

s_step2.dart

소스 코드

전체 코드

```
1 import 'package:buyoil/model/ui_state_step1.dart';
2 import 'package:buyoil/model/ui_state_step2.dart';
3 import 'package:buyoil/view/widget/circular_prograss.dart';
4 import 'package:buyoil/view/widget/w_step_nav.dart';
5 import 'package:buyoil/viewmodel/vm_step1.dart';
6 import 'package:easy_localization/easy_localization.dart';
7 import 'package:flutter/material.dart';
8 import 'package:flutter_riverpod/flutter_riverpod.dart';
9 import 'package:go_router/go_router.dart';

10
11 import '../../../../../common/app_colors.dart';
12 import '../../../../../common/app_strings.dart';
13 import '../../../../../common/app_styles.dart';
14 import '../../../../../router.dart';
15 import '../../../../../viewmodel/vm_step2.dart';
16 import '../widget/w_header.dart';

17
18 class Step2Screen extends ConsumerStatefulWidget {
19   const Step2Screen({Key? key}) : super(key: key);
20
21   @override
22   ConsumerState<ConsumerStatefulWidget> createState() => Step2ScreenState();
23 }

24
25 class Step2ScreenState extends ConsumerState<Step2Screen> {
26   @override
27   void initState() {
28     super.initState();
29     afterLayout();
30   }
31
32   @override
33   Widget build(BuildContext context) {
34     final notifier = ref.watch(step2Provider.notifier);
35     final state = ref.watch(step2Provider);
36     return Scaffold(
37       body: Column(
38         children: [
39           HeaderWidget(),
40           Expanded(
41             child: state.when(
42               init: () {
43                 return _initBody();
44               },
45               completed: () {
```

```

46             return Container();
47         })
48     ],
49 )
50 );
51 }
52 }

53
54 widget? _openButton(BuildContext context, WidgetRef ref) {
55     return Inkwell(
56         borderRadius: BorderRadius.circular(338 / 2),
57         onTap: () {
58             ref.watch(step2Provider.notifier).pressedOpen();
59         },
60         child: Container(
61             width: 338,
62             height: 338,
63             decoration: BoxDecoration(image: DecorationImage(image:
64                 Image.asset("${AppStrings.assetPath}img_open_btn.png", width: 338, height:
65                 338,.image)),
66             child: Center(
67                 child: Text(AppStrings.openAction.tr(), style: AppStyles.tsOpenBtn,),
68             ),
69         );
70     }
71 }

72 void afterLayout() {
73     ref.listenManual(step2Provider, (_, state) {
74         if(state is UIStateStep2Completed) {
75             context.goNamed(RouteGroup.Step3.name);
76         } else {
77             print("State:: ${state.runtimeType}");
78         });
79     }
80 }

81 widget _initBody() {
82     return Row(
83         mainAxisAlignment: MainAxisAlignment.spaceBetween,
84         children: [
85             StepNavwidget(currentStep: 2, totalSteps: 4),
86             Expanded(
87                 child: Center(
88                     child: _openButton(context, ref),
89                 )
90             ),
91         ],
92     );
93 }

94 _openDoorBody() {
95     return Container(

```

```

97     width: double.infinity,
98     height: double.infinity,
99     color: AppColors.EFFDF6,
100    alignment: Alignment.center,
101    child: Column(
102      mainAxisAlignment: MainAxisAlignment.center,
103      children: [
104        Container(
105          width: 110, height: 110,
106          child: RotatingImageCircularProgressBar(
107            progress: 50,
108            imagePath: "${AppStrings.assetPath}img_circular.png",),
109          ),
110          SizedBox(height: 24 - 3,),
111          Text(AppStrings.doorOpening.tr(), style: AppStyles.tsDoorOpeningText,),
112          SizedBox(height: 3,),
113        ],
114      ),
115    );
116  }
117 }

```

1. Step2Screen 역할 개요

Step2Screen은 4단계 중 Step 2: 문 열기(Open) 기능을 담당하는 화면이다.

주요 역할은 다음과 같다.

- Step2 상태(UIStateStep2)에 기반해 화면 전환 및 UI 렌더링
- “문 열기(Open)” 버튼 노출 및 클릭 이벤트 처리
- 버튼 누름 이벤트를 ViewModel(step2Provider)로 전달
- 상태가 `UIStateStep2Completed`로 전환될 경우 Step3 화면으로 네비게이션
- StepNavWidget을 통해 현재 Step(2/4) 표시
- 필요 시 진행 중 UI(회전 로딩 UI) 구성(현재 Branch에서는 미사용)

2. 전체 구조 흐름

```

1 Step2Screen (ConsumerStatefulWidget)
2   └ Step2ScreenState (UI 상태 감시 + 이벤트 처리)
3     └ initState → afterLayout() (상태 리스너 등록)
4     └ state = ref.watch(step2Provider)
5     └ notifier = ref.watch(step2Provider.notifier)
6     └ HeaderWidget()
7     └ 상태별 UI 출력 (state.when)
8       └ init → _initBody()
9       └ completed → Container()
10      └ _openButton() / _openDoorBody() (UI 구성 함수)

```

3. build() 분석

3.1 상태 로드 및 Watch

```
1 final notifier = ref.watch(step2Provider.notifier);  
2 final state = ref.watch(step2Provider);
```

- **state**: UIStateStep2 (init, completed 등)
- **notifier**: Step2 ViewModel (pressedOpen 등의 비즈니스 로직 처리)

3.2 전체 UI 레이아웃

```
1 Scaffold  
2   └── Column  
3     ┌── HeaderWidget()  
4     └── Expanded(  
5       child: state.when(  
6         init: _initBody()  
7         completed: Container()  
8       )  
9     )
```

구성 요약:

- 상단: 고정 헤더
- 본문: 상태에 따라 다른 구성 출력
 - init → 문 열기 버튼 포함한 기본 UI
 - completed → 비어있는 화면(즉시 Step3로 이동하므로 UI 표시 없음)

4. 상태 전파 및 네비게이션 처리

4.1 initState → afterLayout() 호출

```
1 @override  
2 void initState() {  
3   super.initState();  
4   afterLayout();  
5 }
```

4.2 afterLayout 내부 동작

```
1 ref.listenManual(step2Provider, (_, state) {  
2   if (state is UIStateStep2Completed) {  
3     context.goNamed(RouteGroup.Step3.name);  
4   }  
5 });
```

- Flutter 위젯 빌드 직후에 실행되는 후처리용 패턴
- listenManual 을 사용하여 원하는 시점에 구독
- 상태가 `UIStateStep2Completed` 으로 변경되면 즉시 Step3으로 이동
- 네비게이션 로직은 UIState 변화에 완전히 종속되어 있음

5. `_initBody()` 상세 분석

`UIStateStep2.init` 상태에서 호출되는 기본 UI.

레이아웃 구조

```

1 | Row
2 |   └─ StepNavWidget(currentstep: 2, totalSteps: 4)
3 |     └─ Expanded
4 |       └─ Center
5 |         └─ _openButton()

```

StepNavWidget

- 좌측 네비게이션 영역
- 현재 단계: 2 / 전체 4

`_openButton()`

가운데 배치된 대형 원형 버튼 UI.

6. `_openButton()` UI 상세 분석

```

1 | InkWell
2 |   └─ Container(338x338, background image)
3 |     └─ Center → Text(openAction)

```

주요 특징

- InkWell로 만들어져 있으므로 클릭 시 터치 피드백 존재
- 버튼 전체가 이미지(`img_open_btn.png`)로 구성됨
- 이미지 위에 Text(`AppStrings.openAction`) 오버레이
- onTap 이벤트:

```
1 | ref.watch(step2Provider.notifier).pressedOpen();
```

즉, UI는 단순 이벤트 포워딩만 담당하고
문 열기 요청 비즈니스 로직은 ViewModel에서 처리.

7. _openDoorBody() 분석

현재 state.when에서 사용되지 않지만, 구조상 다음 목적을 가진 것으로 보임:

- 문 열림 중 “진행 상태”를 시각적으로 표현하는 UI
- 로딩용 Circular Progress + 텍스트 구성

UI 구조

```
1 Container (full size, 배경 E7FDF6)
2   └─ column(center)
3     └─ RotatingImageCircularProgressBar
4     └─ Text(doorOpening)
```

ViewModel에서 상태가 “opening”과 같은 추가 상태를 가질 경우
이 UI를 출력하도록 확장 가능.

8. UIStateStep2 활용 구조

해당 프로젝트 전반의 Freezed 기반 상태 모델을 사용하므로
UIStateStep2 역시 다음과 같은 구조일 가능성성이 높다.

예상 모델:

```
1 @freezed
2 class UIStateStep2 with _$UIStateStep2 {
3   const factory UIStateStep2.init() = UIStateStep2Init;
4   const factory UIStateStep2.completed() = UIStateStep2Completed;
5 }
```

특징:

- Step2는 입력 상태가 없으므로 단순하게 init → completed 두 상태만 존재
- pressedOpen() → 로직 실행 → 상태 completed 전환 → 라우팅 발생

9. 화면 동작 흐름 (정리)

1. Step2Screen 로드
2. afterLayout()으로 상태 구독 시작
3. state = init
4. 사용자가 “열기(Open)” 버튼 클릭
5. pressedOpen() 실행 → ViewModel이 서버/디바이스와 통신
6. 작업 성공 시 UIStateStep2Completed로 상태 변경
7. 상태 리스너가 Step3 화면으로 즉시 이동 처리
8. completed UI는 별도로 표시되지 않음

10. 코드 관점 개선 제안 (선택)

단순 권고이며 요청 시 실제 코드 제공 가능.

1) `_openDoorBody()` 상태 분리하여 사용

현재 미사용. `opening` 상태 추가 후 로딩 UI로 활용하는 것이 구조적 일관성 향상.

2) `context.goNamed` 대신 `sealed routing function` 사용

RouterGroup 엔트리포인트를 통일하면 maintainability 향상.

3) `InkWell + Container` → `GestureDetector + DecoratedBox` 구조로 단순화 가능

Material ripple 효과가 필요 없을 경우 InkWell은 과한 구성.

4) 버튼 사이즈(338x338) 하드코딩 개선

LayoutBuilder 기반 비율 스케일링을 사용하면 다양한 화면에서 안정적 렌더링 가능.

11. 제공 가능한 추가 분석

필요 시 다음 문서도 동일한 형식으로 제공한다.

- Step2 ViewModel 내부 `pressedOpen()` 로직 해석
- Step2 UIState 전이 시퀀스 다이어그램
- Step2 전체 MVVM 아키텍처 구조도
- Step1~Step4 전체 화면 흐름 및 상태 전이 지도(State Flow Map)
- Step2 최적화된 리팩토링 버전 코드