ui_state_setting.dart

소스 코드

전체 코드

```dart
import 'package:flutter/material.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

part 'ui_state_setting.freezed.dart';

@freezed
sealed class UIStateSetting with _$UIStateSetting {

  const factory UIStateSetting.init({
    @Default(0) int oilContainer,
    @Default(0) double measureOil,
    @Default(0) double measureWater,
    @Default(false) bool isMotorOn,
    @Default(false) bool isValveOn,
  }) = UIStateSettingInit;
}
```

1️⃣ 파일 개요

파일명: `lib/view/widget/step1/custom_test_input.dart`
역할: Flutter에서 사용하는 **커스텀 텍스트 입력 위젯** 정의
주요 목적:

- 고정 크기와 스타일을 가진 **숫자 입력 필드(TextField)** 제공
- 부모 위젯에서 전달한 초기 텍스트를 표시
- 현재는 **읽기 전용**으로, 사용자 입력은 막혀 있음
- 입력 제한(숫자만, 최대 11자리)을 적용할 수 있는 구조

사용 맥락 예시:

- 전화번호, 주민등록번호, 숫자 기반 ID 등 읽기/표시 전용 입력 필드
- UI 스타일 통일과 상태 관리 용이

2️⃣ 주요 기능

1. **텍스트 초기화 및 동기화**
    - 부모 위젯에서 전달한 `text` 값을 `TextEditingController`에 초기화
    - 부모에서 `text` 변경 시 자동 갱신
2. **입력 제한**
    - 숫자만 허용 (`FilteringTextInputFormatter.digitsOnly`)

- 최대 11자리 제한 (`LengthLimitingTextInputFormatter`)
- 현재 `enabled: false` 상태라 제한은 실제로 적용되지 않음

3. **UI 스타일**
   - 고정 크기: `width: 370, height: 70`
   - 라운드 테두리 적용(`borderRadius: 14`)
   - 테두리 색상 및 두께 통일(`AppColors.PRIMARY`, width: 1)
   - 커서 색상과 텍스트 스타일 적용 (`AppStyles.tsTextField`)
4. **상태 관리**
   - `TextEditingController`를 사용하여 텍스트 상태 관리
   - `initState` → 초기화, `didUpdateWidget` → 값 변경 시 갱신, `dispose` → 메모리 해제

---

3 구조 분석

```
CustomTextInputField (StatefulWidget)
 └── TextInputFieldState (State)
       ├─ controller: TextEditingController
       ├─ initState(): controller 초기화
       ├─ didUpdateWidget(): 부모 텍스트 변경 시 갱신
       ├─ dispose(): controller 해제
       └─ build(): TextField 위젯 생성
              ├─ SizedBox (370x70)
              ├─ TextField
              |    ├─ controller
              |    ├─ style, cursorColor
              |    ├─ enabled: false
              |    ├─ inputFormatters (digitsOnly, length 11)
              |    └─ InputDecoration (border, padding)
```

특징:

- 상태 관리가 깔끔하게 분리됨
- UI 속성과 로직이 `build()`에 집중되어 있음
- 입력 필드가 읽기 전용이라 현재는 데이터 표시용

---

4 동작 흐름

1. 부모 위젯에서 `CustomTextInputField(text: "01012345678")` 생성
2. `TextInputFieldState`의 `initState`에서 `controller` 초기화
3. 화면 빌드 시 `TextField`가 controller의 값으로 초기화
4. 부모가 `text` 값을 변경하면 `didUpdateWidget`이 호출되어 `controller.text` 갱신
5. 사용자는 입력 불가(`enabled: false`)
6. 메모리 해제 시 `dispose`에서 controller 해제

---

5 장점

- **부모-자식 데이터 동기화** 잘 처리됨
- **UI 일관성**: 테두리, 색상, 스타일 통일
- **메모리 안전**: controller dispose 처리
- **입력 제한 기능 포함**: 나중에 활성화 가능

---

6 단점 / 개선점

1. **읽기 전용 상태(`enabled: false`)**
    - 현재 숫자 제한과 입력 포맷터가 의미 없음
    - 개선: 필요 시 활성화 가능하도록 옵션화 (`enabled`를 매개변수로)
2. **고정 크기**
    - `SizedBox(width: 370, height: 70)` → 화면 크기나 반응형 디자인에 취약
    - 개선: `width: double.infinity` 또는 부모 constraints 기반
3. **재사용성**
    - 현재 `text`와 `enabled` 외에는 옵션이 없음
    - 개선: placeholder, maxLength, inputType, 스타일 등 매개변수 추가
4. **테스트 / 검증**
    - 유닛 테스트나 위젯 테스트가 없음
    - 개선: 텍스트 변경, 포맷터, UI 상태 테스트 작성
5. **기능 확장**
    - 실시간 입력 시 `onChanged` 콜백 없음
    - 개선: 콜백을 추가하여 부모에서 입력 반응 가능

---

7 개선된 구조 예시 (개념)

```
CustomTextInputField(
  text: "01012345678",
  enabled: true,
  maxLength: 11,
  keyboardType: TextInputType.number,
  onChanged: (value) { ... },
)
```

- 이렇게 하면 읽기/쓰기 모드 전환, 입력 제한, 콜백 모두 지원 가능
- 반응형 화면에도 대응 가능