vm_driver.dart

소스 코드

전체 코드

```dart
import 'package:buyoil/model/ui_state_step3.dart';
import 'package:buyoil/router.dart';
import 'package:buyoil/viewmodel/vm_serial_port.dart';
import 'package:riverpod_annotation/riverpod_annotation.dart';

part 'vm_driver.g.dart';

@riverpod
class Driver extends _$Driver {
  @override
  UIStateStep3 build() {
    return UIStateStep3.init();
  }

  // [CMD]CLOSES# -> 첫페이지로 TODO 응답 받을게 있으면 처리
  void pressedClose() {
    ref.watch(serialPortVMProvider.notifier).close().whenComplete(() {
      ref.read(routerProvider).goNamed(RouteGroup.Splash.name);
    });
    // // todo ok 처리
    // state = UIStateStep3.closeDoor();
    //
    // Future.delayed(Duration(seconds: 2), () {
    //   state = UIStateStep3.completed();
    // });
  }
}
```

1️⃣ 파일 개요

파일명: `lib/viewmodel/vm_driver.dart`
클래스명: `Driver`
역할:

- **State 관리 및 MCU/시리얼 명령 처리 ViewModel**
- Riverpod 기반 상태 관리 (`@riverpod`)
- **UIStateStep3** 초기화 및 "닫기(CLOSES)" 명령 처리

사용 맥락:

- Step3 UI 화면 관련 로직
- 시리얼 포트를 통해 MCU 명령을 보내고, 완료 후 앱 내 라우팅 처리

---

2 주요 기능

1. **초기 상태 관리**
    - `build()` 메서드에서 초기 상태 반환:

```
@override
UIStateStep3 build() {
  return UIStateStep3.init();
}
```

    - `UIStateStep3`는 Step3 화면의 UI 상태 모델 (초기, 완료 등 상태 정의)
2. **닫기 버튼 처리 (`pressedClose`)**

```
void pressedClose() {
  ref.watch(serialPortVMProvider.notifier).close().whenComplete(() {
    ref.read(routerProvider).goNamed(RouteGroup.Splash.name);
  });
}
```

    - 시리얼 포트 종료(`close()`) 후 **Splash 화면**으로 라우팅
    - 주석 처리된 코드: 향후 MCU 응답 OK 처리 → UI 상태 업데이트 예정

---

3 구조 분석

```
Driver (@riverpod)
 ├─ build() -> UIStateStep3.init()
 └─ pressedClose()
       ├─ serialPortVMProvider.notifier.close()
       └─ whenComplete() -> routerProvider.goNamed(Splash)
```

- Riverpod 코드 생성: `part 'vm_driver.g.dart'` → `.g.dart` 파일 자동 생성
- `ref.watch` vs `ref.read`:
    - `ref.watch` → provider 상태 구독
    - `ref.read` → provider 접근만, 구독은 아님 (라우팅 처리 용도)

---

4 동작 흐름

1. Driver Provider 생성 → 초기 상태 `UIStateStep3.init()`
2. 사용자 "닫기" 버튼 클릭 → `pressedClose()` 호출
3. 시리얼 포트 닫기 실행 (`serialPortVMProvider.notifier.close()`)
4. 종료 완료 시 Splash 화면으로 이동 (`routerProvider.goNamed(RouteGroup.Splash.name)`)
5. 향후 MCU 응답 처리에 따라 상태 업데이트 가능 (주석 참고)

---

5 장점

- **Riverpod 패턴 활용**: 상태와 로직 분리 → 테스트 용이
- **시리얼 포트와 UI 상태 연결**: `serialPortVMProvider` 참조
- **비동기 처리 안전**: `whenComplete()` 사용 → 완료 후 라우팅

---

6 단점 / 개선점

1. **현재 상태 업데이트 미구현**
    - 주석 처리된 MCU OK 처리 코드만 존재
    - 개선: MCU 응답 상태에 따라 `state = UIStateStep3.closeDoor()` 등 상태 관리
2. **오직 닫기 기능만 존재**
    - Driver ViewModel 기능 확장 필요
    - 개선: Step3 관련 다른 버튼/명령 처리 추가 가능
3. **에러 처리 부재**
    - 시리얼 포트 닫기 실패 시 예외 처리 없음
    - 개선: try-catch 또는 `catchError`로 에러 핸들링

---

7 결론

- `vm_driver.dart`는 **Step3 화면 관련 ViewModel**으로,
  시리얼 포트 닫기 명령과 라우팅 처리에 특화된 단순 구조입니다.
- 향후 MCU 응답 처리, 상태 관리, 에러 처리 기능을 추가하면 **완전한 Step3 ViewModel**이 됩니다.