MVI

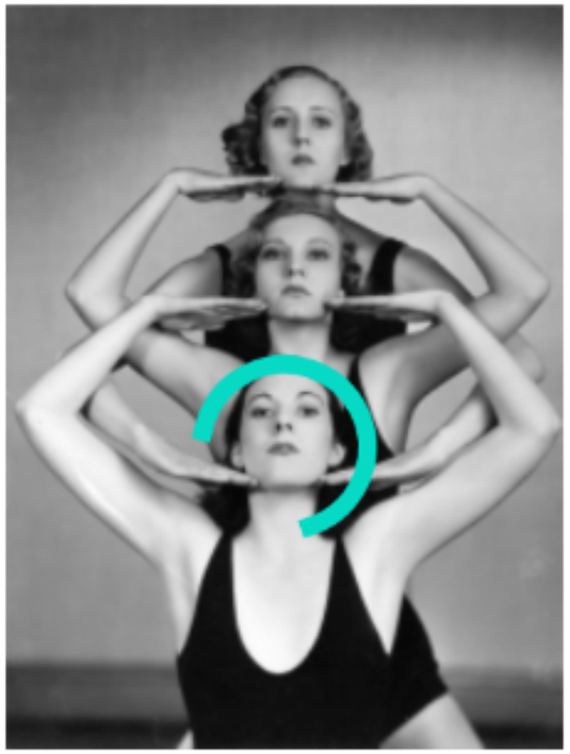
Model - View - Intent

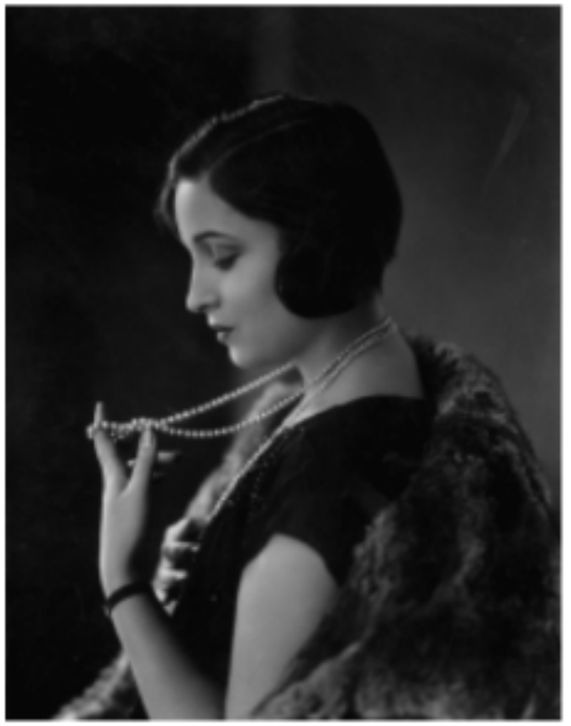
2107 김준호

사용하는 이유

상태 문제

- 앱은 프로그레스바의 노출 상태, 버튼 활성화 여부 등 많은 상태들로 구성되어 있는데 이때 이 상태들을 관리하기 힘들고 의도하지 않은 방향으로 제어가 된다면, 우리는 이것을 상태 문제라고 한다.

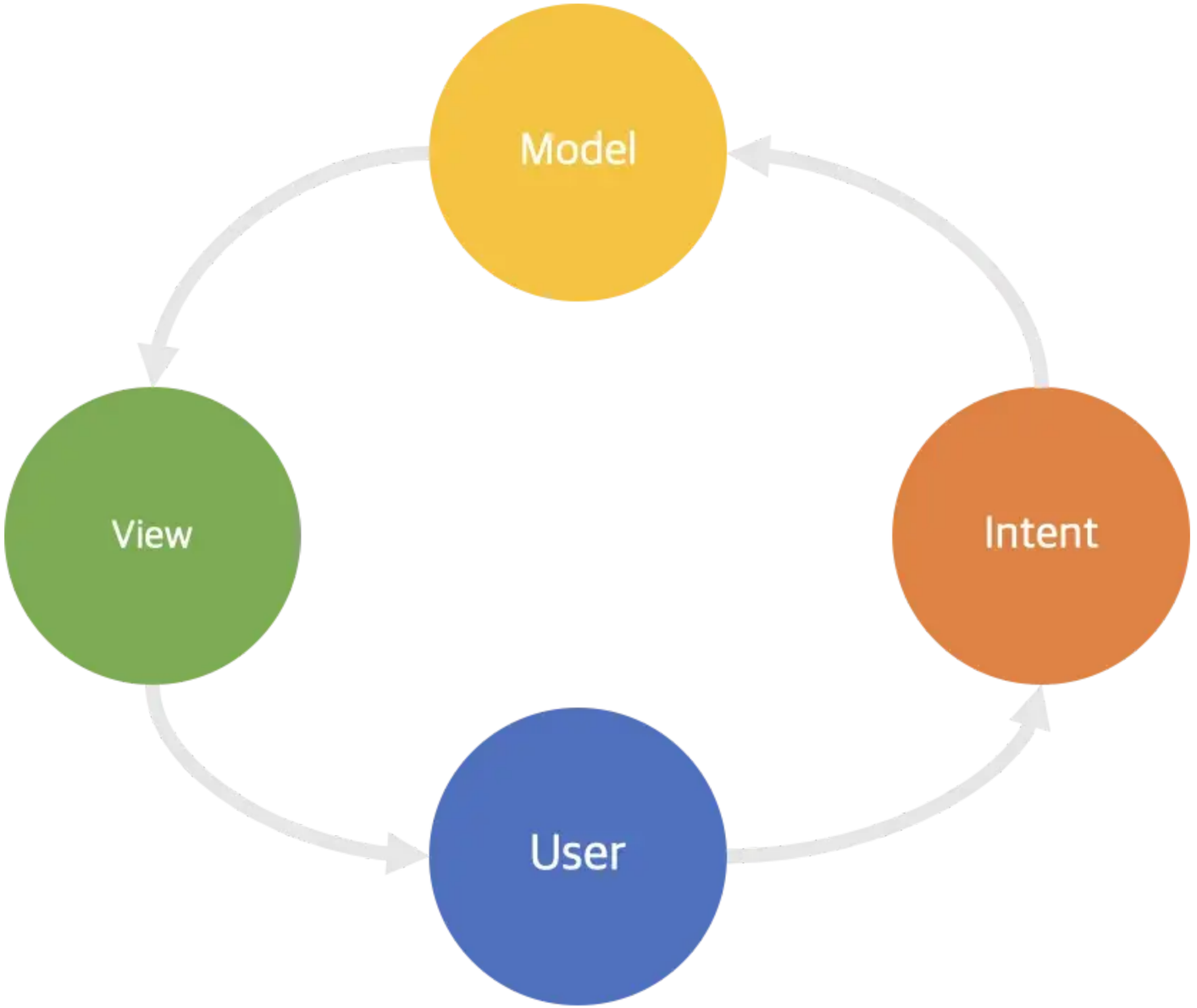


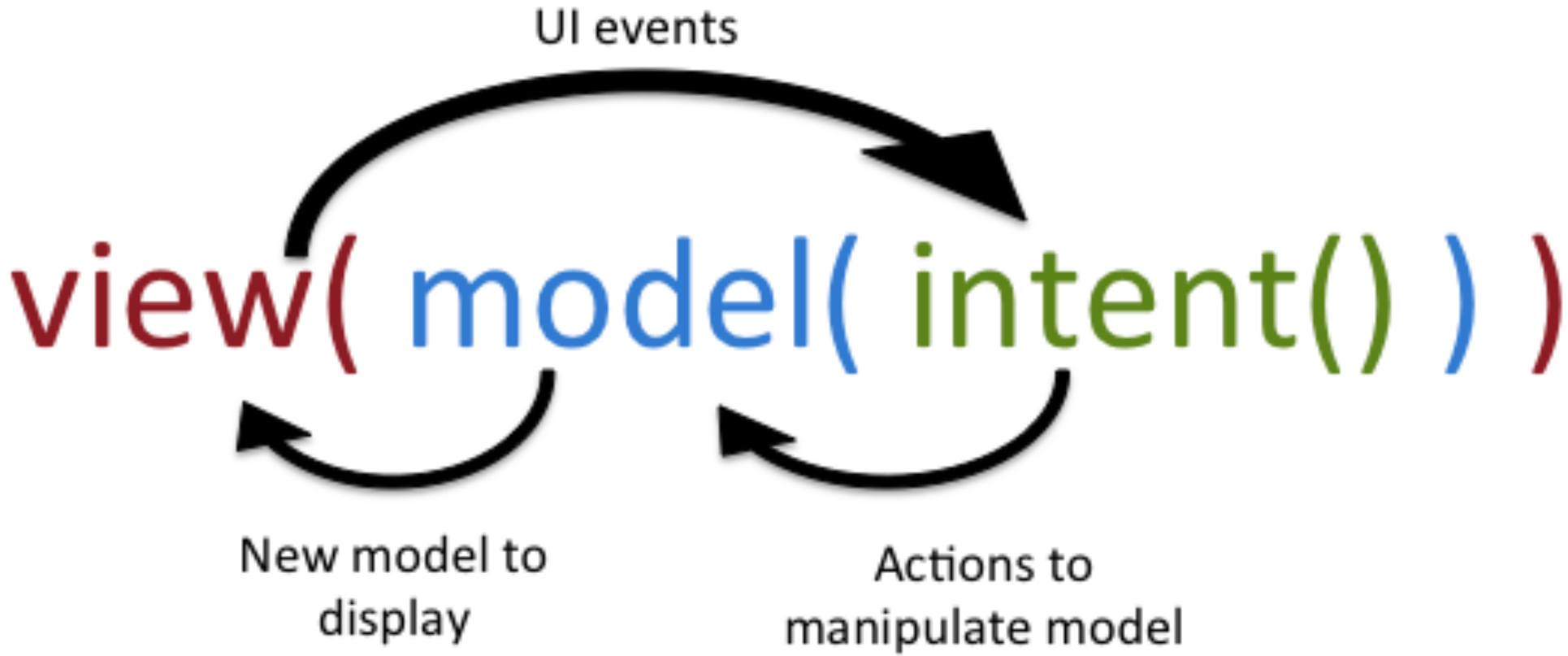


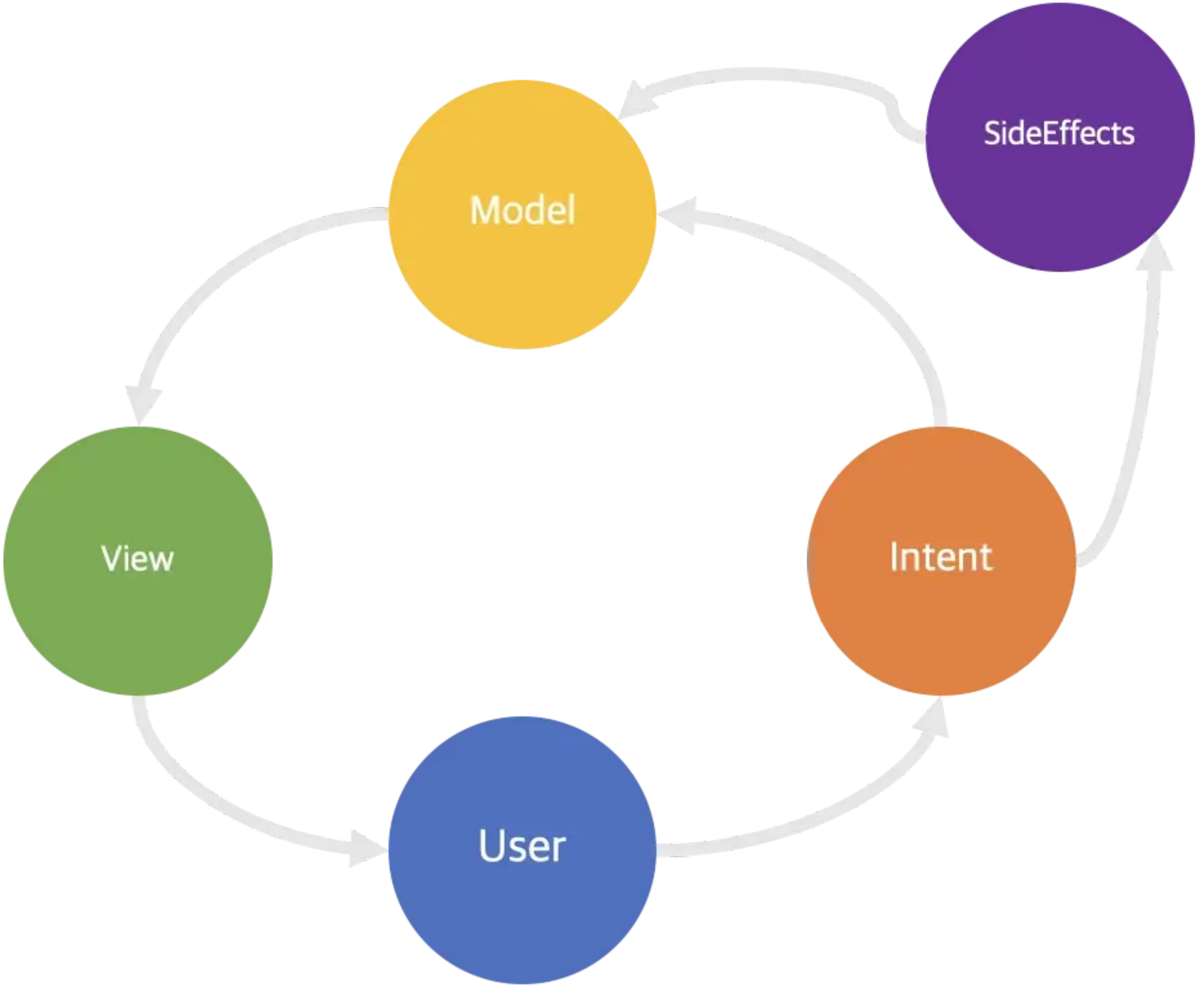
사용하는 이유

부수 효과(Side Effect)

- 부수 효과란 원래의 목적과 다르게 다른 효과 또는 부작용이 나는 상태를 지칭한다
- 안드로이드 같은 경우에는 서버 호출, 데이터베이스 입출력 등 어떤 결과를 얻을 지 예상할 수 없으며 그에 따라 상태 관리에 대한 어려움을 겪게 된다.


Model
View
Intent
User


UI events
view(model(intent()))
New model to display
Actions to manipulate model


Model
SideEffects
View
Intent
User

View

받은 데이터가 우리에게 보여지는 화면

Model

모델은 앱의 유일한 상태와 데이터를 갖고 있는 불변 객체이다.

- 불변이기 때문에 모델을 업데이트 시키려면 새로 만들어야 한다.
- 불변객체는 생성 후 그 상태를 바꿀 수 없는 객체를 말함

```kotlin
data class SignInState(
    val employeeNumber: String = "",
    val password: String = "",

    val errMsgEmployeeNumber: String ? = null,
    val errMsgPassword: String ? = null,
)

sealed class SignInSideEffect {

    object NavigateToHomeScreen : SignInSideEffect()

    object IdOrPasswordNotCorrect : SignInSideEffect()

    object NumberFormat : SignInSideEffect()
}
```

← Model(모델)

← Side Effect
(부수 효과)

```kotlin
fun inputEmployeeNumber(employeeNumber: String) = intent { this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(employeeNumber = employeeNumber) }
}

fun inputPassword(msg: String) = intent { this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(password = msg) }
}

fun inputErrMsgEmployeeNumber(msg: String?) = intent { this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(errMsgEmployeeNumber = msg) }
}

fun inputErrMsgPassword(msg: String?) = intent { this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(errMsgPassword = msg) }
}
```

```
SimTongTextField(
    value = signInState.employeeNumber,
    onValueChange = { it: String
        vm.inputEmployeeNumber(it)
    },
    hintBackgroundColor = SimTongColor.Gray100,
    backgroundColor = SimTongColor.Gray50,
    hint = stringResource(id = "사원번호"),
    keyboardType = KeyboardType.Number,
    error = signInState.errMsgEmployeeNumber,
)
```

Intent

앱의 상태를 바꾸라는 요청

- 우리가 알고 있는 android.content.Intent가 아니다.
- Intent()가 실행되면 새로운 상태를 가진 모델을 받게 된다.
- 앱의 상태 변화에 대해, 행동을 취하는 방법으로서 표현되며 Event와 비슷한 의미이다.
- Model은 이 인텐트를 통해서만 업데이트 받을 수 있다

```kotlin
fun inputEmployeeNumber(employeeNumber: String) = intent {  this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(employeeNumber = employeeNumber) }
}

fun inputPassword(msg: String) = intent {  this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(password = msg) }
}

fun inputErrMsgEmployeeNumber(msg: String?) = intent {  this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(errMsgEmployeeNumber = msg) }
}

fun inputErrMsgPassword(msg: String?) = intent {  this: SimpleSyntax<SignInState, SignInSideEffect>
    reduce { state.copy(errMsgPassword = msg) }
}
```

StateReducer

- model은 불변성을 가지고 있기 때문에 쉽게 변경할 수 없다
- 앱의 새로운 상태를 나타내는 ParticalState 라는 새로운 상태를 만든다
- 이전 상태와 ParticalState를 merge(병합)하여 화면에 표시될 새로운 상태가 된다

Side Effect

- 부수 효과
- postSideEffect()를 사용

```
signInUseCase(
    params = SignInUseCase.Params(
        employeeNumber = employeeNumber.toInt(),
        password = password,
    ),
).onSuccess {
    postSideEffect(SignInSideEffect.NavigateToHomeScreen)
}.onFailure { it: Throwable
    when (it) {
        is UnAuthorizedException -> postSideEffect(SignInSideEffect.IdOrPasswordNotCorrect)
        is NotFoundException -> postSideEffect(SignInSideEffect.IdOrPasswordNotCorrect)
        else -> throwUnknownException(it)
    }
}
```

```
data class SignInState(
    val employeeNumber: String = "",
    val password: String = "",

    val errMsgEmployeeNumber: String ? = null,
    val errMsgPassword: String ? = null,
)

sealed class SignInSideEffect {

    object NavigateToHomeScreen : SignInSideEffect()

    object IdOrPasswordNotCorrect : SignInSideEffect()

    object NumberFormat : SignInSideEffect()
}
```

```
signInSideEffect.observeWithLifecycle { it: SignInSideEffect
    when (it) {
        SignInSideEffect.NavigateToHomeScreen -> {
            navController.navigate(
                route = SimTongScreen.Home.MAIN
            ) { this: NavOptionsBuilder
                popUpTo( id: 0)
            }
        }
        SignInSideEffect.IdOrPasswordNotCorrect -> {
            vm.inputErrMsgPassword(
                msg = ErrMsgIdOrPasswordNotCorrect,
            )
        }
        SignInSideEffect.NumberFormat -> {
            vm.inputErrMsgEmployeeNumber(
                msg = ErrMsgNumberFormat,
            )
        }
    }
}
```

+ Response 값이 있는 경우

```
fetchUserInformationUseCase()
    .onSuccess { it: User
        reduce { this: SimpleContext<MyPageState>
            state.copy(
                name = it.name,
                nickname = it.nickname,
                email = it.email,
                spot = it.spot,
                profileImagePath = it.profileImagePath,
            )
        }
    }
```

장점

- 앱의 상태가 하나 뿐이라, 상호작용이 많아져도 상태 충돌이 없다.
- 데이터의 흐름이 정해져 있어 흐름을 이해하고 관리하기가 쉽다.
- 각각 값이 불변하기 때문에, 스레드에 있어 안정성을 갖는다.

단점

- 다른 MV 형재들에 비해 러닝커브가 높다.
- 작은 변경도 모두 intent를 통해야 하고, 아주 작은 앱도 최소한의 intent와 model를 가져야 한다.
- model를 업데이트 하기 위해 매번 새로운 인스턴스를 만들어야 하므로, 너무 많은 객체가 만들어지면 gc가 빈번하게 작동될 수 있다.(리소스 낭비)

개인적인 생각