s_step4.dart

소스 코드

전체 코드

```dart
import 'dart:ui';

import 'package:buyoil/model/ui_state_usb_port.dart';
import 'package:buyoil/view/widget/w_step_nav.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 'package:go_router/go_router.dart';

import '../../common/app_colors.dart';
import '../../common/app_strings.dart';
import '../../common/app_styles.dart';
import '../../model/ui_state_step4.dart';
import '../../router.dart';
import '../../viewmodel/vm_step4.dart';
import '../widget/w_header.dart';

class Step4Screen extends ConsumerStatefulWidget {
  final double water, oil;

  const Step4Screen({super.key, required this.water, required this.oil});

  @override
  ConsumerState<ConsumerStatefulWidget> createState() => Step4ScreenState();
}

class Step4ScreenState extends ConsumerState<Step4Screen> {
  double water = 0.0;
  double oil = 0.0;

  @override
  void initState() {
    water = widget.water;
    oil = widget.oil;
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      if (mounted) { // 콜백이 실행될 때 위젯이 여전히 트리에 있는지 확인 (안전장치)
        ref.watch(serialPortVMProvider.notifier).initPortState();
      }
    });
  }

  @override
  void didUpdateWidget(covariant Step4Screen oldWidget) {
```

```
    if(widget.oil != oldWidget.oil || widget.water != oldWidget.water) {
      setState(() {
        oil = widget.oil;
        water = widget.water;
      });
    }
    super.didUpdateWidget(oldWidget);
  }

  @override
  Widget build(BuildContext context) {
    final notifier = ref.watch(step4Provider.notifier);
    final state = ref.watch(step4Provider);
    return Scaffold(
      body: Column(
        children: [
          HeaderWidget(),
          Expanded(
            child: _body(),
          )
        ],
      )
    );
  }

  void afterLayout() {
    ref.listenManual(step4Provider, (_, state) {
      if(state is UIStateStep4Checked) {
        context.goNamed(RouteGroup.Step1.name);
      }
      if(state is UIStateStep4Retry) {
        context.goNamed(RouteGroup.Step1.name);
      }
    });
  }

  Widget _body() {
    final stateSerial = ref.watch(serialPortVMProvider);

    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        StepNavWidget(currentStep: 4, totalSteps: 4),
        Expanded(
          child: stateSerial.when(
            init: (_,__,___,____,_____) {
              return defaultBody();
            },
            connected: (_,__,___,____,_____) {
              return defaultBody();
            },
            loading: (_,__,___,____,_____) {
              return Stack(
```

```
                alignment: Alignment.center,
                children: [
                  // 1. 원본 위젯
                  defaultBody(),
                  // 2. 블러 효과를 적용할 위젯
                  ClipRect( // 블러 효과가 위젯 경계를 넘어가지 않도록 ClipRect로 감쌉니다.
                    child: BackdropFilter(
                      filter: ImageFilter.blur(sigmaX: 5.0, sigmaY: 5.0), // 블러 강도 조절
                      child: Container(
                        // BackdropFilter는 자식 위젯이 있어야 렌더링됩니다.
                        // 투명한 컨테이너를 배치하여 블러 효과만 적용되도록 합니다.
                        width: double.infinity,
                        height: double.infinity,
                        color: AppColors.PRIMARY.withAlpha(20),
                      ),
                    ),
                  ),
                  // 3. 중앙에 CircularProgressIndicator 추가
                  const CircularProgressIndicator(
                    color: AppColors.PRIMARY, // 원하는 색상으로 변경 가능
                  ),
                ],
              );
            },
            error: (_,__,___,____,_____,______) {
              return defaultBody();
            },)
        ),
      ],
    );
  }

  _checkButton() {
    return ElevatedButton(
      onPressed: () {
        ref.read(serialPortVMProvider.notifier).okay();
      },
      style: ElevatedButton.styleFrom(
        elevation: 2, // 약간의 그림자
        backgroundColor: AppColors.PRIMARY
      ),
      child: Container(
          width: 331, height: 91, alignment: Alignment.center,
          child: Image.asset("${AppStrings.assetPath}img_check.png", width: 60, height: 60, color: Colors.white,)
      )
    );
  }

  _retryButton() {
    return ElevatedButton(
      onPressed: () {
```

```
        ref.read(serialPortVMProvider.notifier).recheck();
      },
      style: ElevatedButton.styleFrom(
        elevation: 2, // 약간의 그림자
          backgroundColor: AppColors.FF848282
      ),
      child: Container(
        width: 331, height: 91, alignment: Alignment.center,
          child: Image.asset("${AppStrings.assetPath}img_redo.png", width: 60, height: 60, color: Colors.white,)
      )
    );
  }

  Widget defaultBody() {
    return Container(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text("UCO: ${oil}g", style: AppStyles.tsStep4,),
          Text("Water: ${water}g", style: AppStyles.tsStep4,),
          SizedBox(height: 146,),
          Container(
            height: 91,
            child: Center(
              child: Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  _retryButton(),
                  SizedBox(width: 101,),
                  _checkButton(),
                ],
              ),
            ),
          )
        ],
      ),

    );
  }

}
```

Step4Screen 구조 분석

Step4Screen은 유저가 투입한 **폐식용유(oil)**와 **불순물/수분(water)** 값을 확인하고
USB/Serial 장치 상태를 기반으로 **Check / Retry**를 수행하는 최종 단계 화면이다.

---

1. Step4Screen 역할 개요

Step4Screen은 4단계 중 마지막 단계이며 다음 기능을 담당한다.

- StepNavigator 4/4를 표시한다.
- USB Serial 장치의 상태를 모니터링한다.
- 장치 상태 UIState에 따라 화면을 Blur + 로딩 애니메이션으로 전환한다.
- oil/water 데이터를 받아 사용자에게 표시한다.
- Check / Retry 버튼을 통해 ViewModel(serialPortVMProvider)로 명령을 전달한다.
- 완료(Checked) 또는 Retry 상태가 발생하면 Step1로 이동한다.

---

2. 전체 구조 흐름

```
Step4Screen (ConsumerStatefulWidget)
 ├── initState() → serialPortVM.initPortState()
 ├── didUpdateWidget() → oil/water 변경 반영
 ├── build()
 |     ├── HeaderWidget
 |     └── _body()
 |           ├── StepNavWidget(4/4)
 |           └── 상태별 UI 렌더링
 |                ├── init        → defaultBody()
 |                ├── connected   → defaultBody()
 |                ├── loading     → Blur + Spinner + underlying defaultBody()
 |                ├── error       → defaultBody()
 └── afterLayout() (작성되었으나 build 내부에서 호출되지 않음)
```

---

3. initState() 분석

USB Serial 초기화 호출

```
ref.watch(serialPortVMProvider.notifier).initPortState();
```

- USB Serial 장치의 상태를 초기화하고 최초 상태를 로드한다.
- Step4에서 가장 중요한 lifecycle 진입점이다.
- WidgetsBinding.postFrameCallback 내부에서 호출되므로 안전하다.

water, oil 값 반영

```
water = widget.water;
oil = widget.oil;
```

Step3 → Step4 이동 시 전달된 값을 화면에 표시한다.

---

4. didUpdateWidget() 분석

```
if(widget.oil != oldWidget.oil || widget.water != oldWidget.water) {
  setState(() {
    oil = widget.oil;
    water = widget.water;
  });
}
```

- 부모에서 값이 변경되면 자동으로 화면에 반영.
- 올바른 usage.

---

5. build() 분석

provider watch

```
final notifier = ref.watch(step4Provider.notifier);
final state = ref.watch(step4Provider);
```

step4Provider는 UIStateStep4를 제어하지만
실제 UI는 serialPortVMProvider의 상태(stateSerial)를 더 많이 사용한다.

Scaffold 구성

```
Column
 ├── HeaderWidget()
 └── Expanded(_body())
```

---

6. _body() 상세 분석

Serial Port 상태 감시

```
final stateSerial = ref.watch(serialPortVMProvider);
```

상태 분기

```
stateSerial.when(
  init:      → defaultBody()
  connected: → defaultBody()
  loading:   → defaultBody() + Blur + ProgressIndicator
  error:     → defaultBody()
)
```

즉 Serial 장치가 Busy/loading 상태일 때만 다음 UI가 active된다:

- 기본 화면(defaultBody) 위에

BackdropFilter(sigmaX=5, sigmaY=5)로 블러 효과
- Overlay에 `CircularProgressIndicator`

이는 Step4가 장치와 통신 중임을 표현하는 UX다.

---

7. defaultBody() 구성 분석

구성 요소

1. oil/water 수치 표시
2. Retry / Check 버튼

레이아웃

```
Column (Center)
 ├── "UCO: Xg"
 ├── "Water: Yg"
 ├── Spacer
 └── Row
      ├── retryButton
      └── checkButton
```

이 버튼들은 serialPortVMProvider의 기능을 수행한다.

---

8. Check / Retry UI & 상태전이

Check

```
ref.read(serialPortVMProvider.notifier).okay();
```

→ USB 상태 검증 완료 요청
→ 성공 시 UIStateStep4Checked emit
→ Step1으로 이동

Retry

```
ref.read(serialPortVMProvider.notifier).recheck();
```

→ USB 재검증 요청
→ UIStateStep4Retry emit
→ Step1로 이동

---

9. afterLayout() 문제점 분석

코드는 존재하지만 호출되지 않는다.

```
void afterLayout() {
  ref.listenManual(step4Provider, (_, state) {
    if(state is UIStateStep4Checked) {
      context.goNamed(RouteGroup.Step1.name);
    }
    if(state is UIStateStep4Retry) {
      context.goNamed(RouteGroup.Step1.name);
    }
  });
}
```

→ initState()에서 호출되지 않음
→ Step4Provider 상태 변화는 Step4Screen이 감지하지 못함
→ navigation 기능은 사실상 동작하지 않는 상태

이는 설계 상 **중요한 버그**다.

---

10. 구조적 개선 포인트

아래는 Step4 개선 시 가장 중요하게 고쳐야 하는 항목들이다.

1) afterLayout() 반드시 initState()에서 호출해야 한다.

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

Step3에서와 동일한 구조를 유지해야 한다.

---

2) notifier/read/watch 혼합 사용 최적화

현재:

```
ref.watch(step4Provider.notifier);
```

→ watch(notifier)는 rebuild를 유발할 수 있음
→ ref.read()가 의도에 맞다.

---

3) defaultBody()는 const 사용 불가 → OK

동적 oil/water 값 사용하므로 문제 없음.

---

4) loading 상태 UI: BackdropFilter 블러는 비용이 큼 → AnimatedOpacity 가능

옵션: AnimatedContainer + Color overlay 사용이 훨씬 가볍다.

---

5) StepNavWidget 배치 규칙 Step1~Step3과 동일하게 조정 필요

Step1~Step3은 `Row → StepNav + Expanded(body)` 구조.
Step4도 동일하게 맞추고 있으므로 일관성 유지됨.

---

11. Step4 상태 흐름 정리 (State Machine)

```
┌──────────┐
|  init    |
└────┬─────┘
     | (USB Connected)
     ▼
┌──────────────┐
| connected    |
└────┬─────────┘
     | (User: Check)
     ▼
┌──────────────┐
|  loading     |
└────┬─────────┘
     | OK
     ▼
┌──────────────┐
|   checked    | → Step1로 이동
└──────────────┘

User: Retry → loading → retry → Step1 이동
```

---

12. 추가 요청 시 제공 가능한 것들

요청하면 아래 내용을 **정제된 공식 문서 스타일로 제공**할 수 있다.

▷ Step4 전체 리팩토링 코드 (버그 제거 + 구조 개선)

▷ Step4 ViewModel(vm_step4.dart) 분석

▷ USB Serial 구조 흐름도

▷ Step1~Step4 전체 Wizard 아키텍처 다이어그램

▷ UI 개선 버전 (Material 3 적용, Blur 성능 최적화)