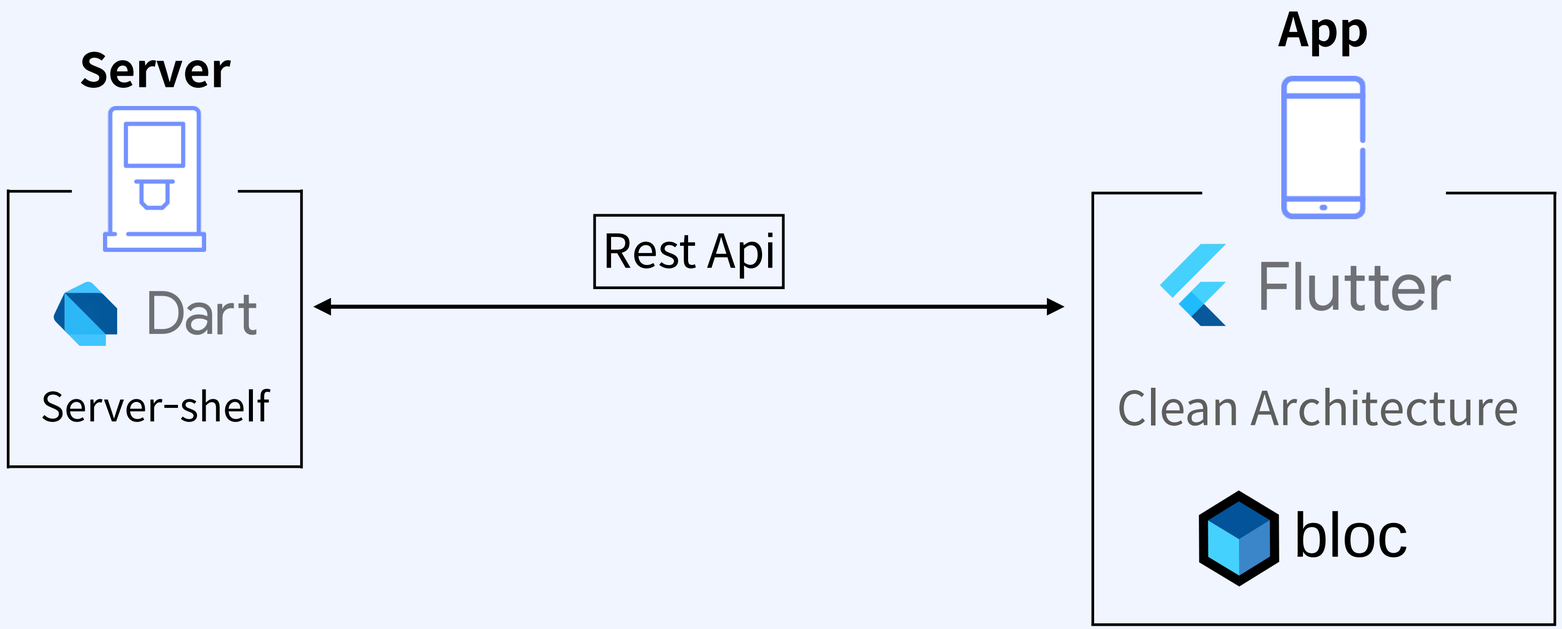
커머스 앱 - 기획 단계

5 대략적인 아키텍쳐 설계

5.
대략적인 아키텍쳐 설계



프로젝트 구조 - Core

```
├── core
│   ├── theme
│   └── utils
├── data
│   ├── data_source
│   ├── dto
│   ├── entity
│   ├── mapper
│   └── repository_impl
├── dependency_injection.config.dart
├── dependency_injection.dart
├── domain
│   ├── model
│   ├── repository
│   └── usecase
├── main.dart
└── presentation
    ├── main
    ├── pages
    └── routes
```

Core

프로젝트 전체에서 공통적으로 쓰이는 디렉토리

```
├── theme
│   ├── constant
│   ├── custom
│   └── theme_data.dart
└── utils
    ├── component
    ├── constant.dart
    ├── dialog
    ├── error
    ├── exception
    ├── extensions.dart
    ├── logger.dart
    └── rest_client
```

프로젝트 구조 - Data_layer

```
├── core
│   ├── theme
│   └── utils
├── data
│   ├── data_source
│   ├── dto
│   ├── entity
│   ├── mapper
│   └── repository_impl
├── dependency_injection.config.dart
├── dependency_injection.dart
├── domain
│   ├── model
│   ├── repository
│   └── usecase
├── main.dart
└── presentation
    ├── main
    ├── pages
    └── routes
```

Data_layer

시스템의 데이터 엑세스와 관련된 부분을 담당

```
data
├── data_source
│   ├── data_source_module.dart
│   ├── local_storage
│   ├── mock
│   └── remote
├── dto
│   ├── common
│   └── display
├── entity
│   └── display
├── mapper
│   ├── common.mapper.dart
│   └── display.mapper.dart
└── repository_impl
    └── display.repository_impl.dart
```

프로젝트 구조 - Domain_layer

```
├── core
│   ├── theme
│   └── utils
├── data
│   ├── data_source
│   ├── dto
│   ├── entity
│   ├── mapper
│   └── repository_impl
├── dependency_injection.config.dart
├── dependency_injection.dart
├── domain
│   ├── model
│   ├── repository
│   └── usecase
├── main.dart
└── presentation
    ├── main
    ├── pages
    └── routes
```

Domain_layer

시스템의 핵심 로직을 구현하고 유지관리

```
domain
├── model
│   ├── common
│   └── display
├── repository
│   ├── display.repository.dart
│   └── repository.dart
└── usecase
    ├── base
    └── display
```

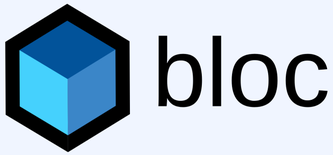
프로젝트 구조 - Presentation_layer

```
├── core
│   ├── theme
│   └── utils
├── data
│   ├── data_source
│   ├── dto
│   ├── entity
│   ├── mapper
│   └── repository_impl
├── dependency_injection.config.dart
├── dependency_injection.dart
├── domain
│   ├── model
│   ├── repository
│   └── usecase
├── main.dart
└── presentation
    ├── main
    ├── pages
    └── routes
```

Presentation_layer

사용자 인터페이스과 상호작용하는 부분

```
presentation
├── main
│   ├── bloc
│   ├── component
│   ├── cubit
│   ├── main_screen.dart
│   └── utils
├── pages
│   ├── cart_list
│   ├── category
│   ├── home
│   ├── search
│   ├── splash
│   └── user
└── routes
    ├── route_path.dart
    └── routes.dart
```



"Business Logic Component"의 약자로 Flutter Application에서 상태를 관리하는 패턴

1. 분리된 비즈니스 로직
2. 뛰어난 재사용성
3. 테스트 용이성
4. 상태 변화 추적
5. 뛰어난 팀 협업