



로딩 애니메이션을 너무 짧게 보여주면 유저 입장에서 오히려 무언가를 놓쳤다는 경험을 주어 좋지 않을 수 있습니다. 따라서 로딩 애니메이션의 최소 실행 시간을 보장해 주도록 개선해 봅시다.

4. `shopping_view_model.dart` 파일에 `searchProductList()` 메소드를 다음과 같이 수정해 주세요.

▼ 코드스니펫 - `shopping_view_model.dart` : `Future.wait` 추가

```
Future<void> searchProductList() async {
  isBusy = true;
  final results = await Future.wait([
    productRepository.searchProductList(keyword),
    Future.delayed(const Duration(milliseconds: 555)),
  ]);
  productList = results[0];
  isBusy = false;
}
```


```
Future<void> searchProductList() async {
  isBusy = true;
  final results = await Future.wait([
    productRepository.searchProductList(keyword),
    Future.delayed(const Duration(milliseconds: 555)),
  ]);
  productList = results[0];
  isBusy = false;
}
```

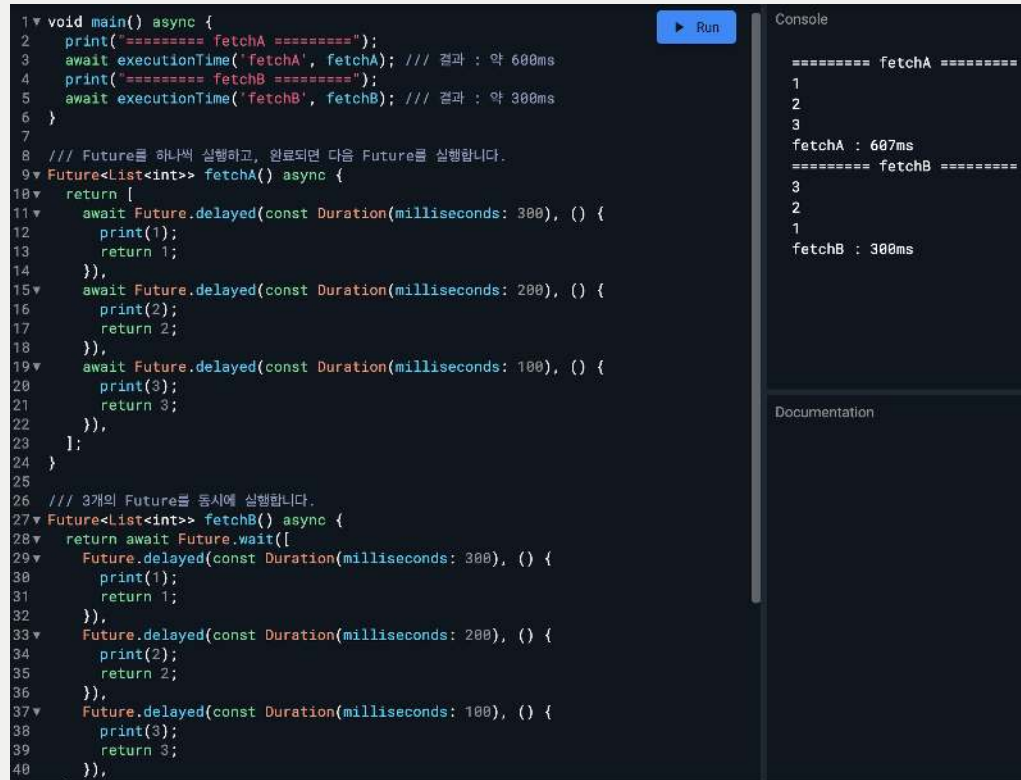


`Future.wait([Future, Future])` 은 여러 Future들을 동시에 실행한 뒤, 모든 Future가 종료된 후 결과를 배열로 반환해 줍니다.

위에선 실행 최소 시간을 555ms로 보장하기 위해 `Future.wait` 을 사용했지만, 여러 비동기 코드를 동시에 병렬 실행하여 시간을 단축할 수 있는 효과도 있습니다. 아래 Dart 를 참고해 주세요.

DartPad

 <https://dartpad.dev/?id=8fcb84ceccb3f3ce6cfad06c05620e81>



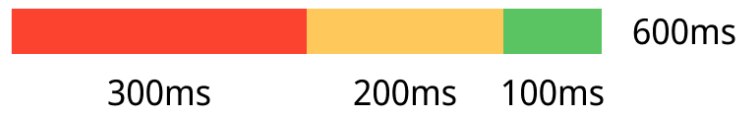
```
1 void main() async {
2   print("===== fetchA =====");
3   await executionTime('fetchA', fetchA); /// 결과 : 약 600ms
4   print("===== fetchB =====");
5   await executionTime('fetchB', fetchB); /// 결과 : 약 300ms
6 }
7
8 /// Future를 하나씩 실행하고, 완료되면 다음 Future를 실행합니다.
9 Future<List<int>> fetchA() async {
10  return [
11    await Future.delayed(const Duration(milliseconds: 300), () {
12      print(1);
13      return 1;
14    }),
15    await Future.delayed(const Duration(milliseconds: 200), () {
16      print(2);
17      return 2;
18    }),
19    await Future.delayed(const Duration(milliseconds: 100), () {
20      print(3);
21      return 3;
22    }),
23  ];
24 }
25
26 /// 3개의 Future를 동시에 실행합니다.
27 Future<List<int>> fetchB() async {
28  return await Future.wait([
29    Future.delayed(const Duration(milliseconds: 300), () {
30      print(1);
31      return 1;
32    }),
33    Future.delayed(const Duration(milliseconds: 200), () {
34      print(2);
35      return 2;
36    }),
37    Future.delayed(const Duration(milliseconds: 100), () {
38      print(3);
39      return 3;
40    }),
41  ]);
42 }
```

Console

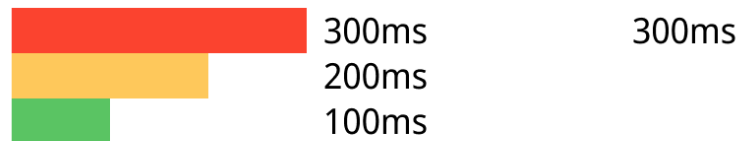
```
===== fetchA =====
1
2
3
fetchA : 607ms
===== fetchB =====
3
2
1
fetchB : 300ms
```

Documentation

fetchA



fetchB



`Future.wait()` 에 대한 자세한 내용은 아래 공식 문서를 참고해 주세요.

wait method - Future class - dart:async library - Dart API

Waits for multiple futures to complete and collects their results. Returns a future which will complete once all the provided futures have completed, either with their results, or with an error if any of the provided futures fail.

<https://api.flutter.dev/flutter/dart-async/Future/wait.html>



`Future.wait` 을 추가한 뒤 실행해보면 로딩 시간이 최소 555ms를 보장하여 작동합니다.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/0165ad05-3058-4a39-9129-3025b605b124/%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_1.12.25.mov

▼ ProductViewModel



`product_view.dart` 파일을 보면 View State와 View Logic 코드가 View와 함께 존재합니다.

```

29 class _ProductViewState extends State<ProductView> {
30     /// 선택한 수량
31     int count = 1;
32
33     /// 선택한 색상
34     int colorIndex = 0;
35
36     /// 수량 업데이트 이벤트 함수
37     void onCountChanged(int newCount) {
38         setState(() {
39             count = newCount;
40         });
41     }
42
43     /// 색상 업데이트 이벤트 함수
44     void onColorIndexChanged(int newColorIndex) {
45         setState(() {
46             colorIndex = newColorIndex;
47         });
48     }
49
50     /// 카트에 상품 추가
51     void onAddToCartPressed() {
52         final CartService cartService = context.read();
53         final CartItem newCartItem = CartItem(
54             colorIndex: colorIndex,
55             count: count,
56             isSelected: true,
57             product: widget.product,
58         );
59         cartService.add(newCartItem);
60         Toast.show(S.current.productAdded(widget.product.name));
61     }
62
63     @override
64     Widget build(BuildContext context) {

```

View State (Lines 30-34)

View Logic (Lines 36-61)



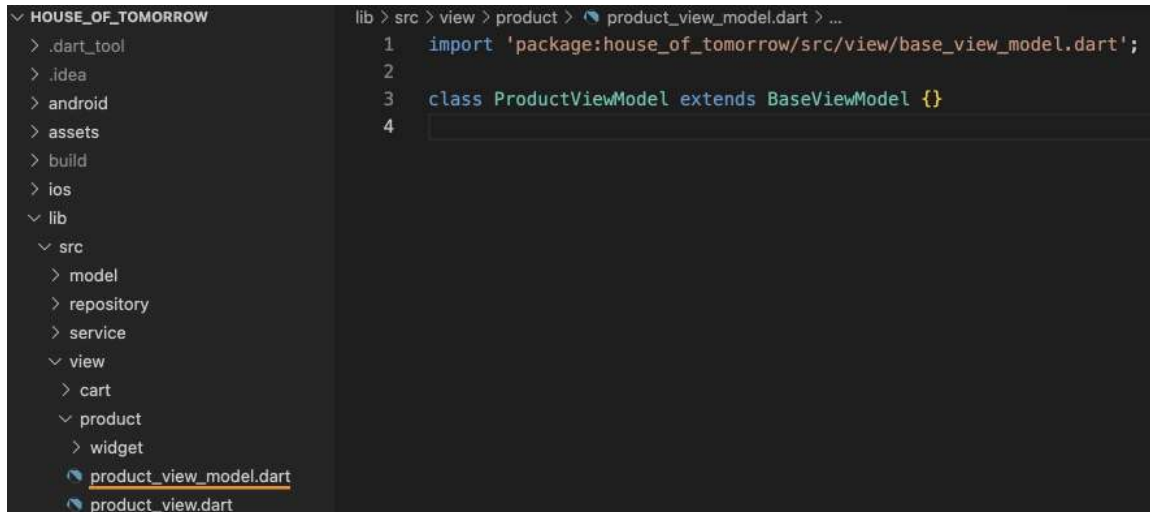
`ProductView`의 ViewModel 역할을 수행할 `ProductViewModel` 을 만들어 관심사 분리를 진행해 봅시다.

1. `view/product` 폴더 밑에 `product_view_model.dart` 파일을 생성해 주세요.

▼ 코드스니펫 - `product_view_model.dart` : 시작

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

class ProductViewModel extends BaseViewModel {}
```



2. `product_view.dart` 파일에 `BaseView` 를 사용하여 `ProductViewModel` 을 추가해 주세요.

▼ 코드스니펫 - `product_view.dart` : `BaseView` 추가

```
return BaseView(
  viewModel: ProductViewModel(),
  builder: (context, viewModel) => Scaffold(
    appBar: AppBar(
      title: Text(S.current.product),
      leading: const PopButton(),
      titleSpacing: 0,
      actions: const [
        /// 카트 버튼
        CartButton(),
      ],
    ),
    body: ProductLayout(
      productInfo: SingleChildScrollView(
        padding: const EdgeInsets.symmetric(vertical: 32),
        child: Wrap(
          runSpacing: 32,
          alignment: WrapAlignment.center,
          children: [
```

```

        /// ProductColorPreview
        ProductColorPreview(
          colorIndex: colorIndex,
          product: widget.product,
        ),

        /// ColorPicker
        ColorPicker(
          colorIndex: colorIndex,
          colorList: widget.product.productColorList.map((
            return e.color;
          )).toList(),
          onColorSelected: onColorIndexChanged,
        ),

        /// ProductDesc
        ProductDesc(product: widget.product),
      ],
    ),
  ),
);

/// ProductBottomSheet
productBottomSheet: ProductBottomSheet(
  count: count,
  product: widget.product,
  onCountChanged: onCountChanged,
  onAddToCartPressed: onAddToCartPressed,
),
),
);

```

```

65   @override
66   Widget build(BuildContext context) {
67+     return BaseView(
68+       viewModel: ProductViewModel(),
69+       builder: (context, viewModel) => Scaffold(
70+         appBar: AppBar(

```

3. 아래 표시한 View State와 View Logic을 `ProductViewModel`로 옮겨주세요.

```

29 class _ProductViewState extends State<ProductView> {
30     /// 선택한 수량
31     int count = 1;
32
33     /// 선택한 색상
34     int colorIndex = 0;
35
36     /// 수량 업데이트 이벤트 함수
37     void onCountChanged(int newCount) {
38         setState(() {
39             count = newCount;
40         });
41     }
42
43     /// 색상 업데이트 이벤트 함수
44     void onColorIndexChanged(int newColorIndex) {
45         setState(() {
46             colorIndex = newColorIndex;
47         });
48     }
49
50     /// 카트에 상품 추가
51     void onAddToCartPressed() {
52         final CartService cartService = context.read();
53         final CartItem newCartItem = CartItem(
54             colorIndex: colorIndex,
55             count: count,
56             isSelected: true,
57             product: widget.product,
58         );
59         cartService.add(newCartItem);
60         Toast.show(S.current.productAdded(widget.product.name));
61     }
62
63     @override
64     Widget build(BuildContext context) {

```

▼ 코드스니펫 - `product_view_model.dart` : 최종

```

import 'package:house_of_tomorrow/src/model/cart_item.dart';
import 'package:house_of_tomorrow/src/model/product.dart';
import 'package:house_of_tomorrow/src/service/cart_service.dart';
import 'package:house_of_tomorrow/src/view/base_view_model.dart';
import 'package:house_of_tomorrow/theme/component/toast/toast.dart';
import 'package:house_of_tomorrow/util/lang/generated/l10n.dart';

class ProductViewModel extends BaseViewModel {
  ProductViewModel({

```

```

        required this.cartService,
        required this.product,
    });

    final CartService cartService;

    /// 선택한 상품
    final Product product;

    /// 선택한 수량
    int count = 1;

    /// 선택한 색상
    int colorIndex = 0;

    /// 수량 업데이트 이벤트 함수
    void onCountChanged(int newCount) {
        count = newCount;
        notifyListeners();
    }

    /// 색상 업데이트 이벤트 함수
    void onColorIndexChanged(int newColorIndex) {
        colorIndex = newColorIndex;
        notifyListeners();
    }

    /// 카트에 상품 추가
    void onAddToCartPressed() {
        final CartItem newCartItem = CartItem(
            colorIndex: colorIndex,
            count: count,
            isSelected: true,
            product: product,
        );
        cartService.add(newCartItem);
        Toast.show(S.current.productAdded(product.name));
    }
}

```



```

8  class ProductViewModel extends BaseViewModel {
9      ProductViewModel({
10         required this.cartService,
11         required this.product,
12     });
13
14     final CartService cartService;
15
16     /// 선택한 상품
17     final Product product;
18
19     /// 선택한 수량
20     int count = 1;
21
22     /// 선택한 색상
23     int colorIndex = 0;
24
25     /// 수량 업데이트 이벤트 함수
26     void onCountChanged(int newCount) {
27         count = newCount;
28         notifyListeners();
29     }
30
31     /// 색상 업데이트 이벤트 함수
32     void onColorIndexChanged(int newColorIndex) {
33         colorIndex = newColorIndex;
34         notifyListeners();
35     }
36
37     /// 카트에 상품 추가
38     void onAddToCartPressed() {
39         final CartItem newCartItem = CartItem(
40             colorIndex: colorIndex,
41             count: count,
42             isSelected: true,
43             product: product,
44         );
45         cartService.add(newCartItem);
46         Toast.show(S.current.productAdded(product.name));
47     }
48 }

```



`setState()` 를 `notifyListeners()` 로 변경하였습니다.



`ProductViewModel` 의 생성자에 `Product` 와 `CartService` 를 전달 받도록 구현한 뒤 `onAddToCartPressed()` 함수를 수정해 주었습니다.



`BuildContext` 는 어떤 위젯에서 가져오느냐에 따라 다른 정보가 들어있기 때문에 `ViewModel` 에서 다루는 것은 좋지 않으므로, `CartService` 를 전달받도록 구현하였습니다.

4. `product_view.dart` 를 다음과 같이 수정해 주세요.

▼ 코드스니펫 - `product_view.dart` : 최종

```
import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/model/product.dart';
import 'package:house_of_tomorrow/src/view/base_view.dart';
import 'package:house_of_tomorrow/src/view/product/product_view.dart';
import 'package:house_of_tomorrow/src/view/product/widget/product_widget.dart';
import 'package:house_of_tomorrow/src/view/product/widget/product_widget.dart';
import 'package:house_of_tomorrow/src/view/product/widget/product_widget.dart';
import 'package:house_of_tomorrow/theme/component/cart_button.dart';
import 'package:house_of_tomorrow/theme/component/color_picker.dart';
import 'package:house_of_tomorrow/theme/component/pop_button.dart';
import 'package:house_of_tomorrow/util/lang/generated/l10n.dart';
import 'package:provider/provider.dart';

import 'widget/product_bottom_sheet.dart';

class ProductView extends StatelessWidget {
  const ProductView({
    super.key,
    required this.product,
  });

  final Product product;

  @override
  Widget build(BuildContext context) {
    return BaseView(
      viewModel: ProductViewModel(
        product: product,
        cartService: context.read(),
      ),
      builder: (context, viewModel) => Scaffold(
        appBar: AppBar(
```

```

        title: Text(S.current.product),
        leading: const PopButton(),
        titleSpacing: 0,
        actions: const [
          /// 카트 버튼
          CartButton(),
        ],
      ),
    body: ProductLayout(
      productInfo: SingleChildScrollView(
        padding: const EdgeInsets.symmetric(vertical: 32),
        child: Wrap(
          runSpacing: 32,
          alignment: WrapAlignment.center,
          children: [
            /// ProductColorPreview
            ProductColorPreview(
              colorIndex: viewModel.colorIndex,
              product: product,
            ),

            /// ColorPicker
            ColorPicker(
              colorIndex: viewModel.colorIndex,
              colorList: product.productColorList.map((e) {
                return e.color;
              }).toList(),
              onColorSelected: viewModel.onColorIndexChanged,
            ),

            /// ProductDesc
            ProductDesc(product: product),
          ],
        ),
      ),
    ),

    /// ProductBottomSheet
    productBottomSheet: ProductBottomSheet(
      count: viewModel.count,
      product: product,

```

```

        onCountChanged: viewModel.onCountChanged,
        onAddToCartPressed: viewModel.onAddToCartPressed,
      ),
    ),
  );
}
}

```



`ProductView` 를 `StatelessWidget` 으로 변경하였습니다.

```

class ProductView extends StatefulWidget {
  const ProductView({
    super.key,
    required this.product,
  });
}

```

More Actions...
💡 Convert to StatelessWidget



`ProductViewModel` 생성자에 `Product` 와 `CartService` 를 전달하였습니다.

```

25  Widget build(BuildContext context) {
26    return BaseView(
27+      viewModel: ProductViewModel(
28+        product: product,
29+        cartService: context.read(),
30+      ),
31    builder: (context, viewModel) => Scaffold(

```



기존 `ProductView`의 변수와 함수를 호출하는 코드 앞에 `viewModel.`을 추가하였습니다.

```
children: [
  /// ProductColorPreview
  ProductColorPreview(
    colorIndex: viewModel.colorIndex,
    product: product,
  ),

  /// ColorPicker
  ColorPicker(
    colorIndex: viewModel.colorIndex,
    colorList: product.productColorList.map((e) {
      return e.color;
    }).toList(),
    onColorSelected: viewModel.onColorIndexChanged,
  ),

  /// ProductDesc
  ProductDesc(product: product),
],
),
),

/// ProductBottomSheet
productBottomSheet: ProductBottomSheet(
  count: viewModel.count,
  product: product,
  onCountChanged: viewModel.onCountChanged,
  onAddToCartPressed: viewModel.onAddToCartPressed,
),
),
),
```

▼ CartViewModel

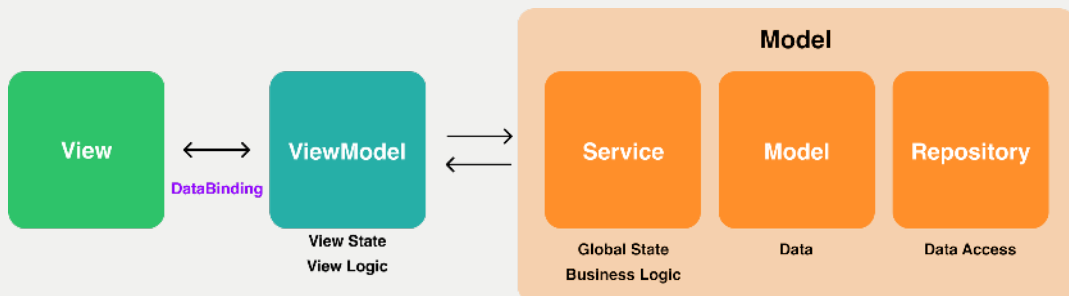


`CartView` 는 `CartService` 로 어느 정도 관심사 분리가 구현되어 있지만, 중간 중간에 View Logic도 섞여 있습니다.

```
/// CartBottomSheet
cartBottomSheet: CartBottomSheet(
  totalPrice: cartService.selectedCartItemList.isEmpty
    ? '0'
    : IntlHelper.currency(
        symbol:
          cartService.selectedCartItemList.first.product.priceUnit,
        number:
          cartService.selectedCartItemList.fold(0, (prev, curr) {
            return prev + curr.count * curr.product.price;
          }),
      ),
  selectedCartItemList: cartService.selectedCartItemList,
```



`View` 와 `Service` 사이에 `ViewModel` 을 추가하여 `View` 에서 `Service` 를 알지 못하도록 만들겠습니다.



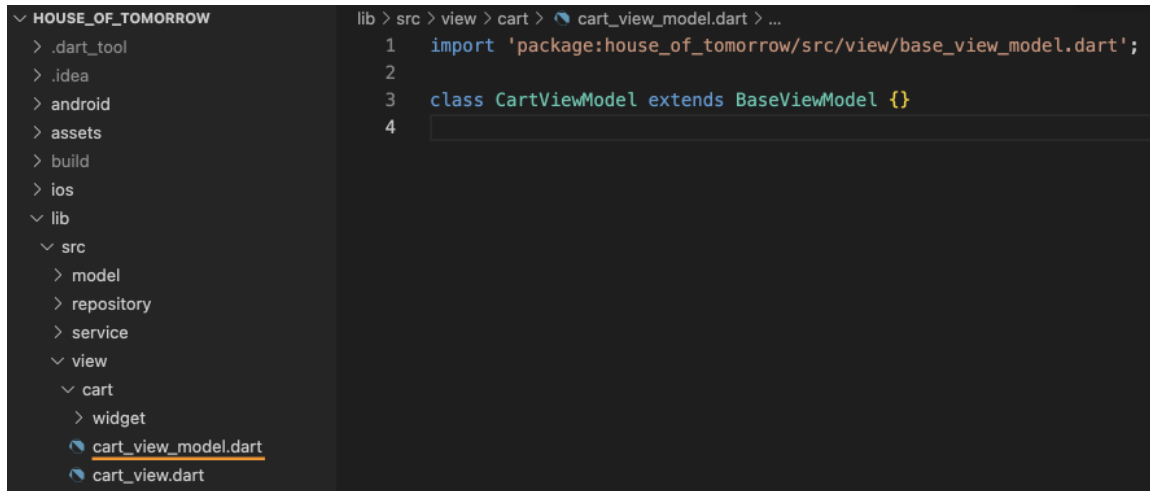
`CartView` 의 ViewModel 역할을 수행할 `CartViewModel` 을 만들어 관심사 분리를 진행해 봅시다.

1. `view/cart` 폴더 밑에 `cart_view_model.dart` 파일을 생성해 주세요.

▼ 코드스니펫 - `cart_view_model.dart` : 시작

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

```
class CartViewModel extends BaseViewModel {}
```



2. `cart_view.dart` 파일에 `BaseView` 를 이용하여 `CartViewModel` 을 추가해 주세요.

▼ 코드스니펫 - `cart_view.dart` : BaseView 추가 전체 코드

```
import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/service/cart_service.dart';
import 'package:house_of_tomorrow/src/service/theme_service.dart';
import 'package:house_of_tomorrow/src/view/base_view.dart';
import 'package:house_of_tomorrow/src/view/cart/cart_view_model.dart';
import 'package:house_of_tomorrow/src/view/cart/widget/cart_button.dart';
import 'package:house_of_tomorrow/src/view/cart/widget/cart_checkbox.dart';
import 'package:house_of_tomorrow/src/view/cart/widget/cart_delete.dart';
import 'package:house_of_tomorrow/src/view/cart/widget/cart_empty.dart';
import 'package:house_of_tomorrow/src/view/cart/widget/cart_item.dart';
import 'package:house_of_tomorrow/src/view/cart/widget/cart_layout.dart';
import 'package:house_of_tomorrow/theme/component/button/button.dart';
import 'package:house_of_tomorrow/theme/component/pop_button.dart';
import 'package:house_of_tomorrow/theme/component/toast/toast.dart';
import 'package:house_of_tomorrow/util/helper/intl_helper.dart';
import 'package:house_of_tomorrow/util/lang/generated/l10n.dart';
import 'package:provider/provider.dart';

class CartView extends StatelessWidget {
  const CartView({super.key});

  @override
```

```

Widget build(BuildContext context) {
  final CartService cartService = context.watch();
  return BaseView(
    viewModel: CartViewModel(),
    builder: (context, viewModel) => Scaffold(
      appBar: AppBar(
        title: Text(S.current.cart),
        leading: const PopButton(),
        titleSpacing: 0,
        actions: [
          /// Delete Button
          Button(
            onPressed: () {
              /// Show delete dialog
              showDialog(
                context: context,
                builder: (context) {
                  return CartDeleteDialog(
                    onDeletePressed: () {
                      cartService.delete(cartService.selectedItem);
                      Toast.show(S.current.deleteDialogSuccess);
                    },
                  );
                },
              );
            },
            text: S.current.delete,
            type: ButtonType.flat,
            color: context.color.secondary,
            isInactive: cartService.selectedCartItemList.isEmpty,
          ),
        ],
      ),
      body: CartLayout(
        /// CartItemList
        cartItemList: cartService.cartItemList.isEmpty
          ? const CartEmpty()
          : ListView.builder(
              itemCount: cartService.cartItemList.length,
              itemBuilder: (context, index) {

```



```

        final cartItem = cartService.cartItemList[index];
        return CartItemTile(
          index: index,
          cartItem: cartItem,
          onPressed: () {
            cartService.update(
              index,
              cartItem.copyWith(
                isSelected: !cartItem.isSelected,
              ),
            );
          },
          onCountChanged: (count) {
            cartService.update(
              index,
              cartItem.copyWith(
                count: count,
              ),
            );
          },
        );
      },
    );
  ),
),

```

/// CartBottomSheet

```

cartBottomSheet: CartBottomSheet(
  totalPrice: cartService.selectedCartItemList.isEmpty
    ? '0'
    : IntlHelper.currency(
      symbol: cartService
        .selectedCartItemList.first.product.pr
      number:
        cartService.selectedCartItemList.fold(0, (prev, curr) {
          return prev + curr.count * curr.product.pr
        }),
    ),
  selectedCartItemList: cartService.selectedCartItemList,
  onCheckoutPressed: () {
    /// Show checkout dialog
    showDialog(

```

```

        context: context,
        builder: (context) {
          return CartCheckoutDialog(
            onCheckoutPressed: () {
              cartService.delete(cartService.selectedCart);
              Toast.show(S.current.checkoutDialogSuccess);
            },
          );
        },
      ),
    ),
  ),
);
}
}

```

```

21
22   @override
23   Widget build(BuildContext context) {
24     final CartService cartService = context.watch();
25+    return BaseView(
26+      viewModel: CartViewModel(),
27+      builder: (context, viewModel) => Scaffold(
28        appBar: AppBar(
29          title: Text(S.current.cart),

```

3. CartView가 아닌 CartViewModel에서 CartService와 소통하도록 구현해 줍시다.

▼ 코드스니펫 - `cart_view_model.dart` : CartService 상호작용

```

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

class CartViewModel extends BaseViewModel {
  CartViewModel({
    required this.cartService,
  }) {
    cartService.addListener(notifyListeners);
  }
}

```

```

final CartService cartService;

@override
void dispose() {
  cartService.removeListener(notifyListeners);
  super.dispose();
}
}

```

```

1  import 'package:house_of_tomorrow/src/service/cart_service.dart';
2  import 'package:house_of_tomorrow/src/view/base_view_model.dart';
3
4  class CartViewModel extends BaseViewModel {
5    CartViewModel({
6      required this.cartService,
7    }) {
8      cartService.addListener(notifyListeners);
9    }
10
11    final CartService cartService;
12
13    @override
14    void dispose() {
15      cartService.removeListener(notifyListeners);
16      super.dispose();
17    }
18  }
19

```



CartView는 CartService에서 변경사항이 있을 때 함께 갱신되도록 만들기 위해 `addListener()` 로 CartViewModel의 `notifyListeners` 를 연결해 주었습니다.



CartViewModel이 더 이상 사용되지 않을 때, `removeListener` 를 호출하여 불필요한 알림이 발생하지 않도록 처리해 주었습니다.

4. `cart_view.dart` 파일에서 CartViewModel에 CartService를 전달해 주세요.

▼ 코드스니펫 - `cart_view.dart` : CartViewModel에 CartService 전달

```
cartService: context.read(),
```

```
22  @override
23  Widget build(BuildContext context) {
24    final CartService cartService = context.watch();
25    return BaseView(
26+      viewModel: CartViewModel(
27+        cartService: context.read(),
28+      ),
29    builder: (context, viewModel) => Scaffold(
30      appBar: AppBar(
```



위 이미지 기준 24번째 줄의 cartService는 제거할 예정이므로 `context.read()`를 이용해 전달하였습니다.

5. `cart_view_model.dart` 파일에 CartView의 상태와 로직을 구현해 주세요.

▼ 코드스니펫 - `cart_view_model.dart` : View State & Logic 구현 최종

```
import 'package:house_of_tomorrow/src/model/cart_item.dart';
import 'package:house_of_tomorrow/src/service/cart_service.dart';
import 'package:house_of_tomorrow/src/view/base_view_model.dart';
import 'package:house_of_tomorrow/theme/component/toast/toast.dart';
import 'package:house_of_tomorrow/util/helper/intl_helper.dart';
import 'package:house_of_tomorrow/util/lang/generated/l10n.dart';

class CartViewModel extends BaseViewModel {
  CartViewModel({
    required this.cartService,
  }) {
    cartService.addListener(notifyListeners);
  }

  final CartService cartService;

  @override
  void dispose() {
    cartService.removeListener(notifyListeners);
  }
}
```

```

    super.dispose();
}

/// 전체 CartItem
List<CartItem> get cartItemList => cartService.cartItemList;

/// 선택한 CartItem 목록
List<CartItem> get selectedCartItemList => cartService.selectedCartItemList;

/// 최종 가격
String get totalPrice {
    return selectedCartItemList.isEmpty
        ? '0'
        : IntlHelper.currency(
            symbol: cartService.selectedCartItemList.first.product.symbol,
            number: cartService.selectedCartItemList.fold(0, (prev, curr) {
                return prev + curr.count * curr.product.price;
            }),
        );
}

/// 선택한 CartItem 삭제
void onDeletePressed() {
    cartService.delete(cartService.selectedCartItemList);
    Toast.show(S.current.deleteDialogSuccessToast);
}

/// CartItem 클릭
void onCartItemPressed(int index) {
    final cartItem = cartItemList[index];
    cartService.update(
        index,
        cartItem.copyWith(
            isSelected: !cartItem.isSelected,
        ),
    );
}

/// CartItem 개수 변경
void onCountChanged(int index, int count) {

```

```

        cartService.update(
            index,
            cartItemList[index].copyWith(
                count: count,
            ),
        );
    }

    /// 선택한 CartItem 결제
    void onCheckoutPressed() {
        cartService.delete(cartService.selectedCartItemList);
        Toast.show(S.current.checkoutDialogSuccessToast);
    }
}

```



`CartView` 에서 필요한 상태를 Getter를 이용해 `CartService` 에서 가져오도록 구현하였습니다.

```

22
23    /// 전체 CartItem
24    List<CartItem> get cartItemList => cartService.cartItemList;
25
26    /// 선택한 CartItem 목록
27    List<CartItem> get selectedCartItemList => cartService.selectedCartItemList;
28
29    /// 최종 가격
30    String get totalPrice {
31        return selectedCartItemList.isEmpty
32            ? '0'
33            : IntlHelper.currency(
34                symbol: cartService.selectedCartItemList.first.product.priceUnit,
35                number: cartService.selectedCartItemList.fold(0, (prev, curr) {
36                    return prev + curr.count * curr.product.price;
37                }),
38            );
39    }
40

```



`CartView` 에서 발생하는 모든 이벤트를 처리해줄 View Logic을 다음과 같이 구현해 주었습니다. `CartService` 와 바인딩 되어 있으므로 함수에 `notifyListeners()` 를 작성하지 않았습니다.

```

40
41    /// 선택한 CartItem 삭제
42    void onDeletePressed() {
43        cartService.delete(cartService.selectedCartItem);
44        Toast.show(S.current.deleteDialogSuccessToast);
45    }
46
47    /// CartItem 클릭
48    void onCartItemPressed(int index) {
49        final cartItem = cartItemList[index];
50        cartService.update(
51            index,
52            cartItem.copyWith(
53                isSelected: !cartItem.isSelected,
54            ),
55        );
56    }
57
58    /// CartItem 개수 변경
59    void onCountChanged(int index, int count) {
60        cartService.update(
61            index,
62            cartItemList[index].copyWith(
63                count: count,
64            ),
65        );
66    }
67
68    /// 선택한 CartItem 결제
69    void onCheckoutPressed() {
70        cartService.delete(cartService.selectedCartItem);
71        Toast.show(S.current.checkoutDialogSuccessToast);
72    }

```

6. `cart_view.dart` 에서 `CartService` 관련 코드를 모두 `CartViewModel`로 변경해 주세요.

▼ 코드스니펫 - `cart_view.dart` : `CartService` 의존성 제거 최종

```

import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/service/theme_service.dart';
import 'package:house_of_tomorrow/src/view/base_view.dart';
import 'package:house_of_tomorrow/src/view/cart/cart_view_model.dart';

```

```

import 'package:house_of_tomorrow/src/view/cart/widget/cart_bo
import 'package:house_of_tomorrow/src/view/cart/widget/cart_ch
import 'package:house_of_tomorrow/src/view/cart/widget/cart_de
import 'package:house_of_tomorrow/src/view/cart/widget/cart_em
import 'package:house_of_tomorrow/src/view/cart/widget/cart_it
import 'package:house_of_tomorrow/src/view/cart/widget/cart_la
import 'package:house_of_tomorrow/theme/component/button/button
import 'package:house_of_tomorrow/theme/component/pop_button.d
import 'package:house_of_tomorrow/util/lang/generated/l10n.dar
import 'package:provider/provider.dart';

```

```

class CartView extends StatelessWidget {
  const CartView({super.key});

  @override
  Widget build(BuildContext context) {
    return BaseView(
      viewModel: CartViewModel(
        cartService: context.read(),
      ),
      builder: (context, viewModel) => Scaffold(
        appBar: AppBar(
          title: Text(S.current.cart),
          leading: const PopButton(),
          titleSpacing: 0,
          actions: [
            /// Delete Button
            Button(
              onPressed: () {
                /// Show delete dialog
                showDialog(
                  context: context,
                  builder: (context) {
                    return CartDeleteDialog(
                      onDeletePressed: viewModel.onDeletePress
                    );
                  },
                );
              },
              text: S.current.delete,

```



```

        type: ButtonType.flat,
        color: context.color.secondary,
        isInactive: viewModel.selectedCartItemList.isEmpty
      ),
    ],
  ),
  body: CartLayout(
    /// CartItemList
    cartItemList: viewModel.cartItemList.isEmpty
      ? const CartEmpty()
      : ListView.builder(
          itemCount: viewModel.cartItemList.length,
          itemBuilder: (context, index) {
            final cartItem = viewModel.cartItemList[index];
            return CartItemTile(
              cartItem: cartItem,
              onPressed: () {
                viewModel.onCartItemPressed(index);
              },
              onCountChanged: (count) {
                viewModel.onCountChanged(index, count);
              },
            );
          },
        ),

    /// CartBottomSheet
    cartBottomSheet: CartBottomSheet(
      totalPrice: viewModel.totalPrice,
      selectedCartItemList: viewModel.selectedCartItemList,
      onCheckoutPressed: () {
        /// Show checkout dialog
        showDialog(
          context: context,
          builder: (context) {
            return CartCheckoutDialog(
              onCheckoutPressed: viewModel.onCheckoutPressed;
            );
          },
        );
      },
    );
  );

```

```

    },
  ),
),
);
}
}

```



Delete Button 수정

```

30      actions: [
31        /// Delete Button
32        Button(
33          onPressed: () {
34            /// Show delete dialog
35            showDialog(
36              context: context,
37              builder: (context) {
38                return CartDeleteDialog(
39                  onDeletePressed: viewModel.onDeletePressed,
40                ); // CartDeleteDialog
41              },
42            );
43          },
44          text: S.current.delete,
45          type: ButtonType.flat,
46          color: context.color.secondary,
47          isInactive: viewModel.selectedCartItemList.isEmpty,
48        ), // Button
49      ],

```



CartItemList 수정

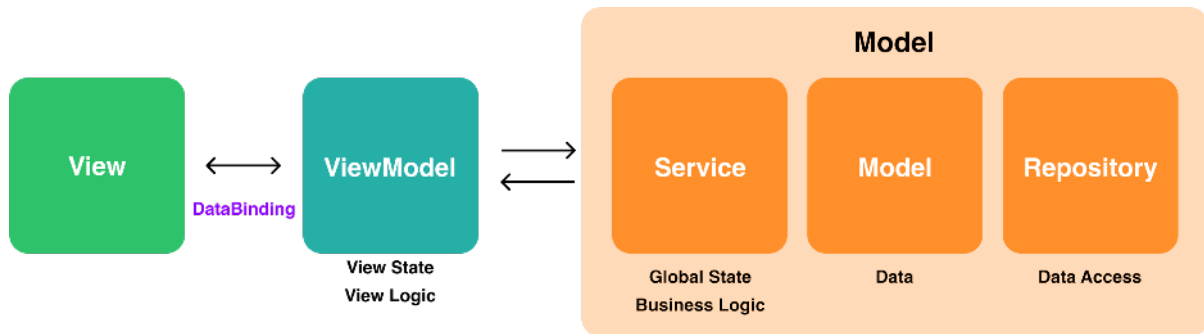
```
51     body: CartLayout(  
52       /// CartItemList  
53       cartItemList: viewModel.cartItemList.isEmpty  
54       ? const CartEmpty()  
55       : ListView.builder(  
56         itemCount: viewModel.cartItemList.length,  
57         itemBuilder: (context, index) {  
58           final cartItem = viewModel.cartItemList[index];  
59           return CartItemTile(  
60             index: index,  
61             cartItem: cartItem,  
62             onPressed: () {  
63               viewModel.onCartItemPressed(index);  
64             },  
65             onCountChanged: (count) {  
66               viewModel.onCountChanged(index, count);  
67             },  
68           ); // CartItemTile  
69         },  
70       ), // ListView.builder
```



CartBottomSheet 수정

```
71  
72     /// CartBottomSheet  
73     cartBottomSheet: CartBottomSheet(  
74       totalPrice: viewModel.totalPrice,  
75       selectedCartItemList: viewModel.selectedCartItemList,  
76       onCheckoutPressed: () {  
77         /// Show checkout dialog  
78         showDialog(  
79           context: context,  
80           builder: (context) {  
81             return CartCheckoutDialog(  
82               onCheckoutPressed: viewModel.onCheckoutPressed,  
83             ); // CartCheckoutDialog  
84           },  
85         );  
86       },  
87     ), // CartBottomSheet  
88   ), // CartLayout
```

▼ MVVM 정리



```

src
├── model
├── repository
├── service
├── view
│   ├── cart
│   │   ├── widget
│   │   ├── cart_view_model.dart
│   │   └── cart_view.dart
│   ├── product
│   │   ├── widget
│   │   ├── product_view_model.dart
│   │   └── product_view.dart
│   ├── shopping
│   │   ├── widget
│   │   ├── shopping_view_model.dart
│   │   └── shopping_view.dart
│   ├── base_view_model.dart
│   └── base_view.dart
  
```



MVVM 아키텍처를 다음과 같이 구현하였습니다.

- **View** : 화면
 - **ViewModel** : 화면 상태 & 로직
-
- **Service** : 비즈니스 로직 & 전역 상태
 - **Model** : 데이터 클래스 정의
 - **Repository** : 데이터 요청



MVVM은 UI와 비즈니스 로직을 **View**와 **ViewModel**로 분리하고, **데이터바인딩**을 이용해 **ViewModel**이 **View**에 대한 의존성을 갖지 않도록 만들어, **View**를 손쉽게 변경할 수 있는 아키텍처입니다.



한 걸음 더 나아가 특정 프레임워크나 Database에도 종속되지 않는 시스템을 만들고 싶으신 경우, **Clean Architecture** 학습을 권장 드립니다.

Clean Coder Blog



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

04. 소프트웨어 테스트

▼ 테스트의 필요성



소프트웨어 테스트란, **제품의 품질에 관한 정보를 조사하는 과정**이다. 또한 **소프트웨어에 대한 객관적이고 독립적인 시각을 제공**한다. (출처 - 위키백과)



기능이 하나뿐인 서비스라면 수작업으로 테스트를 진행해도 간단하지만, **기능이 수백 개인 서비스를 테스트하려면 시간과 비용이 많이 들기 때문에 테스트 코드를 작성하여 자동화하는 게 경제적**입니다.



Flutter는 누구든지 코드를 추가하거나 수정할 수 있는 오픈소스 프로젝트입니다. 다만 코드를 반영하기 위해선 **기존에 작성된 테스트 코드를 모두 통과해야 합니다**. 이와 같이 **대규모 프로젝트에서 테스트 코드를 이용해 소프트웨어의 품질을 관리할 수 있습니다**.


<https://github.com/flutter/flutter>



Flutter 프로젝트에 이미 수많은 테스트 코드들이 작성되어 있기 때문에 테스트 코드 작성하는 구체적인 사례를 참고 할 수 있습니다. 아래 링크에선 Material 위젯들의 테스트 코드를 확인할 수 있습니다.

flutter/packages/flutter/test/material at master · flutter/flutter

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

 <https://github.com/flutter/flutter/tree/master/packages/flutter/test/material>



Flutter
flutter.dev

▼ Flutter 테스트 종류



Flutter 테스트는 다음과 같이 세 카테고리로 나눌 수 있습니다.


1. **단위 테스트(Unit Test)** : 특정 함수 및 클래스 테스트
2. **위젯 테스트(Widget Test)** : 단일 위젯 테스트
3. **통합 테스트(Integration Test)** : 앱의 전체적인 동작 테스트



Flutter 테스트에 대한 공식 문서는 다음 링크를 참고해 주세요.

Testing Flutter apps

Learn more about the different types of testing and how to write them.

 <https://docs.flutter.dev/testing>



Flutter

05. 단위 테스트


▼ 단위 테스트



단위 테스트(Unit Test)는 함수 또는 클래스의 동작을 확인하는 테스트입니다. 단위 테스트에 대한 자세한 내용은 공식 문서를 참고해 주세요.

An introduction to unit testing

How can you ensure that your app continues to work as you add more features or change existing functionality? By writing tests. Unit tests are handy for verifying the behavior of a single

 <https://docs.flutter.dev/cookbook/testing/unit/introduction>



단위 테스트(Unit Test)는 `test` 패키지를 사용하여 테스트를 진행할 수 있습니다.



Flutter SDK에 내장되어 있는 `flutter_test` 패키지는 내부적으로 `test` 패키지를 기반으로 만들어져 있으며, `flutter_test` 패키지가 `dev_dependencies`에 기본적으로 추가되어 있기 때문에 별도 패키지 설치 없이 테스트 코드를 작성할 수 있습니다.

```
48 dev_dependencies:
49   flutter_test:
50     sdk: flutter
51
```



단위 테스트는 `test` 폴더 밑에 작성하며, 테스트 파일의 이름은 항상 `_test.dart`로 끝나야 테스트 실행기가 해당 파일을 테스트 파일로 인식합니다.

```
lib/
  counter.dart
test/
  counter_test.dart
```

예를들어 `counter.dart` 파일의 테스트 코드를 작성하는 경우 `counter_test.dart`라고 파일을 생성하면 됩니다.



단위 테스트는 `test()` 함수를 사용하여 진행합니다.

```
void main() {  
  test("테스트1", () {  
    /// 테스트 작성  
  });  
}
```



`expect(실제값, 기댓값)` 함수를 이용하여 값을 검증 할 수 있으며, 값이 일치하지 않으면 테스트가 실패합니다.

```
test("테스트 이름", () {  
  expect("1", "2"); // 실패  
  expect("1", "1"); // 성공  
});
```


`expect()` 는 Matcher 라이브러리에서 제공되는 기능을 활용하여 보다 다양하게 값을 검증 할 수 있습니다.

```
test("테스트 이름", () {  
  expect(10, greaterThan(5));  
  expect(5, lessThan(10));  
});
```

Matcher가 가진 보다 다양한 기능들은 아래 문서를 참고해 주세요.

matcher library - Dart API

Support for specifying test expectations, such as for unit tests. A useful utility class for implementing other matchers through inheritance. Derived classes should call the base constructor with a feature name and description, and an instance matcher, and should implement the `featureValueOf` abstract method. Matchers

 https://api.flutter.dev/flutter/package-matcher_matcher/package-matcher_matcher-library.html



`group()` 함수를 이용해 `test()` 들을 묶어 테스트를 관심사 별로 묶을 수 있습니다.

```
void main() {  
  group('그룹1', () {  
    test("테스트1", () {});  
  
    test("테스트2", () {});  
  });  
}
```

`group()` 안에 `group()` 을 만들 수 있습니다.

```
void main() {  
  group('그룹1', () {  
    group('그룹2', () {  
      test("테스트1", () {});  
  
      test("테스트2", () {});  
    });  
  });  
}
```



`setUp()` 는 테스트가 실행되기 전에 호출되는 함수로, 테스트 초기 환경을 동일하게 만들 때 사용합니다.

```
void main() {
  setUp() {
    /// 테스트1 실행 전 실행
  }

  test("테스트1", () {});
}
```

아래 코드의 `setUp()` 함수는 `setUp → 테스트1 → setUp → 테스트2` 순서로 두 번 실행 됩니다.

```
void main() {
  setUp() {
    /// 테스트1 실행 전 실행
    /// 테스트2 실행 전 실행
  }

  test("테스트1", () {});

  test("테스트2", () {});
}
```

`setUp()` 함수를 그룹 내에서 호출하면 해당 그룹의 테스트에만 적용되어 `테스트1 → setUp → 테스트2 → setUp → 테스트3` 순서로 실행 됩니다.

```
void main() {
  test("테스트1", () {});

  group('그룹', () {
    setUp() {
      /// 테스트2 실행 전 실행
      /// 테스트3 실행 전 실행
    }

    test("테스트2", () {});

    test("테스트3", () {});
  });
}
```

```
});  
}
```



`setUpAll()` 는 테스트 시작 전 최초 한 번만 실행 됩니다.

아래 코드의 코드는 `setUpAll` → `테스트1` → `테스트2` 순서로 실행 됩니다.

```
void main() {  
  setUpAll() {  
    /// 최초 한 번만 실행  
  }  
  
  test("테스트1", () {});  
  
  test("테스트2", () {});  
}
```



`tearDown()` 는 테스트 실행 후 호출되는 함수입니다.

```
void main() {
  tearDown() {
    /// 테스트1 종료 후 실행
  }

  test("테스트1", () {
    /// 테스트 작성
  });
}
```

아래 코드의 `tearDown()` 함수는 `테스트1 → tearDown → 테스트2 → tearDown` 순서로 두 번 실행됩니다.

```
void main() {
  tearDown() {
    /// 테스트1 실행 후 실행
    /// 테스트2 실행 후 실행
  }

  test("테스트1", () {
    /// 테스트 작성
  });

  test("테스트2", () {
    /// 테스트 작성
  });
}
```

`tearDown()` 함수를 그룹 내에서 호출하면 해당 그룹의 테스트에만 적용되어 `테스트1 → 테스트2 → tearDown → 테스트3 → tearDown` 순서로 실행됩니다.

```
void main() {
  test("테스트1", () {});

  group('그룹', () {
    tearDown() {
      /// 테스트2 실행 후 실행
    }
  });
}
```

```

        /// 테스트3 실행 후 실행
    }

    test("테스트2", () {});

    test("테스트3", () {});
  });
}

```



`tearDownAll()` 는 테스트 종료 후 마지막 한 번만 실행 됩니다.

아래 코드의 코드는 `테스트1` → `테스트2` → `tearDownAll` 순서로 실행 됩니다.

```

void main() {
  tearDownAll() {
    /// 마지막 한 번만 실행
  }

  test("테스트1", () {});

  test("테스트2", () {});
}

```

▼ Dummy



테스트를 진행하는데 필요한 가짜 데이터(Dummy Data)를 만들도록 하겠습니다.

1. `test` 폴더의 `widget_test.dart` 파일 이름을 `dummy.dart` 파일로 변경한 뒤 다음 코드를 붙여 넣어 주세요.

▼ 코드스니펫 - `dummy.dart` : 최종

```

import 'package:flutter/material.dart';
import 'package:house_of_tomorrow/src/model/cart_item.dart';
import 'package:house_of_tomorrow/src/model/lang.dart';
import 'package:house_of_tomorrow/src/model/product.dart';
import 'package:house_of_tomorrow/src/model/product_color.dart

```

```

abstract class Dummy {
  static const Product product = Product(
    name: Lang(
      ko: "3인용 섹션",
      en: "3-seat section",
    ),
    brand: Lang(
      ko: "쇠데르함",
      en: "SÖDERHAMN",
    ),
    desc: Lang(
      ko: "스타일리시하고 산뜻한 느낌이 좋다면 시트가 깊고 넓은 소파는 어떠세요?",
      en: "If you like the stylish airy look, you have to try it.",
    ),
    price: 6990000,
    priceUnit: "₩",
    rating: "4.9",
    productColorList: [
      ProductColor(
        imageUrl:
          "https://i.ibb.co/YTRbKqD/01-soederhamn-3-seat-section",
        color: Color(0xFFEBE1D8),
      ),
      ProductColor(
        imageUrl:
          "https://i.ibb.co/ZLB8kfW/02-soederhamn-3-seat-section",
        color: Color(0xFFC7BC7BF),
      )
    ],
  );

  static const String jsonProductList = '''[
    {
      "name": {
        "ko": "3인용 섹션",
        "en": "3-seat section"
      },
      "brand": {
        "ko": "쇠데르함",

```

```

        "en": "SÖDERHAMN"
      },
      "desc": {
        "ko": "스타일리시하고 산뜻한 느낌이 좋다면 시트가 깊고 넓은 소파는",
        "en": "If you like the stylish airy look, you have to",
      },
      "price": 699000,
      "priceUnit": "₩",
      "rating": "4.9",
      "colorList": [
        {
          "imageUrl": "https://i.ibb.co/YTRbKqD/01-soederhamn-",
          "hexColor": "0xFFEBE1D8"
        },
        {
          "imageUrl": "https://i.ibb.co/ZLB8kfw/02-soederhamn-",
          "hexColor": "0xFFCBC7BF"
        }
      ]
    },
    {
      "name": {
        "ko": "2인용 소파",
        "en": "2-seat sofa"
      },
      "brand": {
        "ko": "페루프",
        "en": "PÄRUP"
      },
      "desc": {
        "ko": "이 커버는 폴리에스테르 소재의 GUNNARED/군나레드 원착 패브",
        "en": "Do you believe in love at first sight? Sleek de",
      },
      "price": 499000,
      "priceUnit": "₩",
      "rating": "4.8",
      "colorList": [
        {
          "imageUrl": "https://i.ibb.co/D9dg49Y/06-paerup-3-se",
          "hexColor": "0xFFE1DAD1"
        }
      ]
    }
  ]
}

```

```

    },
    {
      "imageUrl": "https://i.ibb.co/0B9jMft/07-paerup-3-sea",
      "hexColor": "0xFF4D6452"
    }
  ]
}
]
'''

static const CartItem cartItem = CartItem(
  product: product,
  colorIndex: 0,
  count: 1,
  isSelected: true,
);
}

```



`Dummy.product` : Product 인스턴스

```

abstract class Dummy {
  static const Product product = Product(
    name: Lang(
      ko: "3인용 섹션",
      en: "3-seat section",
    ), // Lang
    brand: Lang(
      ko: "신데렐라"

```




`Dummy.jsonProductList` : 상품 API 호출시 응답으로 넘어오는 JSON 형태의 문자열

```
static const String jsonProductList = '''[
  {
    "name": {
      "ko": "3인용 섹션",
      "en": "3-seat section"
    },
    "brand": {
      "ko": "쇠데르함",
      "en": "SÖDERHAMN"
    }
  }
]
```



`Dummy.cartItem` : CartItem 인스턴스

```
static const CartItem cartItem = CartItem(
  product: product,
  colorIndex: 0,
  count: 1,
  isSelected: true,
);
```

▼ CartService 테스트



CartService 클래스의 단위 테스트를 작성해 봅시다.

1. `test` 폴더 밑에 `src/service/cart_service_test.dart` 파일을 만들고 다음과 같이 작성해 주세요.

▼ 코드스니펫 - `cart_service_test.dart` : 시작코드

```
import 'package:flutter_test/flutter_test.dart';
import 'package:house_of_tomorrow/src/service/cart_service.dart';

void main() {
  late CartService cartService;

  setUp(() {
    cartService = CartService();
  });
}
```

```

group('CartService', () {
  group('add()', () {
    test('신규 CartItem을 cartItemList에 추가한다.', () {});
  });

  group('selectedCartItem', () {
    test('isSelected가 true인 CartItem만 반환한다.', () {});
  });

  group('update()', () {
    test('선택한 index의 CartItem을 수정한다.', () {});
  });

  group('delete()', () {
    test('deleteList에 포함된 cartItemList의 CartItem을 삭제한다.', () {});
  });
});
}

```

```
test > src > service > cart_service_test.dart > ...
1 import 'package:flutter_test/flutter_test.dart';
2 import 'package:house_of_tomorrow/src/service/cart_service.dart';
3
4 void main() {
5   late CartService cartService;
6
7   setUp(() {
8     cartService = CartService();
9   });
10
11   group('CartService', () {
12     group('add()', () {
13       test('신규 CartItem을 cartItemList에 추가한다.', () {});
14     });
15
16     group('selectedCartItem', () {
17       test('isSelected가 true인 CartItem만 반환한다.', () {});
18     });
19
20     group('update()', () {
21       test('선택한 index의 CartItem을 수정한다.', () {});
22     });
23
24     group('delete()', () {
25       test('deleteList에 포함된 cartItemList의 CartItem을 삭제한다.', () {});
26     });
27   });
28 }
```



모든 테스트 시작 전 `setUp()` 함수에서 `CartService` 클래스를 새로 생성하여 테스트 조건을 동일하게 통일하였습니다.



전체 테스트를 `CartService` 그룹으로 묶고, 하위에 테스트하려는 메소드를 그룹으로 만들었습니다. 그리고 각 메소드에 테스트하려는 기능을 명시하였습니다.



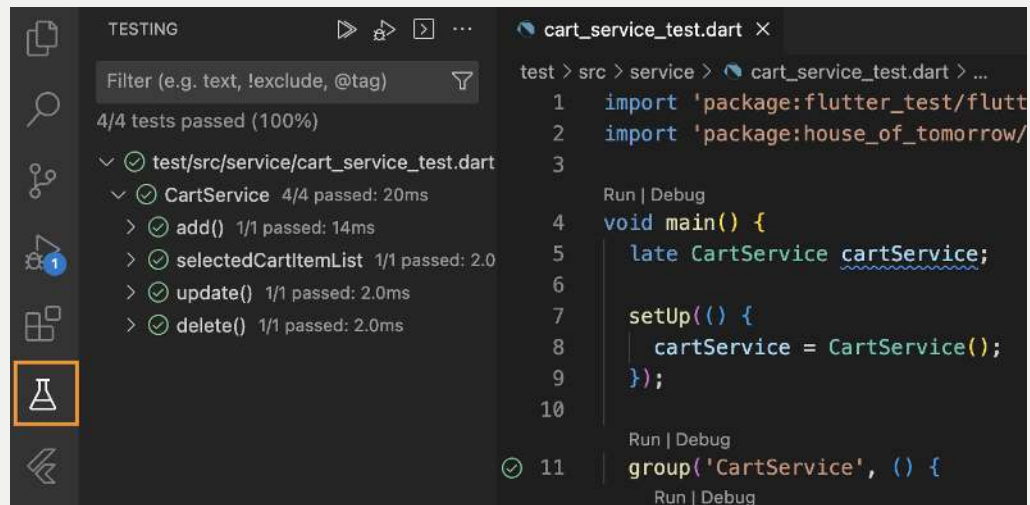
`group` 이나 `test` 왼쪽에 초록색 화살표를 눌러 테스트를 실행할 수 있습니다.

```

Run | Debug
11 | group('CartService', () {
    Run | Debug
12 |   group('add()', () {
        Run | Debug
13 |     test('신규 CartItem을 cartItemList에 추가한다.', () {});
14 |   });
15 |
    Run | Debug
16 |   group('selectedCartItem', () {
        Run | Debug
17 |     test('isSelected가 true인 CartItem만 반환한다.', () {});
18 |   });

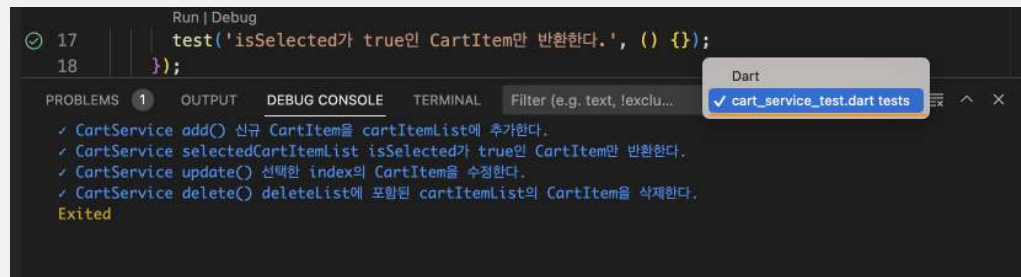
```

11번째 줄의 화살표를 누르면 좌측에 테스트 패널이 열리면서 테스트 결과가 나옵니다.



테스트 항목 좌측 화살표를 눌러서 펼치면 위 이미지 같이 확인할 수 있습니다.

또한 Debug Console에서 `cart_service_test.dart tests` 를 선택하여 테스트 실행시 출력된 결과를 확인할 수 있습니다..



2. `add()` 그룹의 테스트를 다음과 같이 작성해 주세요.

▼ 코드스니펫 - `cart_service_test.dart` : `add()`

```
cartService.add(Dummy.cartItem);  
expect(cartService.cartItemList.length, 2);
```

```
4+ import '../dummy.dart';  
5+  
6 void main() {  
7   late CartService cartService;  
8  
9   setUp(() {  
10     cartService = CartService();  
11   });  
12  
13   group('CartService', () {  
14     group('add()', () {  
15+       test('신규 CartItem을 cartItemList에 추가한다.', () {  
16+         cartService.add(Dummy.cartItem);  
17+         expect(cartService.cartItemList.length, 2);  
18+       });  
19     });  
20
```



16번째 줄에서 `cartService`에 `Dummy.cartItem` 을 하나만 넣었는데, 17번째 줄에서 `cartItemList.length` 값을 2로 기대했기 때문에 해당 테스트는 실패할 것입니다.



15번째 줄 왼쪽에 화살표 아이콘을 누르면 해당 테스트만 실행해 봅시다.

```
Run | Debug  
15 test('신규 CartItem을 cartItemList에 추가한다.', () {  
16   cartService.add(Dummy.cartItem);  
17   expect(cartService.cartItemList.length, 2);  
18 });
```