1. Architecture

1. 이 프로젝트에서 사용된 아키텍처 개요

소스 트리와 패키지 구성으로 판단했을 때, 본 프로젝트는 **Flutter + Riverpod + GoRouter 기반의 MVVM 아키텍처**를 사용하고 있다.
특징을 정리하면 다음과 같다.

---

1) Presentation / UI Layer

경로:
`lib/view/screen/*`
`lib/view/widget/*`

구성 요소:

- UI(View)
- StatelessWidget, StatefulWidget
- 사용자 입력, 화면 구성
- ViewModel의 state를 구독하여 화면 반영

UI 레이어는 **상태를 직접 관리하지 않고**, ViewModel이 제공하는 값만 사용한다.
GoRouter 기반의 화면 이동도 이 레이어에서 이루어진다.

---

2) ViewModel Layer (State Management Layer)

경로:
`lib/viewmodel/*`

사용 기술:

- Riverpod (Provider)
- code generation (`router.g.dart` 생성 기반으로 추정 → `riverpod_generator` 사용)

역할:

- 비즈니스 로직 처리
- 서버 API 또는 로컬 데이터 접근
- Validation
- 사용자 이벤트 처리
- 상태(state) 생성 및 변경

View(UI)는 ViewModel에 이벤트만 요청하고
ViewModel이 상태를 갱신하며
Riverpod Provider가 이를 UI에 전달하는 구조다.

---

3) Model Layer

경로:
`lib/model/*`

역할:

- DTO
- Domain Model
- API response / request 구조 정의
- 데이터 변환 및 직렬화

ViewModel과 Data Source 사이의 데이터 전용 구조체로 사용된다.

---

4) Common / Core Layer

경로:
`lib/common`
`lib/common/utils`
`lib/common/extension` 등

역할:

- 공통 유틸리티
- Toast, Logger
- 상수, 타입 정의
- Error handler
- 확장 모듈(extension)

모든 레이어에서 참조 가능한 공용 범용 모듈이다.

---

5) Routing Layer

경로:
`lib/router.dart`
`lib/router.g.dart`

구성:

- Declarative Routing (GoRouter)
- Provider로 Router를 주입하는 구조

`router.g.dart`는 riverpod_generator가 만든 자동 생성 파일로
라우터 Provider 정의 코드가 포함되어 있다.

---

6) Config Layer

경로:
`lib/config.dart`

역할:

- 환경 설정(Debug 여부 등)
- 글로벌 설정값 관리
- 싱글톤 형태로 앱 전체에서 접근 가능

애플리케이션 환경을 하나의 Config 객체로 관리하도록 설계되어 있다.

---

7) Entry Point Layer

경로:
`lib/main.dart`

역할:

- runApp
- 애플리케이션 초기화
- ProviderScope 초기화
- Locale, Theme, Orientation 설정
- Router 초기 바인딩

애플리케이션의 전체 구조가 출발하는 가장 중요한 파일이다.

---

2. 아키텍처 계층도

아래는 해당 프로젝트가 실제로 어떤 방식으로 계층 분리되어 있는지를 나타낸 ASCII 아키텍처 다이어그램이다.

```
┌──────────────────────────────────────────┐
│                 main.dart                │
│       - App 초기화 / ProviderScope       │
│       - Router 바인딩 / Locale 설정       │
└──────────────────────────────────────────┘
                    |
                    ▼
┌──────────────────────────────────────────┐
│               Presentation(UI)           │
│    lib/view/screen/*  lib/view/widget/*  │
│ - View: 화면 구성                         │
│ - 사용자 입력 처리 → ViewModel 요청        │
│ - Provider 통해 상태 구독                 │
└──────────────────────────────────────────┘
                    |
                    ▼
┌──────────────────────────────────────────┐
```

```
|                 ViewModel                  |
|               lib/viewmodel/*              |
| - 비즈니스 로직                              |
| - 상태(State) 관리                          |
| - Provider로 UI에 상태 공개                  |
└──────────────────────────────────────────┘
                     |
                     ▼
┌──────────────────────────────────────────┐
|                  Model                     |
|               lib/model/*                  |
| - DTO / Domain Model                       |
| - JSON 변환 / 데이터 구조체                  |
└──────────────────────────────────────────┘
                     |
                     ▼
┌──────────────────────────────────────────┐
|         Common / Utils / Config            |
|       lib/common/*   lib/config.dart       |
| - Logger, Toast, Exception Handler         |
| - App 설정 정보                             |
| - 공통적으로 사용되는 유틸 함수              |
└──────────────────────────────────────────┘
                     |
                     ▼
┌──────────────────────────────────────────┐
|               Routing Layer                |
|          router.dart / router.g.dart       |
| - GoRouter route 정의                      |
| - Riverpod Provider 기반 라우팅             |
└──────────────────────────────────────────┘
```

---

3. 이 프로젝트 코드를 분석할 때의 최적 순서

단순히 파일 열어보는 순서가 아니라,
"구조를 이해하고 로직을 파악할 수 있는 가장 효율적인 흐름"으로 정리했다.

---

1) `main.dart`

분석 포인트:

- 앱 초기화 순서
- Router 연결
- ProviderScope 생성
- Locale 설정
- Orientation 또는 Theme 설정

앱 전체 구조를 빠르게 파악할 수 있다.

---

2) `router.dart`

분석 포인트:

- 어떤 화면이 있는지
- 초기 라우트는 무엇인지
- 화면 간 이동 구조
- 로그인 여부에 따른 guard 존재 여부

GoRouter 기반 앱의 전체 흐름을 가장 빠르게 이해할 수 있다.

---

3) 디렉터리 구조 전체 훑어보기

특히 다음을 반드시 확인:

- `/view/screen` → 앱 기능 구성
- `/viewmodel` → 상태관리
- `/model` → 데이터 구조
- `/common` → 공통 로직

이 단계에서 "앱이 어떤 기능을 제공하는지"가 윤곽이 나온다.

---

4) `pubspec.yaml`

분석 포인트:

- 사용된 라이브러리
- build_runner 사용 여부
- asset 경로
- localization (easy_localization) 설정

앱의 기술 스택이 모두 나온다.

---

5) Config

```
lib/config.dart
```

- debug 여부
- 환경 분기 처리
- 글로벌 전역 설정 흐름 이해

---

6) ViewModel

앱 로직 핵심이 여기에 있다.

분석 포인트:

- Provider 생성 구조
- State 구조
- UI와의 연결 방식
- API나 내부 데이터 처리

---

7) View(Screen/UI)

ViewModel과 연결하여 화면 흐름을 이해한다.

분석 포인트:

- ViewModel 데이터 사용하는 방식
- 사용자 입력이 어떤 method로 전달되는지
- route → screen → viewmodel 흐름 확인

---

4. 결론 — 이 프로젝트는 어떤 구조인가?

본 프로젝트는 다음을 기반으로 설계된 **표준적인 현대 Flutter 아키텍처**이다.

1. **GoRouter 기반의 Declarative Routing**
2. **Riverpod 기반 상태관리 (Provider 자동 생성)**
3. **MVVM 구조 (View ↔ ViewModel ↔ Model)**
4. **공통 유틸리티와 설정 분리 (common, config)**
5. `build_runner` 기반 generator 활용 (`router.g.dart`)

구조가 명확하게 나뉘어 있어 유지보수, 확장에 적합하다.