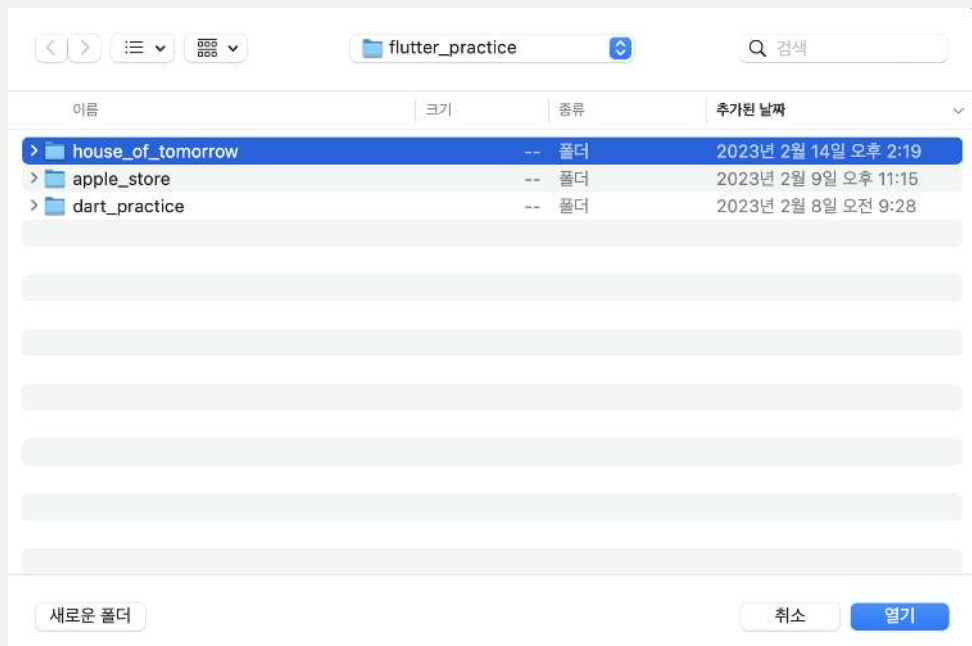


만약 다른 프로젝트가 열려 있다면, 다음과 같이 실행해 주세요.

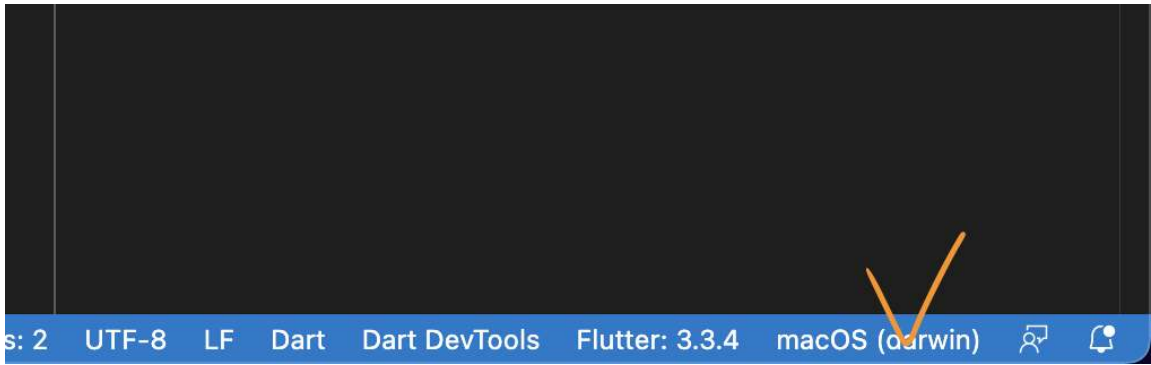
1. **File** → **Open Folder...** 를 선택해 주세요.



2. 이전에 만든 **house_of_tomorrow** 프로젝트를 선택한 뒤 **열기** 를 클릭해 주세요.



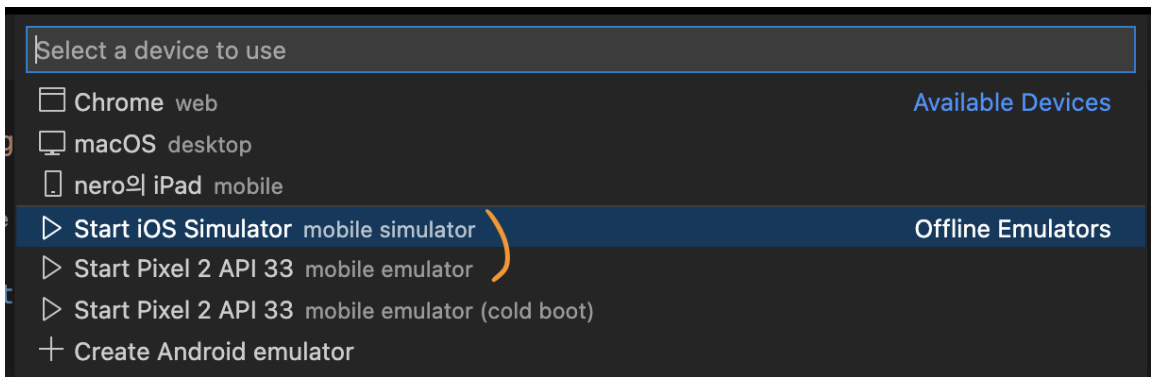
2. VSCode 우측 하단에 현재 선택된 에뮬레이터를 클릭해 주세요. (아래 사진과 다를 수 있습니다)



3. iOS 또는 Andorid 기기를 선택해 주세요.



참고로 iOS 기기는 MacOS에서만 이용 가능합니다.

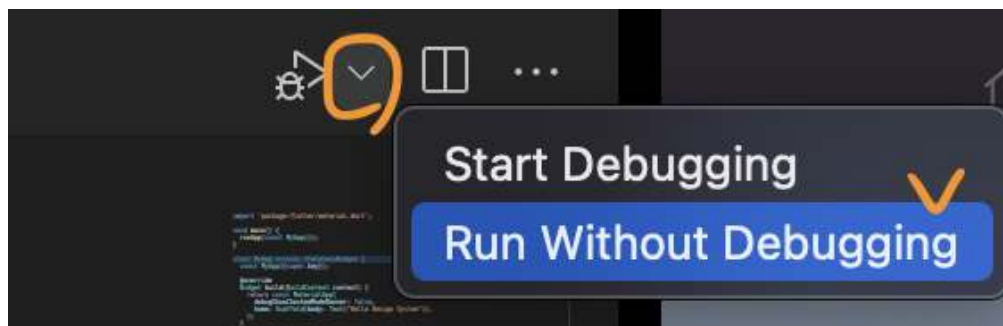


4. VSCode 우측 상단 화살표 → **Run Without Debugging** 을 선택해 주세요.

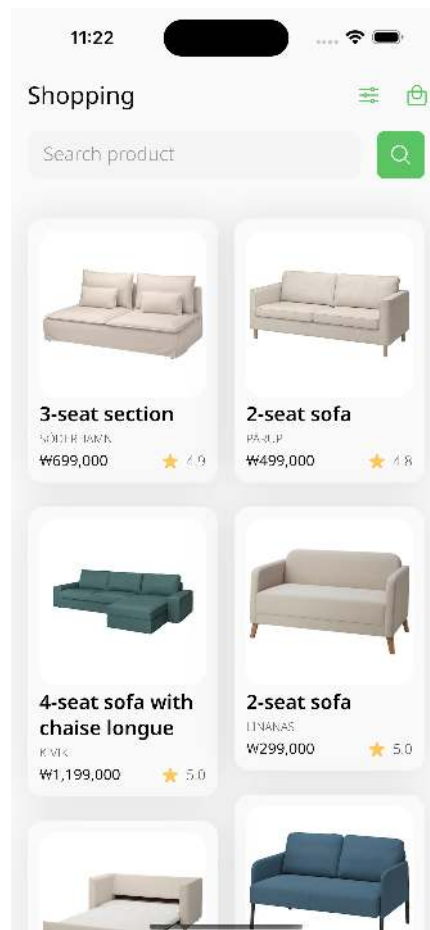


Start Debugging 으로 진행하는 경우 에러가 발생하는 지점에서 코드 실행이 멈추기 때문에 수업에선 **Run Without Debugging** 으로 진행하도록 하겠습니다.

- **Run Without Debugging** 단축키 : **Ctrl + F5**



아래와 같은 화면이 나오면 준비 완료!



▼ 코드 분석



사전에 `ViewModel` 과 `Repository` 를 제외한 다른 부분은 관심사 분리를 구현해 두었습니다.

```
✓ src
  > model
  > service
  ✓ view
    ✓ cart
      > widget
      📄 cart_view.dart
    ✓ product
      > widget
      📄 product_view.dart
    ✓ shopping
      > widget
      📄 shopping_view.dart
```



각 화면의 `widget` 폴더 밑에있는 위젯들은 모든 이벤트를 부모 위젯에서 처리하도록 구현되어 있습니다.

```
class CartBottomSheet extends StatelessWidget {  
  const CartBottomSheet({  
    super.key,  
    required this.totalPrice,  
    required this.selectedCartItemList,  
    required this.onCheckoutPressed,  
  });  
  
  final String totalPrice;  
  final List<CartItem> selectedCartItemList;  
  final void Function() onCheckoutPressed;
```

```
class ProductBottomSheet extends StatelessWidget {  
  const ProductBottomSheet({  
    super.key,  
    required this.count,  
    required this.product,  
    required this.onCountChanged,  
    required this.onAddToCartPressed,  
  });  
  
  final int count;  
  final Product product;  
  final void Function(int count) onCountChanged;  
  final void Function() onAddToCartPressed;
```

내부에 상태나 로직을 가지고 있지 않은 `widget` 들은 재사용하기 편합니다.



`ShoppingView`, `ProductView`, `CartView` 에 모든 로직이 구현되어 있기 때문에 `View` 에 섞여있는 상태와 로직을 `ViewModel` 로 분리하는 방식으로 리팩터링(refactoring, 결과의 변경 없이 코드의 구조를 재조정함)을 진행하겠습니다.

▼ BaseView & BaseViewModel



모든 View에 있는 상태와 로직을 분리하여 ViewModel을 만들 예정입니다. 시작하기 앞서 **ViewModel의 공통 로직을 관리하는 BaseViewModel** 클래스를 만들어 봅시다.

1. **view** 폴더 밑에 **base_view_model.dart** 파일을 생성해 주세요.

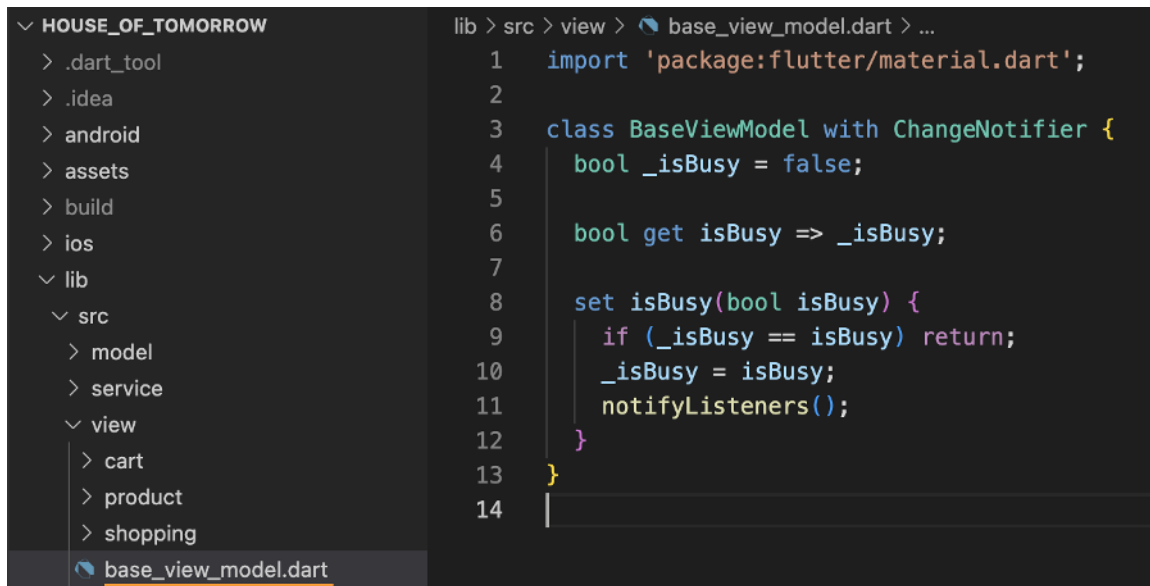
▼ 코드스니펫 - **base_view_model.dart** : 최종

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

class BaseViewModel with ChangeNotifier {
  bool _isBusy = false;

  bool get isBusy => _isBusy;

  set isBusy(bool isBusy) {
    if (_isBusy == isBusy) return;
    _isBusy = isBusy;
    notifyListeners();
  }
}
```





`isBusy` 속성은 View에서 시간이 걸리는 작업 진행시 유저에게 로딩 애니메이션을 보여 주기 위한 속성으로 모든 `ViewModel`에 공통으로 사용될 수 있으므로 `BaseViewModel`에 구현하였습니다.



`_isBusy` 값이 변경되는 경우 항상 `notifyListeners();`를 호출하도록 Getter와 Setter를 이용하여 캡슐화를 구현하였습니다.

2. 다음으로 `view` 폴더 밑에 `base_view.dart` 파일을 만들고 다음과 같이 코드를 작성해 주세요.

▼ 코드스니펫 - `base_view.dart` : 시작

```
import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/view/base_view_model.dart';
import 'package:provider/provider.dart';

class BaseView<T extends BaseViewModel> extends StatelessWidget {
  const BaseView({
    super.key,
    required this.viewModel,
    required this.builder,
  });

  final T viewModel;
  final Widget Function(BuildContext context, T viewModel) builder;

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => viewModel,
      child: Consumer<T>({
        builder: (context, viewModel, child) {
          return builder(context, viewModel);
        },
      ),
    );
  }
}
```

```
lib > src > view > base_view.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:house_of_tomorrow/src/view/base_view_model.dart';
3 import 'package:provider/provider.dart';
4
5 class BaseView<T extends BaseViewModel> extends StatelessWidget {
6   const BaseView({
7     super.key,
8     required this.viewModel,
9     required this.builder,
10   });
11
12   final T viewModel;
13   final Widget Function(BuildContext context, T viewModel) builder;
14
15   @override
16   Widget build(BuildContext context) {
17     return ChangeNotifierProvider(
18       create: (context) => viewModel,
19       child: Consumer<T>({
20         builder: (context, viewModel, child) {
21           return builder(context, viewModel);
22         },
23       }), // Consumer
24     ); // ChangeNotifierProvider
25   }
26 }
```



ViewModel은 각 페이지별로 다르게 전달할 예정이므로 Dart 제네릭(Generic)을 이용하여 `BaseViewModel`을 상속 받았다면 전달 가능하도록 만들었습니다.



각 View에서 ViewModel을 아래와 같이 주입하여 사용할 예정입니다. 중복되는 코드를 `BaseView` 라는 위젯으로 분리하였습니다.

```
84 class View2 extends StatelessWidget {
85   @override
86   Widget build(BuildContext context) {
87     /// ViewModel in view
88     /// - disposed with the view.
89     return ChangeNotifierProvider(
90       create: (context) => ViewModel('ViewModel in view'),
91       child: Consumer<ViewModel>(
92         builder: (context, viewModel, child) {
93           return Scaffold(
94             appBar: AppBar(title: const Text('View2')),
95             body: Center(
96               child: Text(
97                 "${viewModel.counter}",
98                 style: const TextStyle(
99                   fontSize: 24,
100                 ),
101             ),
102           ),
103           floatingActionButton: FloatingActionButton(
104             onPressed: viewModel.incrementCounter,
105             child: const Icon(Icons.add),
106           ),
107         );
108       ),
109     );
110   };
111 }
112 }
```

ViewModel in View

MaterialApp

View

ViewModel

`BaseView` 를 사용하면 아래와 같이 간단하게 위 코드를 줄일 수 있습니다.

```
@override
Widget build(BuildContext context) {
  return BaseView(
    viewModel: ShoppingViewModel(),
    builder: (context, viewModel) => HideKeyboard(
      child: Scaffold(
        appBar: AppBar(
```

▼ ShoppingViewModel



shopping_view.dart 에 존재하는 코드를 다음과 같이 분류할 수 있습니다.

```
23 class _ShoppingViewState extends State<ShoppingView> {
24   List<Product> productList = [];
25   final TextEditingController textController = TextEditingController();
26
27   String get keyword => textController.text.trim();
28
29   Future<void> searchProductList() async {
30     try {
31       final res = await NetworkHelper.dio.get(
32         'https://gist.githubusercontent.com/nero-angela/d16a5078c7959bf5abf6a9e0
33       );
34
35       setState(() {
36         productList = jsonDecode(res.data).map<Product>((json) {
37           return Product.fromJson(json);
38         }).where((product) {
39           /// 키워드가 비어있는 경우 모두 반환
40           if (keyword.isEmpty) return true;
41
42           /// name이나 brand에 키워드 포함 여부 확인
43           return "${product.name}${product.brand}"
44             .toLowerCase()
45             .contains(keyword.toLowerCase());
46         }).toList();
47       });
48     } catch (e, s) {
49       log('Failed to searchProductList', error: e, stackTrace: s);
50     }
51   }
52
53   @override
54   void initState() {
55     super.initState();
56     searchProductList();
57   }
58
59   @override
60   Widget build(BuildContext context) {
61     > return HideKeyboard( // HideKeyboard ...
124   }
125 }
```

View State

Data Access & View Logic

View



하나의 파일에 여러 관심사의 코드가 섞여 있는 경우, 유지 보수가 어려워집니다.

- 재사용 불가능
- 중복 코드 발생
- 하나의 클래스가 담당하는 기능이 점점 많아짐



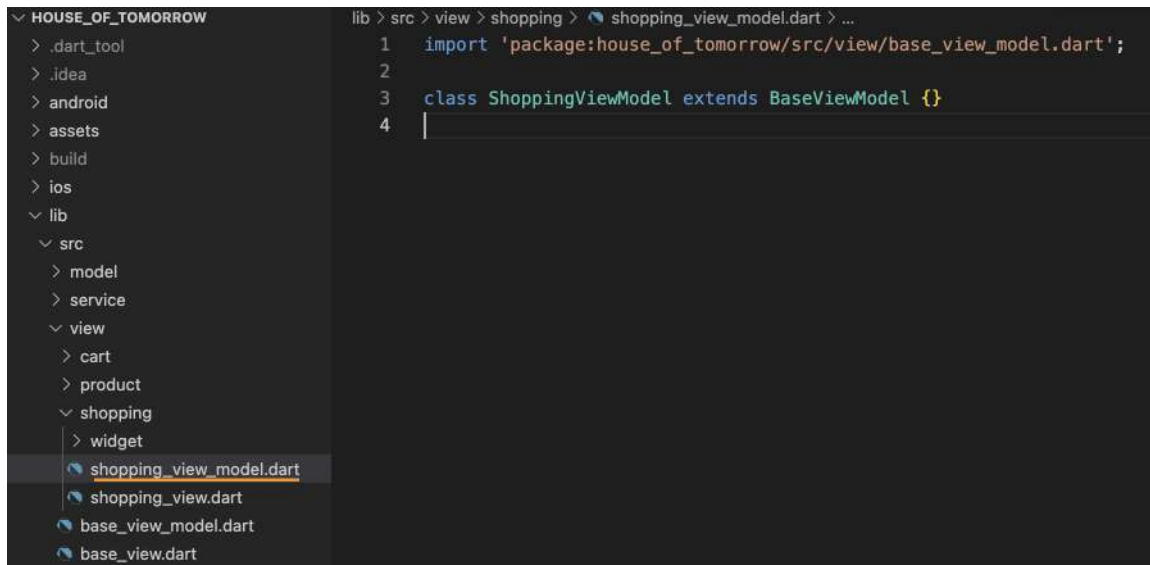
`shopping_view_model.dart` 파일을 만들고, `shopping_view.dart` 파일에서 화면을 담당하는 `build()` 메소드를 제외한 다른 로직들을 모두 `ShoppingViewModel` 로 옮겨봅시다.

1. `view/shopping` 폴더 밑에 `shopping_view_model.dart` 파일을 만들고 다음과 같이 코드를 작성해 주세요.

▼ 코드스니펫 - `shopping_view_model.dart` : 시작 코드

```
import 'package:house_of_tomorrow/src/view/base_view_model.dart';

class ShoppingViewModel extends BaseViewModel {}
```



2. `shopping_view.dart` 에 `BaseView` 를 이용하여 `ShoppingViewModel()` 을 등록해 주세요.

▼ 코드스니펫 - `shopping_view.dart` : build 메소드 변경 후

```

return BaseView(
  viewModel: ShoppingViewModel(),
  builder: (context, viewModel) => HideKeyboard(
    child: Scaffold(
      appBar: AppBar(
        title: Text(S.current.shopping),
        actions: [
          /// 설정 버튼
          Button(
            icon: 'option',
            type: ButtonType.flat,
            onPressed: () {
              showModalBottomSheet(
                context: context,
                isScrollControlled: true,
                builder: (context) {
                  return const SettingBottomSheet();
                },
              );
            },
          ),
          /// 카트 버튼
          const CartButton(),
        ],
      ),
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(
              horizontal: 16,
              vertical: 8,
            ),
            child: Row(
              children: [
                /// 검색
                Expanded(
                  child: InputField(
                    controller: textController,

```

```

        onClear: searchProductList,
        onSubmitted: (text) => searchProductList(),
        hint: S.current.searchProduct,
      ),
    ),
    const SizedBox(width: 16),

    /// 검색 버튼
    Button(
      icon: 'search',
      onPressed: searchProductList,
    ),
  ],
),
),

/// ProductCardList
Expanded(
  child: productList.isEmpty
    ? const ProductEmpty()
    : ProductCardGrid(productList),
),
],
),
),
),
);

```

```

60
61 @override
62 Widget build(BuildContext context) {
63+   return BaseView(
64+     viewModel: ShoppingViewModel(),
65+     builder: (context, viewModel) => HideKeyboard(
66       child: Scaffold(
67         appBar: AppBar(
68           title: Text(S.current.shopping),

```



`builder` 에서 두 번째 매개 변수로 전달되는 `viewModel` 은 `ShoppingViewModel` 의 인스턴스입니다.

3. `shopping_view.dart` 파일에서 아래 표시한 부분의 코드를 `shopping_view_model.dart` 파일로 잘라내 옮겨 주세요. `initState()` 의 `searchProductList()` 는 주석 처리해 주세요.

```

23 class _ShoppingViewState extends State<ShoppingView> {
24     List<Product> productList = [];
25     final TextEditingController textController = TextEditingController();
26
27     String get keyword => textController.text.trim();
28
29     Future<void> searchProductList() async {
30         try {
31             final res = await NetworkHelper.dio.get(
32                 'https://gist.githubusercontent.com/nero-angela/d16a5078c7959bf5abf6a9e0
33             );
34
35             setState(() {
36                 productList = jsonDecode(res.data).map<Product>((json) {
37                     return Product.fromJson(json);
38                 }).where((product) {
39                     /// 키워드가 비어있는 경우 모두 반환
40                     if (keyword.isEmpty) return true;
41
42                     /// name이나 brand에 키워드 포함 여부 확인
43                     return "${product.name}${product.brand}"
44                         .toLowerCase()
45                         .contains(keyword.toLowerCase());
46                 }).toList();
47             });
48         } catch (e, s) {
49             log('Failed to searchProductList', error: e, stackTrace: s);
50         }
51     }
52
53     @override
54     void initState() {
55         super.initState();
56         searchProductList(); 주석처리
57     }
58
59     @override
60     Widget build(BuildContext context) {
61 >     return HideKeyboard( // HideKeyboard ...
124 }
125 }

```

ShoppingViewModel로 이동

▼ 코드스니펫 - `shopping_view_model.dart` : ShoppingView 로직 이전

```
import 'dart:convert';
import 'dart:developer';

import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/model/product.dart';
import 'package:house_of_tomorrow/src/view/base_view_model.dart';
import 'package:house_of_tomorrow/util/helper/network_helper.dart';

class ShoppingViewModel extends BaseViewModel {
  List<Product> productList = [];
  final TextEditingController textController = TextEditingCont

  String get keyword => textController.text.trim();

  Future<void> searchProductList() async {
    try {
      final res = await NetworkHelper.dio.get(
        'https://gist.githubusercontent.com/nero-angela/d16a50'
      );

      productList = jsonDecode(res.data).map<Product>((json) {
        return Product.fromJson(json);
      }).where((product) {
        /// 키워드가 비어있는 경우 모두 반환
        if (keyword.isEmpty) return true;

        /// name이나 brand에 키워드 포함 여부 확인
        return "${product.name}${product.brand}"
          .toLowerCase()
          .contains(keyword.toLowerCase());
      }).toList();
      notifyListeners();
    } catch (e, s) {
      log('Failed to fetchProductList', error: e, stackTrace: s);
    }
  }
}
```

```

1  import 'dart:convert';
2  import 'dart:developer';
3
4  import 'package:flutter/material.dart';
5  import 'package:house_of_tomorrow/src/model/product.dart';
6  import 'package:house_of_tomorrow/src/view/base_view_model.dart';
7  import 'package:house_of_tomorrow/util/helper/network_helper.dart';
8
9  class ShoppingViewModel extends BaseViewModel {
10     List<Product> productList = [];
11     final TextEditingController textController = TextEditingController();
12
13     String get keyword => textController.text.trim();
14
15     Future<void> searchProductList() async {
16         try {
17             final res = await NetworkHelper.dio.get(
18                 'https://gist.githubusercontent.com/nero-angela/d16a5078c7959bf5ab
19             );
20
21             productList = jsonDecode(res.data).map<Product>((json) {
22                 return Product.fromJson(json);
23             }).where((product) {
24                 /// 키워드가 비어있는 경우 모두 반환
25                 if (keyword.isEmpty) return true;
26
27                 /// name이나 brand에 키워드 포함 여부 확인
28                 return "${product.name}${product.brand}"
29                     .toLowerCase()
30                     .contains(keyword.toLowerCase());
31             }).toList();
32             notifyListeners();
33         } catch (e, s) {
34             log('Failed to fetchProductList', error: e, stackTrace: s);
35         }
36     }
37 }

```



`searchProductList()` 메소드의 `setState()` 는 제거한 뒤 `notifyListeners();` 로 변경하였습니다.



`shopping_view.dart` 파일은 다음과 같이 남게 됩니다.

```
20 class _ShoppingViewState extends State<ShoppingView> {
21   @override
22   void initState() {
23     super.initState();
24     // searchProductList();
25   }
26
27   @override
28   Widget build(BuildContext context) {
```

4. `_ShoppingViewState` 의 기존 코드 `ShoppingViewModel` 로 옮겼기 때문에 `build()` 메소드에서 에러가 발생합니다. 에러 앞에 `viewModel.` 을 추가하여 ViewModel의 값을 바라보도록 수정해 주세요.

▼ 코드스니펫 - `shopping_view.dart` : 최종

```
import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/view/base_view.dart';
import 'package:house_of_tomorrow/src/view/shopping/shopping_view.dart';
import 'package:house_of_tomorrow/src/view/shopping/widget/product_list.dart';
import 'package:house_of_tomorrow/src/view/shopping/widget/product_detail.dart';
import 'package:house_of_tomorrow/theme/component/bottom_sheet.dart';
import 'package:house_of_tomorrow/theme/component/button/button.dart';
import 'package:house_of_tomorrow/theme/component/cart_button.dart';
import 'package:house_of_tomorrow/theme/component/hide_keyboard.dart';
import 'package:house_of_tomorrow/theme/component/input_field.dart';
import 'package:house_of_tomorrow/util/lang/generated/l10n.dart';

class ShoppingView extends StatefulWidget {
  const ShoppingView({super.key});

  @override
  State<ShoppingView> createState() => _ShoppingViewState();
}

class _ShoppingViewState extends State<ShoppingView> {
  @override
  void initState() {
    super.initState();
```



```

    // searchProductList();
  }

  @override
  Widget build(BuildContext context) {
    return BaseView(
      viewModel: ShoppingViewModel(),
      builder: (context, viewModel) => HideKeyboard(
        child: Scaffold(
          appBar: AppBar(
            title: Text(S.current.shopping),
            actions: [
              /// 설정 버튼
              Button(
                icon: 'option',
                type: ButtonType.flat,
                onPressed: () {
                  showModalBottomSheet(
                    context: context,
                    isScrollControlled: true,
                    builder: (context) {
                      return const SettingBottomSheet();
                    },
                  );
                },
              ),
            ],
          ),
          body: Column(
            children: [
              Padding(
                padding: const EdgeInsets.symmetric(
                  horizontal: 16,
                  vertical: 8,
                ),
                child: Row(
                  children: [

```

```

        /// 검색
        Expanded(
          child: InputField(
            controller: viewModel.textController,
            onClear: viewModel.searchProductList,
            onSubmitted: (text) => viewModel.searchProductList,
            hint: S.current.searchProduct,
          ),
        ),
        const SizedBox(width: 16),

        /// 검색 버튼
        Button(
          icon: 'search',
          onPressed: viewModel.searchProductList,
        ),
      ],
    ),
  ),
),

/// ProductCardList
Expanded(
  child: viewModel.productList.isEmpty
    ? const ProductEmpty()
    : ProductCardGrid(viewModel.productList),
),
],
),
),
),
);
}
}

```

```

64      /// 검색
65      Expanded(
66        child: InputField(
67+          controller: viewModel.textController,
68+          onClear: viewModel.searchProductList,
69+          onSubmitted: (text) => viewModel.searchProductList(),
70          hint: S.current.searchProduct,
71        ),
72      ),
73      const SizedBox(width: 16),
74
75      /// 검색 버튼
76      Button(
77        icon: 'search',
78+        onPressed: viewModel.searchProductList,
79      ),
80    ],
81  ),
82 ),
83
84  /// ProductCardList
85  Expanded(
86+    child: viewModel.productList.isEmpty
87    ? const ProductEmpty()
88+    : ProductCardGrid(viewModel.productList),
89  ),
90 ],

```

5. `StatefulWidget` 의 `initState()` 트리거를 사용할 수 있도록 아래와 같이 수정해 주세요.

▼ 코드스니펫 - `shopping_view.dart` : `initState()` 트리거 사용

```

final ShoppingViewModel shoppingViewModel = ShoppingViewModel(

shoppingViewModel.searchProductList();

viewModel: shoppingViewModel,

```

```

20 class _ShoppingViewState extends State<ShoppingView> {
21+   final ShoppingViewModel shoppingViewModel = ShoppingViewModel();
22+
23   @override
24   void initState() {
25     super.initState();
26+   shoppingViewModel.searchProductList();
27   }
28
29   @override
30   Widget build(BuildContext context) {
31     return BaseView(
32+     viewModel: shoppingViewModel,
33     builder: (context, viewModel) => HideKeyboard(
34     child: Scaffold(

```

7. Restart 버튼을 누르면 정상적으로 화면이 출력 됩니다.



에뮬레이터에서 테스트시 기존과 동일하게 작동합니다.

▼ ProductRepository



`shopping_view_model.dart` 에서 `searchProductList()` 메소드는 데이터 요청 로직과 View 로직을 모두 가지고 있습니다. 데이터 요청 로직은 MVVM 패턴에서 Model에 해당하므로 분리해 봅시다.

```
Future<void> searchProductList() async {
  try {
    final res = await NetworkHelper.dio.get(
      'https://gist.githubusercontent.com/nero-angela/d16a5078c7959bf5abf6a9e...');

    productList = jsonDecode(res.data).map<Product>((json) {
      return Product.fromJson(json);
    }).where((product) {
      /// 키워드가 비어있는 경우 모두 반환
      if (keyword.isEmpty) return true;

      /// name이나 brand에 키워드 포함 여부 확인
      return "${product.name}${product.brand}"
        .toLowerCase()
        .contains(keyword.toLowerCase());
    }).toList();
    notifyListeners();
  } catch (e, s) {
    log('Failed to fetchProductList', error: e, stackTrace: s);
  }
}
```



데이터 요청 로직을 별도로 분리하면, 로직을 재활용할 수 있습니다.

1. `lib/src` 폴더 밑에 `repository` 폴더를 만들고, 밑에 `product_repository.dart` 파일을 다음과 같이 생성해 주세요.

▼ 코드스니펫 - `product_repository.dart`

```
import 'dart:convert';
import 'dart:developer';

import 'package:house_of_tomorrow/src/model/product.dart';
import 'package:house_of_tomorrow/util/helper/network_helper.dart';

class ProductRepository {
```

```

Future<List<Product>> searchProductList(String keyword) async
try {
  final res = await NetworkHelper.dio.get(
    'https://gist.githubusercontent.com/nero-angela/d16a50'
  );

  return jsonDecode(res.data).map<Product>((json) {
    return Product.fromJson(json);
  }).where((product) {
    /// 키워드가 비어있는 경우 모두 반환
    if (keyword.isEmpty) return true;

    /// name이나 brand에 키워드 포함 여부 확인
    return "${product.name}${product.brand}"
      .toLowerCase()
      .contains(keyword.toLowerCase());
  }).toList();
} catch (e, s) {
  log('Failed to searchProductList', error: e, stackTrace:
  return []);
}
}
}

```

```

HOUSE_OF_TOMORROW
├── .dart_tool
├── .idea
├── android
├── assets
├── build
├── ios
├── lib
│   ├── src
│   │   ├── model
│   │   ├── repository
│   │   │   └── product_repository.dart
│   │   ├── service
│   │   ├── view
│   │   ├── theme
│   │   ├── util
│   │   └── main.dart
│   ├── linux
│   ├── macos
│   ├── test
│   ├── web
│   ├── windows
│   ├── .flutter-plugins
│   ├── .flutter-plugins-dependencies
│   ├── .gitignore
│   ├── .metadata
│   ├── ! analysis_options.yaml
│   ├── house_of_tomorrow.iml
│   ├── pubspec.lock
│   └── pubspec.yaml
├── .gitignore
├── .metadata
├── ! analysis_options.yaml
├── house_of_tomorrow.iml
├── pubspec.lock
└── pubspec.yaml

lib > src > repository > product_repository.dart > ...
1  import 'dart:convert';
2  import 'dart:developer';
3
4  import 'package:house_of_tomorrow/src/model/product.dart';
5  import 'package:house_of_tomorrow/util/helper/network_helper.dart';
6
7  class ProductRepository {
8      Future<List<Product>> searchProductList(String keyword) async {
9          try {
10             final res = await NetworkHelper.dio.get(
11                 'https://gist.githubusercontent.com/nero-angela/d16a5078c7959bf5ab
12             );
13
14             return jsonDecode(res.data).map<Product>((json) {
15                 return Product.fromJson(json);
16             }).where((product) {
17                 /// 키워드가 비어있는 경우 모두 반환
18                 if (keyword.isEmpty) return true;
19
20                 /// name이나 brand에 키워드 포함 여부 확인
21                 return "${product.name}${product.brand}"
22                     .toLowerCase()
23                     .contains(keyword.toLowerCase());
24             }).toList();
25         } catch (e, s) {
26             log('Failed to searchProductList', error: e, stackTrace: s);
27             return [];
28         }
29     }
30 }

```



데이터 요청하는 로직만 별도로 분리하여 키워드를 전달받고 `Future<List<Product>>` 를 반환하는 `searchProductList()` 를 구현하였습니다.

2. `shopping_view_model.dart` 파일에 `productRepository` 속성을 만들고 `searchProductList()` 메소드를 수정해 주세요.

▼ 코드스니펫 - `shopping_view_model.dart` : ProductRepository 추가

```

import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/model/product.dart';
import 'package:house_of_tomorrow/src/repository/product_repository.dart';
import 'package:house_of_tomorrow/src/view/base_view_model.dart';

class ShoppingViewModel extends BaseViewModel {
    List<Product> productList = [];
    final TextEditingController textController = TextEditingController();
    final ProductRepository productRepository = ProductRepository();

    String get keyword => textController.text.trim();

```

```

Future<void> searchProductList() async {
  productList = await productRepository.searchProductList(keyword);
  notifyListeners();
}
}

```

```

1  import 'package:flutter/material.dart';
2  import 'package:house_of_tomorrow/src/model/product.dart';
3  import 'package:house_of_tomorrow/src/repository/product_repository.dart';
4  import 'package:house_of_tomorrow/src/view/base_view_model.dart';
5
6  class ShoppingViewModel extends BaseViewModel {
7    List<Product> productList = [];
8    final TextEditingController textController = TextEditingController();
9    final ProductRepository productRepository = ProductRepository();
10
11    String get keyword => textController.text.trim();
12
13    Future<void> searchProductList() async {
14      productList = await productRepository.searchProductList(keyword);
15      notifyListeners();
16    }
17  }

```



Restart한 뒤 테스트를 해보면 정상적으로 작동하는 것을 확인할 수 있습니다.

▼ CircularProgressIndicator



`shopping_view_model.dart` 파일에서 `searchProductList()` 함수는 네트워크 통신을 합니다. 네트워크 통신은 시간이 오래 걸릴 수 있기 때문에 `BaseViewModel`에 추가한 `isBusy` 속성을 이용해 다음과 같이 로딩 애니메이션을 추가해 봅시다.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d8c9d645-b66d-43de-b694-55f990ed6008/%E1%84%92%E1%85%AA%E1%84%86%E1%85%A7%E1%86%AB_%E1%84%80%E1%85%B5%E1%84%85%E1%85%A9%E1%86%A8_2023-02-23_%E1%84%8B%E1%85%A9%E1%84%92%E1%85%AE_12.55.49.mov

1. `theme/component` 폴더 밑에 `circular_indicator.dart` 파일을 추가해 주세요.

▼ 코드스니펫 - `circular_indicator.dart` : 최종 코드

```
import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/service/theme_service.dart';

class CircularIndicator extends StatelessWidget {
  const CircularIndicator({
    super.key,
    required this.child,
    required this.isBusy,
  });

  final Widget child;
  final bool isBusy;

  @override
  Widget build(BuildContext context) {
    return Stack(
      children: [
        child,

        /// CircularIndicator
        IgnorePointer(
          ignoring: !isBusy,
          child: AnimatedOpacity(
            duration: const Duration(milliseconds: 222),
            opacity: isBusy ? 1 : 0,
            child: Container(
              color: context.color.background,
              alignment: Alignment.center,
              child: CircularProgressIndicator(
                color: context.color.primary,
                value: isBusy ? null : 0,
              ),
            ),
          ),
        ),
      ],
    );
  }
}
```

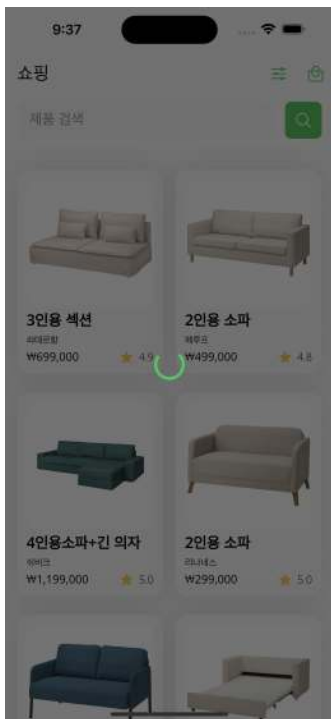
```
}
}
```

```

HOUSE_OF_TOMORROW
  > .dart_tool
  > .idea
  > android
  > assets
  > build
  > ios
  > lib
    > src
    > theme
      > component
        > bottom_sheet
        > button
        > toast
        > asset_icon.dart
        > base_dialog.dart
        > cart_button.dart
        > circular_indicator.dart
        > color_picker.dart
        > constrained_screen.dart
        > counter_badge.dart
        > counter_button.dart
        > hide_keyboard.dart
        > input_field.dart
        > pop_button.dart
        > rating.dart
        > tile.dart
      > foundation
      > res
      > dark_theme.dart
      > light_theme.dart
      > util
      > main.dart
    > linux
    > macos
    > test
  > OUTLINE
  > TIMELINE

lib > theme > component > circular_indicator.dart > ...
1  import 'package:flutter/material.dart';
2  import 'package:house_of_tomorrow/src/service/theme_service.dart';
3
4  class CircularIndicator extends StatelessWidget {
5    const CircularIndicator({
6      super.key,
7      required this.child,
8      required this.isBusy,
9    });
10
11    final Widget child;
12    final bool isBusy;
13
14    @override
15    Widget build(BuildContext context) {
16      return Stack(
17        children: [
18          child,
19
20          /// CircularIndicator
21          IgnorePointer(
22            ignoring: !isBusy,
23            child: AnimatedOpacity(
24              duration: const Duration(milliseconds: 222),
25              opacity: isBusy ? 1 : 0,
26              child: Container(
27                color: context.color.background,
28                alignment: Alignment.center,
29                child: CircularProgressIndicator(
30                  color: context.color.primary,
31                  value: isBusy ? null : 0,
32                ), // CircularProgressIndicator
33              ), // Container
34            ), // AnimatedOpacity
35          ), // IgnorePointer
36        ],
37      ); // Stack
38    }
39  }

```



`child` 위에 `Stack` 으로 배경(`Container`)와 로딩 (Material 소속 `CircularProgressIndicator`)을 배치하였습니다.



`isBusy` 의 상태에 따라 투명도를 조절하였고, 투명도 변경을 부드럽게 적용하기 위해 `AnimatedOpacity` 위젯을 활용하였습니다. 또한 투명한 경우 클릭하지 못하도록 `IgnorePointer` 위젯을 사용하였습니다.



`CircularProgressIndicator` 는 `value` 가 `null` 인 경우 애니메이션이 작동하며, `isBusy = false` 인 경우에는 애니메이션이 작동하지 않도록 0을 주었습니다.

2. `base_view.dart` 에서 `builder` 함수를 `CircularIndicator` 위젯으로 감싸고, `viewModel` 의 `isBusy` 속성을 전달해 주세요.

▼ 코드스니펫 - `base_view.dart` : `CircularIndicator` 추가

```
return CircularProgressIndicator(
  isBusy: viewModel.isBusy,
  child: builder(context, viewModel),
);
```

```

2+ import 'package:house_of_tomorrow/src/view/base_view_model.dart';
3+ import 'package:house_of_tomorrow/theme/component/circular_indicator.dart';
4 import 'package:provider/provider.dart';
5
6 class BaseView<T extends BaseViewModel> extends StatelessWidget {
7   const BaseView({
8     super.key,
9     required this.viewModel,
10    required this.builder,
11  });
12
13   final T viewModel;
14   final Widget Function(BuildContext context, T viewModel) builder;
15
16   @override
17   Widget build(BuildContext context) {
18     return ChangeNotifierProvider(
19       create: (context) => viewModel,
20       child: Consumer<T>({
21         builder: (context, viewModel, child) {
22+          return CircularIndicator(
23+            isBusy: viewModel.isBusy,
24+            child: builder(context, viewModel),
25+          );
26        },
27      ),

```



`isBusy` 상태에 따라 로딩 애니메이션을 그릴 준비가 완료되었습니다.

3. `shopping_view_model.dart` 파일에 `searchProductList()` 함수에 통신 코드 앞 뒤에 `isBusy` 를 변경하는 코드를 추가해 주세요.

▼ 코드스니펫 - `shopping_view_model.dart` : `isBusy` 추가

```

Future<void> searchProductList() async {
  isBusy = true;
  productList = await productRepository.searchProductList(keyw
  isBusy = false;
}

```

```

12
13 Future<void> searchProductList() async {
14+   isBusy = true;
15   productList = await productRepository.searchProductList(keyword);
16+   isBusy = false;
17 }
18 }

```



isBusy의 값이 변경될 때 `notifyListeners();`가 호출되기 때문에 기존에 있던 `notifyListeners();`를 삭제하였습니다.

```

class BaseViewModel with ChangeNotifier {
  bool _isBusy = false;

  bool get isBusy => _isBusy;

  set isBusy(bool isBusy) {
    if (_isBusy == isBusy) return;
    _isBusy = isBusy;
    notifyListeners();
  }
}

```



아래와 같이 `notifyListeners()` 를 중복하여 여러번 호출해도, 화면 갱신 횟수는 자동으로 최적화 됩니다.

```
Future<void> searchProductList() async {
  isBusy = true;
  productList = await productRepository.searchProductList(keyword);
  isBusy = false;
  notifyListeners();
  notifyListeners();
  notifyListeners();
  notifyListeners();
}
```

```
29   @override
30   Widget build(BuildContext context) {
31     return BaseView(
32       viewModel: shoppingViewModel,
33       builder: (context, viewModel) {
34         print("notifyListeners 호출됨");
35         return HideKeyboard(
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !exclu... Dart

```
flutter: notifyListeners 호출됨
[log] REQ : [GET] https://gist.githubusercontent.com/nero-angela/d16a5078c7959bf5abf6a9e0d21ddd1ba06f0349a890f5e5347d94d677e/ikeaSofaDataIBB.json
[log] RES : [200] https://gist.githubusercontent.com/nero-angela/d16a5078c7959bf5abf6a9e0d21ddd1ba06f0349a890f5e5347d94d677e/ikeaSofaDataIBB.json
flutter: notifyListeners 호출됨
```



테스트 시 네트워크 통신 응답이 빠른 경우 로딩 애니메이션이 나타났는지 제대로 인지하지 못한 상태로 종료됩니다.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/dd68cd09-cee4-46dd-a054-4ef2fee42a56/%E1%84%92%E1%85%AA%E1%84%86%E1%85%A7%E1%86%AB_%E1%84%80%E1%85%B5%E1%84%85%E1%85%A9%E1%86%A8_2023-02-23_%E1%84%8B%E1%85%A9%E1%84%92%E1%85%AE_1.07.15.mov