s_driver.dart

소스 코드

전체 코드

```dart
import 'package:buyoil/view/widget/w_step_nav.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:go_router/go_router.dart';

import '../../common/app_colors.dart';
import '../../common/app_strings.dart';
import '../../common/app_styles.dart';
import '../../router.dart';
import '../../viewmodel/vm_driver.dart';
import '../widget/w_header.dart';

class DriverScreen extends ConsumerStatefulWidget {
  const DriverScreen({Key? key}) : super(key: key);

  @override
  ConsumerState<ConsumerStatefulWidget> createState() => DriverScreenState();
}

class DriverScreenState extends ConsumerState<DriverScreen> {
  @override
  void initState() {
    super.initState();
    afterLayout();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: [
          HeaderWidget(),
          Expanded(
            child: _initBody()
          )
        ],
      )
    );
  }

  Widget? _closeButton(BuildContext context, WidgetRef ref) {
    return InkWell(
      borderRadius: BorderRadius.circular(338 / 2),
      onTap: () {
```

```
        ref.watch(driverProvider.notifier).pressedClose();
      },
      child: Container(
        width: 338,
        height: 338,
        decoration: BoxDecoration(image: DecorationImage(image:
Image.asset("${AppStrings.assetPath}img_close_btn.png", width: 338, height:
338,).image)),
        child: Center(
          child: Text(AppStrings.closeAction.tr(), style: AppStyles.tsOpenBtn,),
        ),
      )
    );
  }

  void afterLayout() {
    // ref.listenManual(DriverProvider, (_, state) {
    //   if(state is UIStateDriverCompleted) {
    //     context.goNamed(RouteGroup.Step4.name);
    //   }
    // });
  }

  Widget _initBody() {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        StepNavWidget(currentStep: 3, totalSteps: 4),
        Expanded(
            child: Center(
              child: _closeButton(context, ref),
            )
        )
      ],
    );
  }

  _closeDoorBody() {
    return Container(
      width: double.maxFinite,
      height: double.maxFinite,
      color: AppColors.EFFDF6,
    );
  }
}
```

DriverScreen 구조 분석

본 파일은 `DriverScreen`이라는 화면(View)을 정의하며, Riverpod 기반의 상태 관리와 GoRouter 기반의 화면 이동을 수행한다.

MVVM 아키텍처 기준으로 **UI 레이어(View)**에 해당한다.

클래스 개요

`class DriverScreen extends ConsumerStatefulWidget`

- Riverpod Consumer 기능을 포함하는 StatefulWidget이다.
- ViewModel의 상태를 관찰하거나 Provider 호출이 필요한 화면에서 사용된다.

`class DriverScreenState extends ConsumerState<DriverScreen>`

- StatefulWidget의 실제 상태를 담당한다.
- `ConsumerState`를 사용하므로 `ref` 객체를 통해 Provider와 직접 상호작용이 가능하다.

화면 구성 구조

```
Scaffold
└─ Column
   ├─ HeaderWidget
   └─ Expanded
      └─ _initBody()
         └─ Row
            ├─ StepNavWidget
            └─ Expanded
               └─ Center
                  └─ _closeButton()
```

주요 구성 요소 상세 설명

1. initState() → afterLayout()

```
@override
void initState() {
  super.initState();
  afterLayout();
}
```

역할

- 화면이 빌드되기 전에 한 번 실행되는 초기 작업을 처리한다.
- 주석처리된 영역은 원래 다음 기능을 수행한다.

```
ref.listenManual(driverProvider, (_, state) {
  if (state is UIStateDriverCompleted) {
    context.goNamed(RouteGroup.Step4.name);
  }
});
```

즉,
Driver 작업이 완료되는 상태(UIStateDriverCompleted)가 감지되면 Step4 화면으로 이동하는 기능이 설계되어 있으나 현재는 비활성화되어 있다.

---

2. `_closeButton()`

```
Widget? _closeButton(BuildContext context, WidgetRef ref)
```

기능

- 중앙에 노출되는 닫기(Close) 버튼을 렌더링한다.
- 실제 버튼을 눌렀을 때 다음 코드가 실행된다:

```
ref.watch(driverProvider.notifier).pressedClose();
```

의미

View → ViewModel 호출
즉,

- View는 단순하게 "사용자가 버튼을 눌렀다"는 이벤트만 전달하고
- 실제 로직(문 닫기, 상태 변경 등)은 Driver ViewModel에서 처리한다.

---

3. `_initBody()`

```
Widget _initBody()
```

기능

화면의 메인 UI를 구성한다.

- 좌측: StepNavWidget (진행 단계 UI)
- 우측: 닫기 버튼

전형적인 Wizard 단계 기반 UI이다.

---

4. `_closeDoorBody()` (미사용)

닫는 동작 시 표시할 가능성이 있는 배경 레이아웃이다.
현재 화면에서는 호출되지 않지만 향후 “문 닫는 애니메이션 또는 상태 표시 UI”로 확장될 여지가 있다.

아키텍처 관점 분석

DriverScreen은 MVVM 관점에서 다음 기능을 수행한다.

---

1. View 역할

- UI 렌더링
- 사용자 입력 수신
- ViewModel 메서드 호출
- 상태 변화에 따라 화면 이동 수행 가능

---

2. ViewModel과의 연결

DriverScreen은 다음 Provider와 연결되어 있다:

```
driverProvider (vm_driver.dart)
```

역할:

- 버튼 입력 → `pressedClose()`
- 상태 변경 감지 → Step4 화면 이동 (현재 주석)

이는 전형적인 MVVM 구조로

```
View → ViewModel → (State 변화) → View 업데이트
```

흐름을 따른다.

---

3. 라우팅과의 결합

RouteGroup.Step4를 참조하며,
Driver 동작이 완료되면 자동으로 다음 단계로 이동하도록 설계되어 있다.
이는 GoRouter 기반의 선언적 라우팅 구조와 일치한다.

종합 요약

DriverScreen은 다음의 명확한 역할을 수행한다.

- Header + 단계 네비게이션 + 중앙 버튼으로 구성된 화면
- ViewModel(driverProvider) 상태를 관찰하고 이벤트 전달
- close 버튼 입력 시 ViewModel의 pressedClose() 호출
- 특정 상태 발생 시 다음 단계로 화면 이동하도록 설계(현재 비활성화)
- 앱 전체 흐름에서 "문을 닫는 단계"를 관리하는 Step 3 또는 Step 중간 화면 역할