s_splash.dart

소스 코드

전체 코드

```dart
import 'package:buyoil/view/widget/btn_home.dart';
import 'package:buyoil/view/widget/btn_to_setting.dart';
import 'package:buyoil/viewmodel/vm_serial_port.dart';
import 'package:easy_localization/easy_localization.dart';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

import '../../common/app_colors.dart';
import '../../common/app_strings.dart';
import '../../common/app_styles.dart';
import '../../config.dart';
import '../widget/btn_to_connect_port.dart';


class SplashScreen extends ConsumerStatefulWidget {
  const SplashScreen({super.key});

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

class SplashState extends ConsumerState<SplashScreen> {
  @override
  Widget build(BuildContext context) {
    ref.watch(serialPortVMProvider);

    return Scaffold(
      body: Container(
        width: double.maxFinite,
        height: double.maxFinite,
        color: Colors.white,
        child: Stack(
          children: [
            Positioned(
              top: 58,
              bottom: 208,
              left: 300,
              right: 300,
              child: Image.asset("${AppStrings.assetPath}img_splash.png"),
            ),
            Positioned(
              bottom: 72, //20은 터치 영역 고려
              left: 0,
              right: 0,
              child: Center(
```

```
                child: _startButton(context),
              )
            ),
            Positioned(
              right: 0,
              top: 0,
              child: _homeButton(context),
            ),
            Positioned(
              left: 0,
              top: 0,
              child: _settingButton(context),
            ),
            if(Config.instance.isDebugMode) Positioned(
              right: 120,
              top: 16,
              child: Row(
                children: [
                  Text("device
${ref.watch(serialPortVMProvider).connectedDevice?.deviceId ?? "not connected"}",
style: Theme.of(context).textTheme.bodyLarge),
                  SizedBox(width: 20,),
                  OutlinedButton(
                    onPressed: () {
                      ref.watch(serialPortVMProvider.notifier).connectPort(context:
context);
                    },
                    child: Text("Connect
Port[${ref.watch(serialPortVMProvider).availablePorts.length}]", style:
Theme.of(context).textTheme.labelLarge,)
                  ),
                ],
              )
            ),
            Positioned(
              left: 0,
              bottom: 0,
              child: _connectPortButton(context),
            ),
          ],
        )
      )
    );
  }

  Widget _startButton(BuildContext context) {
    return Padding(
      padding: EdgeInsets.only(bottom: 100),
      child: TextButton(
        onPressed: () {
          ref.read(serialPortVMProvider.notifier).start();
        },
        style: ButtonStyle(
```

```
          foregroundColor: MaterialStateProperty.resolveWith<Color>(
                (Set<MaterialState> states) {
              if (states.contains(MaterialState.pressed)) {
                return AppColors.PRIMARY.withAlpha(80); // 터치(pressed) 상태일 때 색
              }
              return AppColors.PRIMARY; // 기본 색
            },
          ),
          // 선택적으로 배경색도 바꿀 수 있음
          backgroundColor: WidgetStatePropertyAll(Colors.transparent),
        ),
        child: Text(AppStrings.startButton.tr(), style: AppStyles.tsStart,),
      ),
    );

    // return TextButton(
    //   onPressed: () {
    //     // onTap 대신 onPressed를 사용하고, ref.read를 사용하는 것이 권장됩니다.
    //
    //   },
    //   style: TextButton.styleFrom(
    //     // 1. 평상시 텍스트 색상 (초록색)
    //     foregroundColor: Colors.green,
    //     // 2. 버튼이 눌렸을 때의 텍스트 색상 (보라색)
    //     // disabled, error 등의 상태 색상도 여기서 지정 가능합니다.
    //     // 여기서는 pressed 상태의 색상만 변경합니다.
    //     disabledForegroundColor: Colors.grey, // 예시: 비활성화 상태 색상
    //
    //     // 3. 터치 시 나타나는 기본 물결 효과(overlay) 제거
    //     overlayColor: Colors.transparent,
    //
    //     // 4. 나머지 스타일 유지
    //     padding: const EdgeInsets.symmetric(vertical: 20, horizontal: 20),
    //     minimumSize: const Size(0, 57),
    //     shape: RoundedRectangleBorder(
    //       borderRadius: BorderRadius.circular(338 / 2),
    //     ),
    //     // 5. 텍스트의 기본 스타일 (색상 제외)
    //     textStyle: AppStyles.tsStart,
    //   ),
    //   child: Text(),
    // );
  }



  Widget _settingButton(BuildContext context) {
    return ToSettingButton();
  }

  Widget _connectPortButton(BuildContext context) {
    return ConnectPortButton();
  }
```

```
147
148   Widget _homeButton(BuildContext context) {
149     return ToHomeBtn();
150   }
151 }
```

1. 프로젝트의 전반적 아키텍처 분석

본 프로젝트는 다음과 같은 구조적 특징을 가진다.

1.1 주요 아키텍처 패턴

프로젝트는 **Flutter + Riverpod(StateNotifier)** 기반 상태관리 구조를 채택한다.
아키텍처 레이어는 다음과 같이 구분된다.

Presentation Layer

- `view/screen`: 화면 단위 위젯(페이지)
- `view/widget`: 화면에서 사용하는 구성요소(UI component)

State Management Layer

- `viewmodel/`: Riverpod StateNotifier 기반 UI 상태 관리
- 예: `vm_setting.dart`, `vm_serial_port.dart`

Domain / Logic Layer

- 장치 연동 로직(Serial Port)
- 비즈니스 로직(`start()`, `toggleMotor()`, `toggleValve()` 등)

Common / Resource Layer

- 공용 상수: `AppStrings`, `AppStyles`, `AppColors`
- 환경 설정: `config.dart`

구조적 특징:

- MVVM 구조를 변형한 형태
- View — ViewModel 간 단방향 데이터 흐름 유지
- View → ViewModel(notifier) → State → View 재빌드

---

2. SplashScreen 코드 분석

해당 화면은 **앱 초기 진입 화면** 역할을 하며, 시리얼 포트 연결 또는 앱 시작 이벤트를 처리하는 구조이다.

2.1 주요 책임 정의

- 스플래시 이미지를 출력
- 개발 모드에서 디바이스 정보 표시
- Start 버튼으로 장치 시작(Serial Port VM의 start 호출)
- 포트 연결 UI 제공
- 설정, 홈 화면 전환 버튼

2.2 동작 흐름

1. `ref.watch(serialPortVMProvider)` 호출
    - 시리얼 포트 상태 변화를 감지하고 UI 리빌드 트리거
    - SerialPortVM의 내부 초기화 로직 수행 가능
2. UI 구조는 다음과 같은 주요 구성요소로 이루어진다.
    - 중앙 로고 이미지
    - 하단 Start 버튼 (`_startButton`)
    - 좌상단 Setting 버튼
    - 우상단 Home 버튼
    - 좌하단 ConnectPort 버튼
    - Debug 모드에서 디바이스 ID + 포트 연결 버튼 표시
3. Start 버튼 동작
   `ref.read(serialPortVMProvider.notifier).start();`
    - 포트 연결 시작
    - 초기화 이벤트 전송
4. ConnectPort 버튼
   SerialPort 연결 전용 UI 호출

2.3 SplashScreen의 역할 요약

- 초기 진입 화면
- 하드웨어 장치 연결 상태 확인
- 앱의 실제 기능 페이지 진입을 위한 준비 단계

---

3. SettingScreen 코드 분석

설정 화면은 **장치 정보 확인 및 하드웨어 제어 인터페이스**를 제공한다.

3.1 주요 책임 정의

- 오일 컨테이너 정보 표시
- 측정값(Oil/Water) 표시
- Motor, Valve On/Off 제어
- 설정 메뉴로 이동 버튼 제공

3.2 상태 흐름 분석

1. `final notifier = ref.watch(settingProvider.notifier);`
    - UI에서 제어 명령 호출용 핸들러
2. `final state = ref.watch(settingProvider);`
    - 상태 변경 시 UI 자동 리빌드
3. 상태 값 예시

    ```
    state.oilContainer
    state.measureOil
    state.measureWater
    state.isMotorOn
    state.isValveOn
    ```

    모두 SettingViewModel에서 관리

3.3 UI 구성 요소

- HeaderWidget
- 설정 메뉴 열기 버튼
- 4개 열(Column)을 갖는 설정 영역
    - 1: 용기명 + 현재 컨테이너 무게
    - 2: 오일/물 측정값
    - 3: Motor 상태/토글
    - 4: Valve 상태/토글

각 영역은 `SettingBox`, `SettingButton`으로 모듈화됨.

3.4 이벤트 흐름

Motor 버튼 누름 →

```
notifier.toggleMotor()
```

- 내부 센서/장치 제어 로직 실행
- 상태 변경 → UI 자동 갱신

Valve 버튼 동일

---

4. Flutter 프로젝트 코드 분석 시 권장 순서

Flutter 프로젝트를 구조적으로 분석하려면 다음 절차를 따르는 것이 효율적이다.

Step 1. 프로젝트 구조 전체 파악

- lib/ 디렉터리 전체 스캔
- layer 기준 기능 구분 여부 확인(view/widget/viewmodel/common 등)

Step 2. 앱 초기 실행 진입점 파악

- `main.dart` → `MaterialApp` / `Router` / `ProviderScope`
- Start point를 기준으로 화면 흐름 정리

Step 3. 라우팅 구조 파악

- GoRouter / Navigator 사용 여부 확인
- 첫 화면이 무엇인지 파악하여 UI 흐름 산출

Step 4. 상태관리 구조 분석

- Provider, Riverpod, Bloc 등
- StateNotifier → State → View 순서 파악
- ViewModel별 책임을 식별

Step 5. 공용 자원(Common Layer) 조사

- 색상, 폰트, 문자열, 애셋 경로 등
- 공통 스타일 구조를 파악해 UI 분석 속도 향상

Step 6. 각 화면(Screen) 살펴보기

- 화면별 상태 의존성
- 주요 이벤트 핸들러
- 위젯 구성 패턴

Step 7. 비즈니스 로직 및 장치 연동 코드 분석

- Serial Port 연동 / API 호출 / 디바이스 제어 로직
- 상태 변화를 유발하는 핵심 로직

Step 8. 위젯들의 재사용 패턴 확인

- 공통 UI 컴포넌트(SettingButton 등)
- 스타일 일관성 확인

Step 9. 전체 동작 흐름 정리

- 앱 초기 → Splash → 장치 연결 → Setting → 기능 동작
- 전체 시나리오 기반으로 UX/로직 정리를 수행

---

5. 결론

`SplashScreen`은 앱 초기 장치 연결 및 시작을 담당하고,
`SettingScreen`은 장치 상태 표시와 제어를 담당한다.

전체 아키텍처는 다음 특징을 가진다.

- **Riverpod StateNotifier 기반 MVVM 구조**
- UI와 상태의 단방향 흐름 유지
- 장치 제어 로직과 UI 렌더링을 명확히 분리