

프로토콜 지향 프로그래밍

도메인 모델 설계

2025-05-01 THU
컴퓨터공학과 박효준

학습 목표

- 개발을 할 때, 기획과 디자인에 맞는 도메인 모델을 설계할 수 있다.
- 기획 혹은 디자인 팀의 요구사항이 추가/변경 되어도,
변화에 잘 대응할 수 있는 도메인 모델을 설계할 수 있다.
- 디자인을 보고 도메인 모델들 간의 공통되는 부분을 추출하여 추상화하고,
이를 통해 프로토콜 지향 프로그래밍을 할 수 있다.

프로토콜 지향 프로그래밍이란

- Protocol-Oriented Programming (= POP)
- POP = OOP + FP + Value Semantic
- Swift에는 값 타입과 참조 타입이 모두 존재
 - 값 타입은 C언어의 구조체라고 생각 == 빠름, 복사, 메모리 효율성 좋음
 - 참조 타입은 자바의 클래스라고 생각 == 느림, 참조(공유), OOP 특징 활용 가능
- 프로토콜 (= 인터페이스)
 - 자바, C# 등의 언어에서 인터페이스는 클래스만 따르는 것이 가능
 - Swift는 구조체나 열거형도 OOP 처럼 사용 가능함

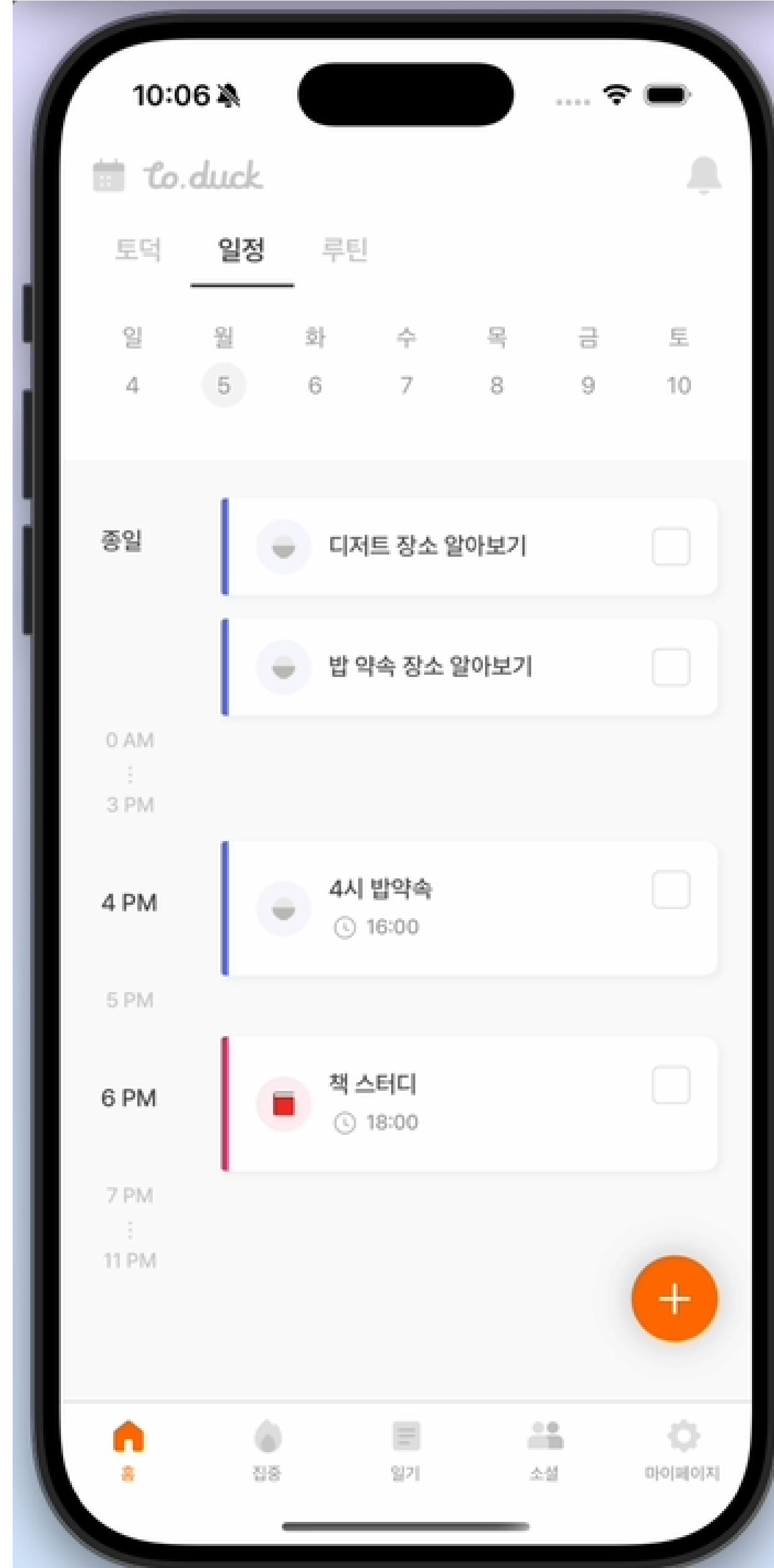
프로토콜 지향 프로그래밍이란

- Protocol-Oriented Programming (= POP)
- $POP = OOP + FP + \text{Value Semantic}$
- Swift에는 값 타입과 참조 타입이 모두 존재
 - 값 타입은 C언어의 구조체라고 생각 == 빠름, 복사, 메모리 효율성 좋음
 - 참조 타입은 자바의 클래스라고 생각 == 느림, 참조(공유), OOP 특징 활용 가능
- 프로토콜 (= 인터페이스)
 - 자바, C# 등의 언어에서 인터페이스는 클래스만 따르는 것이 가능
 - Swift는 구조체나 열거형도 OOP 처럼 사용 가능함

기획 명세서

일정

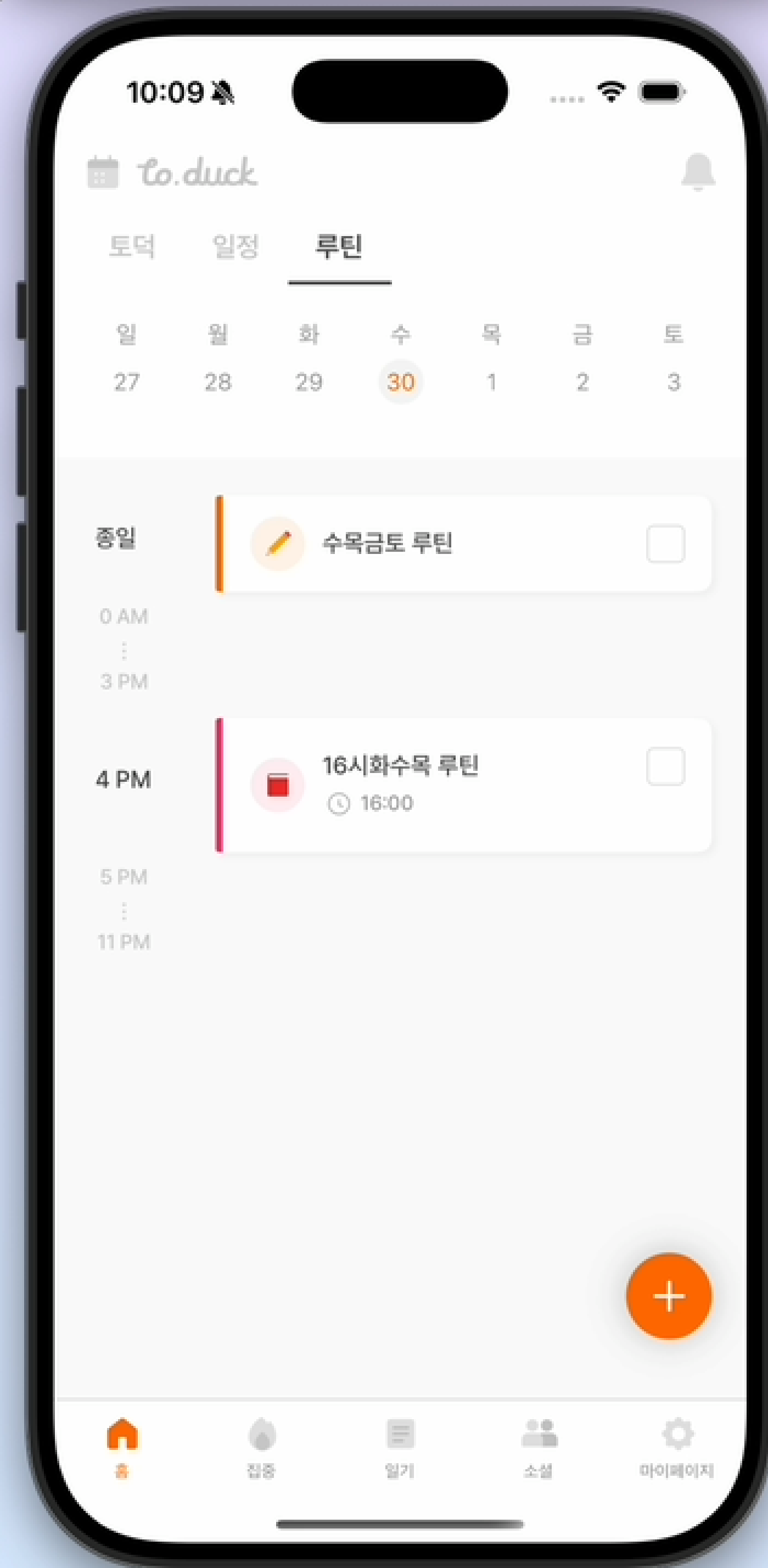
- 나의 할 일
- 제목, 카테고리, 시간
- 장소 설정 가능



기획 명세서

루틴

- 반복되는 나의 습관
- 제목, 카테고리, 시간
- 공개여부 설정 가능
- 장소가 없음



무지성 설계 - 일정

설계를 1도 모르던 시절, 작년의 나..



```
public struct Schedule {  
    public let title: String  
    public let category: String?  
    public let time: Date?  
    public let memo: String?  
    public let isFinish: Bool  
  
    public let place: String?  
}
```

10:57



to.duck



토덕 일정 루틴

일	월	화	수	목	금	토
4	5	6	7	8	9	10

종일

밥 약속 장소 알아보기 ☐

디저트 장소 알아보기 ☐

0 AM
⋮
6 AM

7 AM

7시 기상 후 운동 ☐
⌚ 07:00

8 AM
⋮
9 AM

10 AM

랩실 세미나 ☐
⌚ 10:00
📍 D128

11 AM
⋮
3 PM



홈

집중

일기

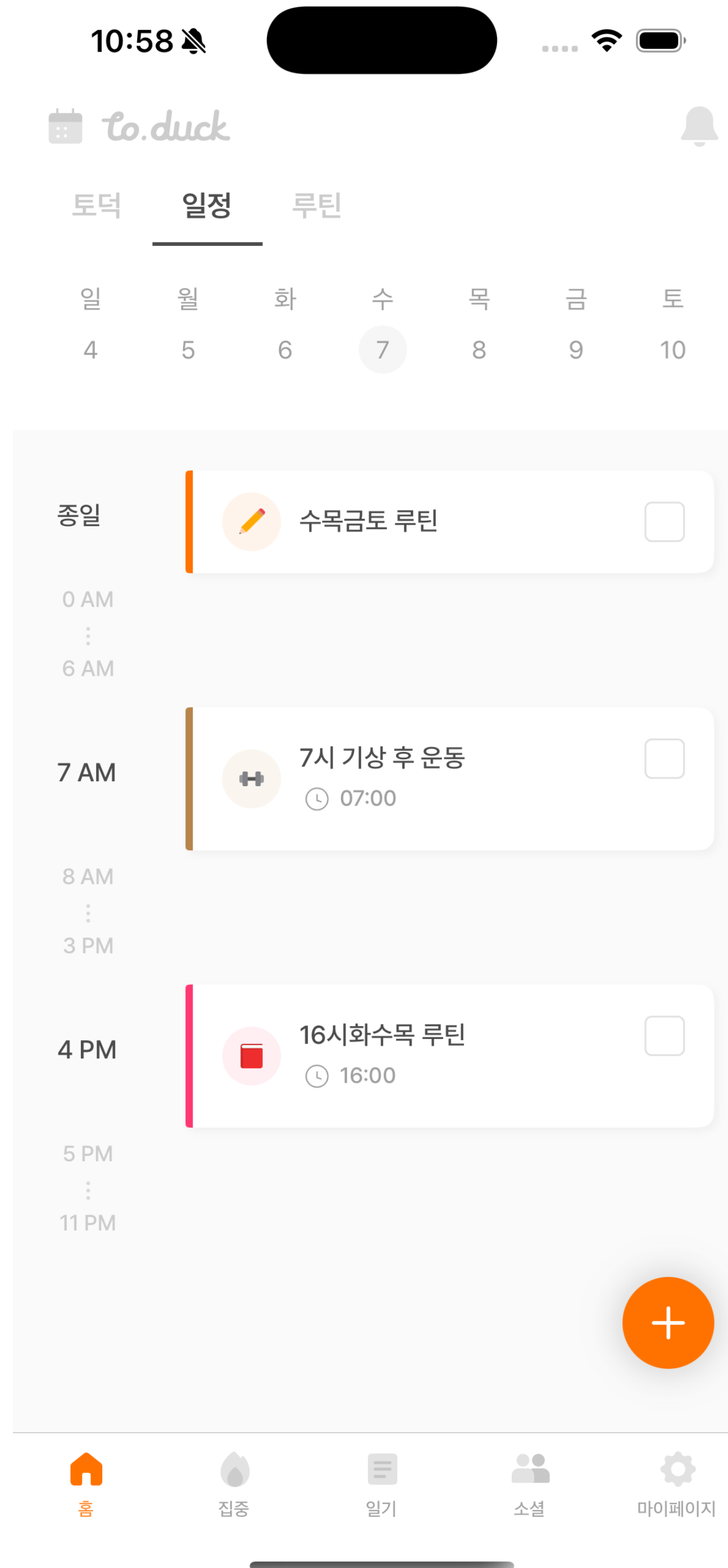
소셜

마이페이지

무지성 설계 - 루틴

설계를 1도 모르던 시절, 작년의 나..

```
public struct Routine {  
    public let title: String  
    public let category: String?  
    public let time: Date?  
    public let memo: String?  
    public let isFinish: Bool  
  
    public let isPublic: Bool?  
}
```



무지성 설계

일정

일정 UI 그리는 클래스

일정 CRUD 클래스

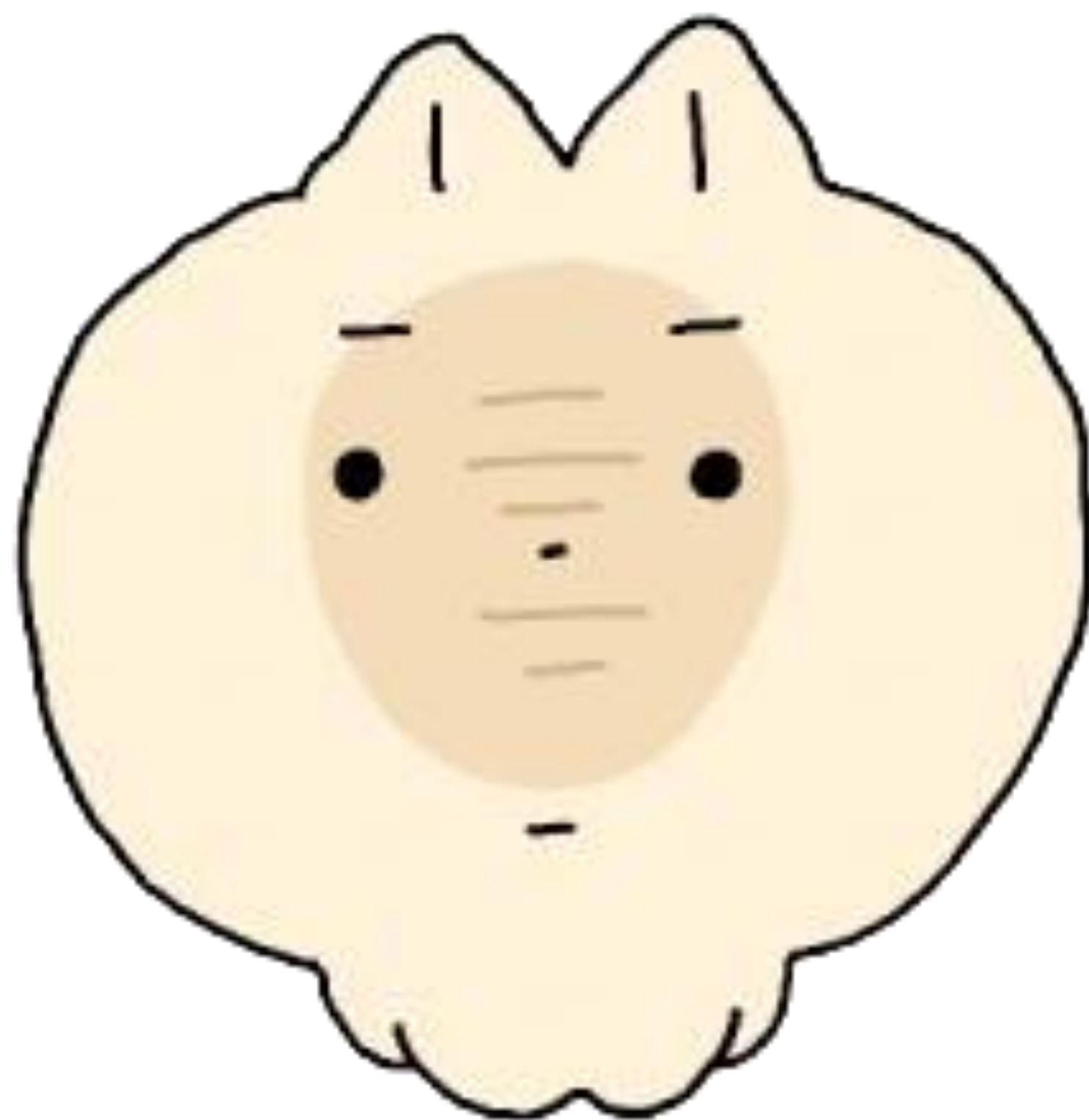
[일정] 배열을 갖는 클래스

루틴

루틴 UI 그리는 클래스

루틴 CRUD 클래스

[루틴] 배열을 갖는 클래스



그러던 어느 날.

“일정과 루틴을 ‘투두’에서 함께 볼 수 있게 해주세요”

뭐가 문제죠?

11:13

to.duck

토덕

투두

일

월

화

수

목

금

토

27

28

29

30

1

2

3

종일

0 AM

...

6 AM

7 AM

8 AM

...

9 PM

10 PM

11 PM

수목금토 루틴

7시 기상 후 운동
07:00

일정일정
22:00

일정 2
22:30

+

홈

집중

일기

소셜

마이페이지

뭐가 문제죠?

1. 같은 날짜, 같은 시간에 ‘루틴, 일정’을 한 번에 보여줘야 함

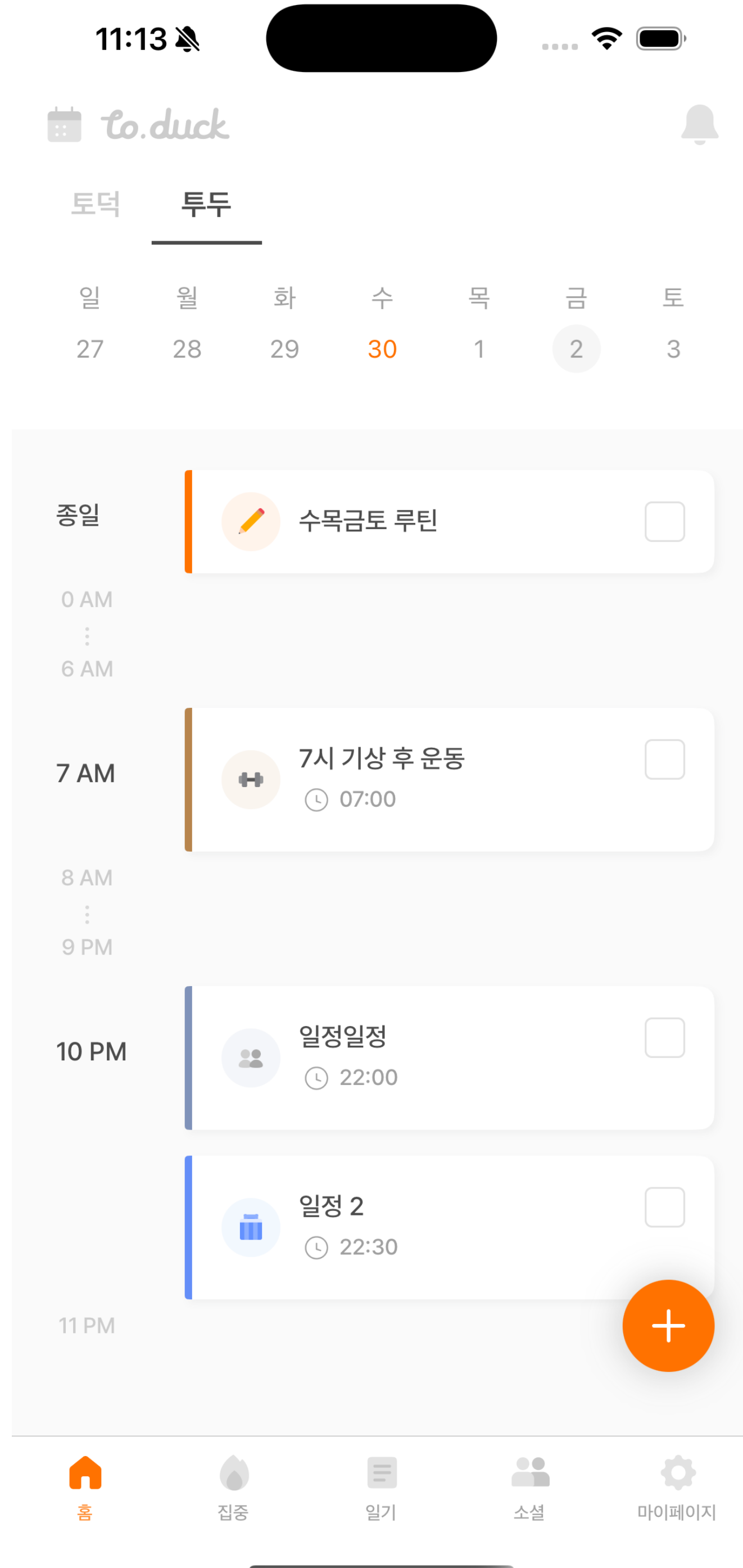
기존의 경우,

– ‘일정’ 세그먼트에서 10시에 [일정1, 일정2]

– ‘루틴’ 세그먼트에서 10시에 [루틴1, 루틴2]

-> 10시에 [일정1, 일정2, 루틴1, 루틴2]가 되어야 함

== [일정], [루틴]이 아닌, [일정과 루틴]이 되어야 함

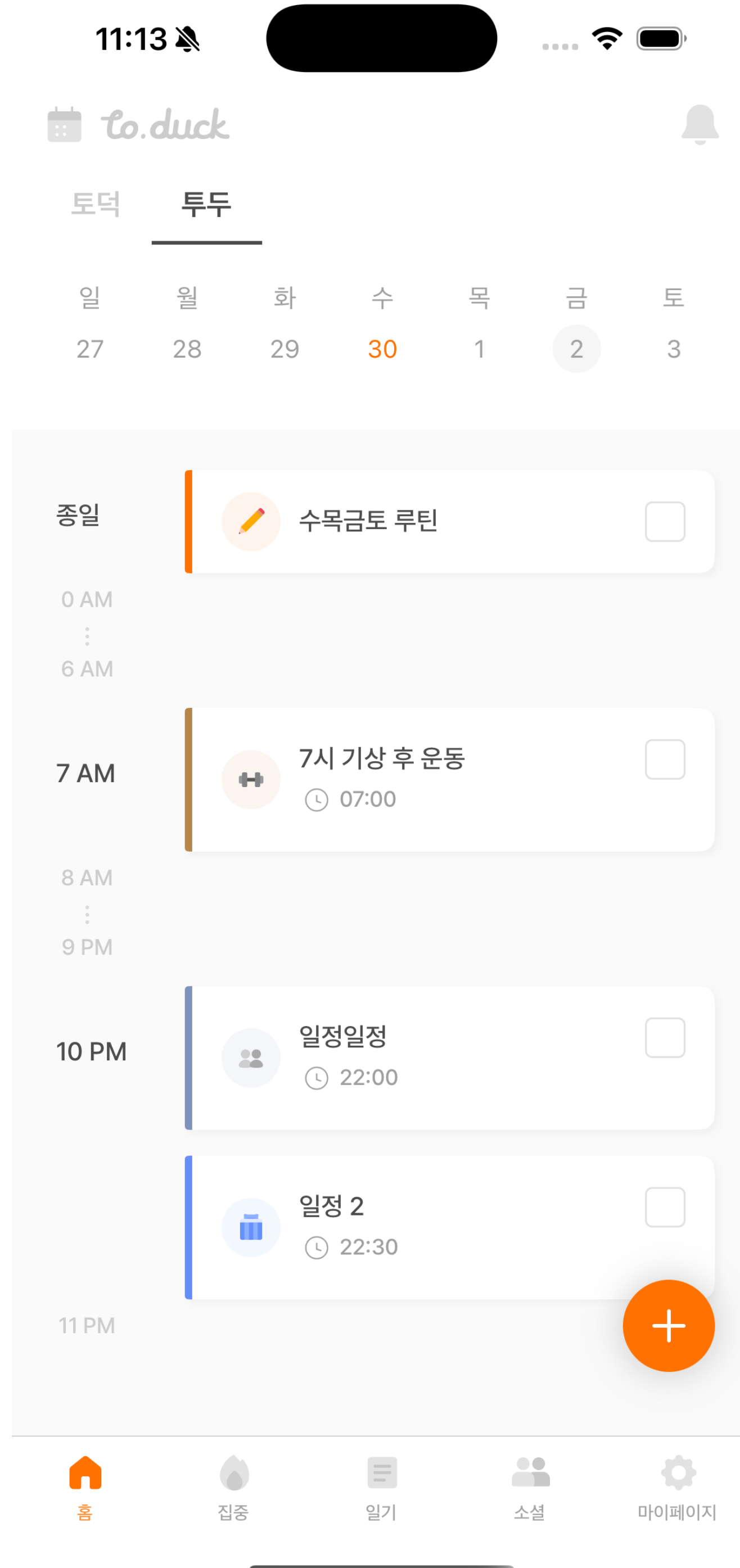


뭐가 문제죠?

2. 일정&루틴 역할을 할 수 있는 하나의 모델을 만들까 했지만,
서버는 일정과 루틴 CRUD가 나뉘어져 있음

서버로부터

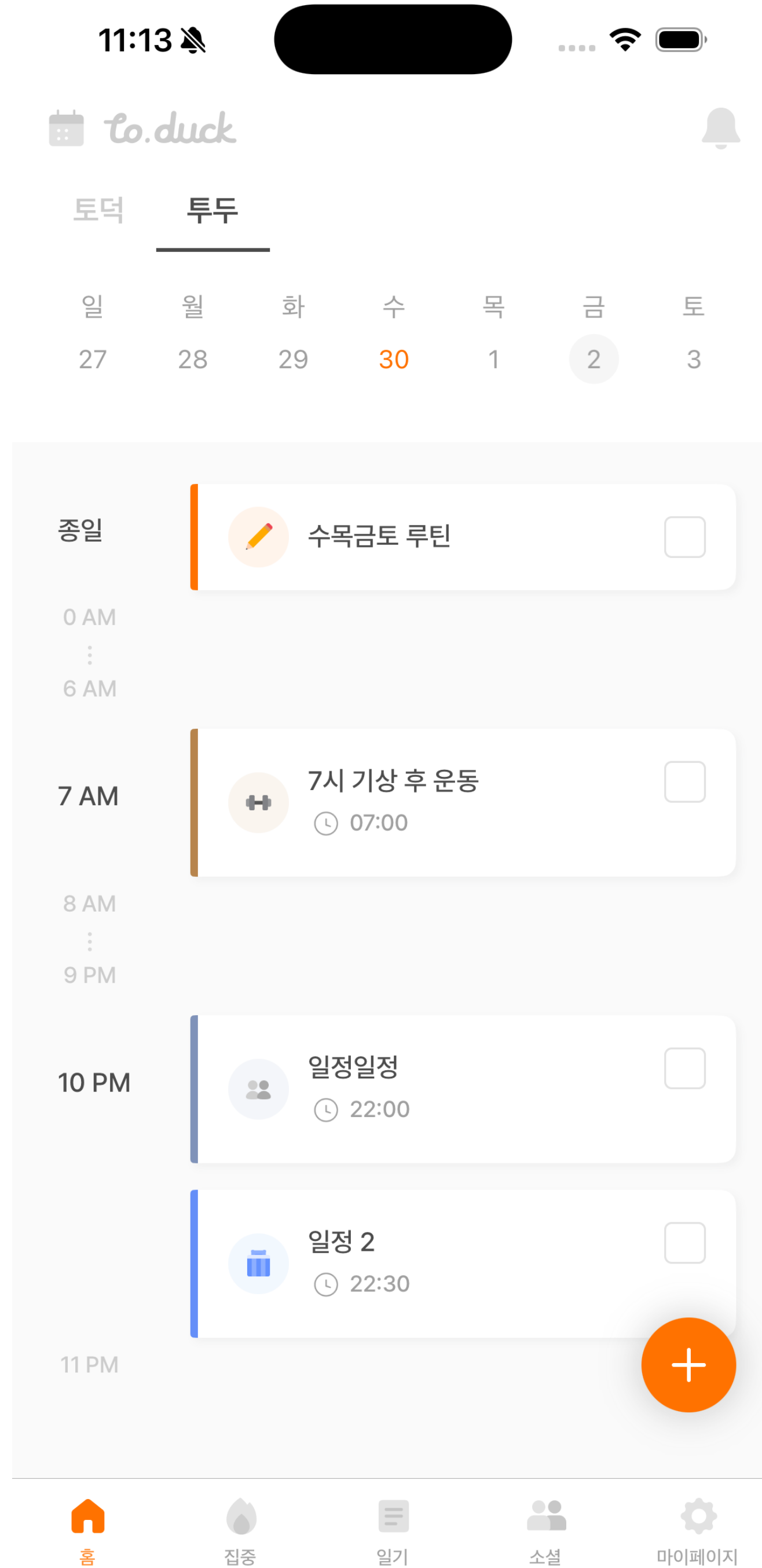
- 일정은 일정대로 받아오고
- 루틴은 루틴대로 받아와서
- 조합한 다음에 UI에 보여주는 역할이 필요함



뭐가 문제죠?

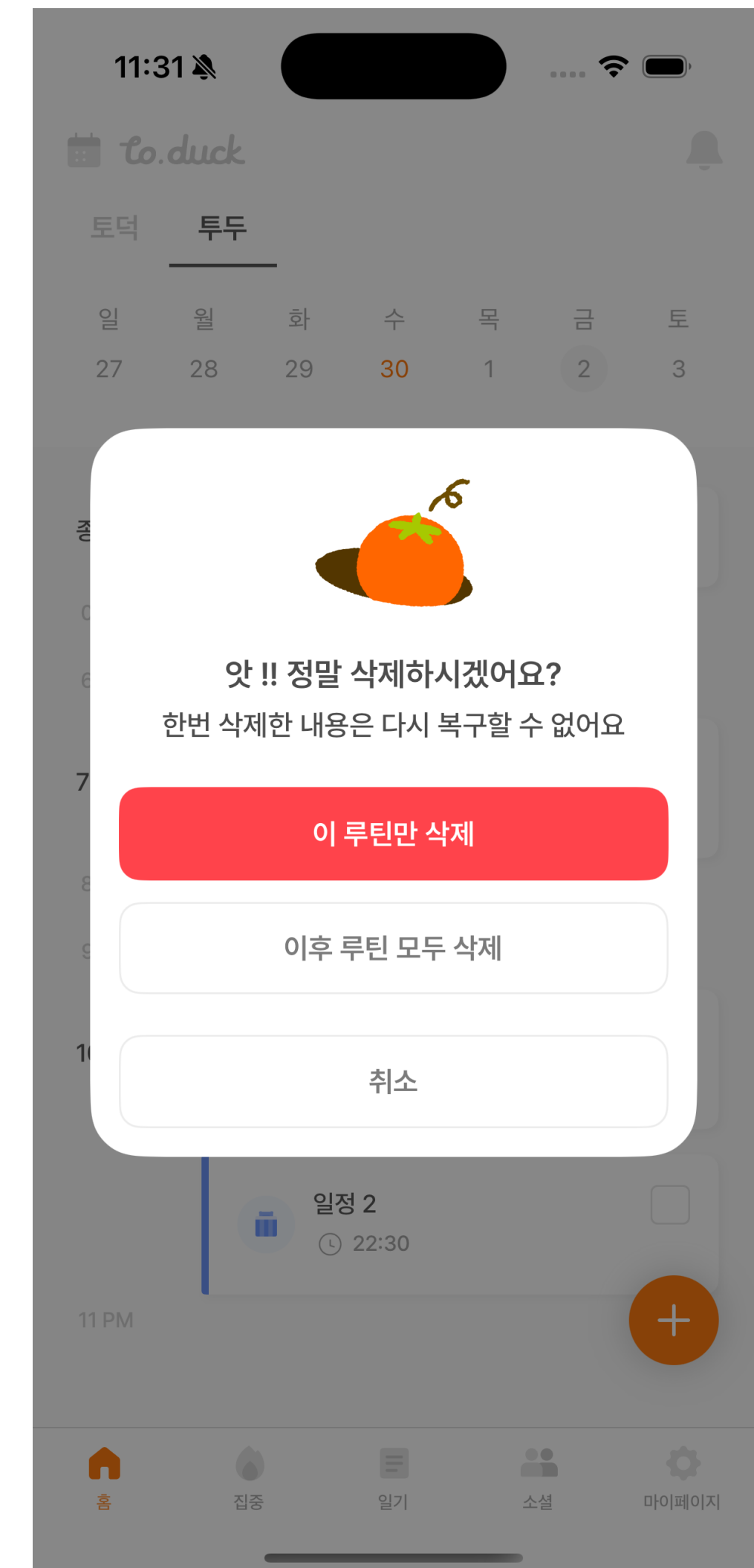
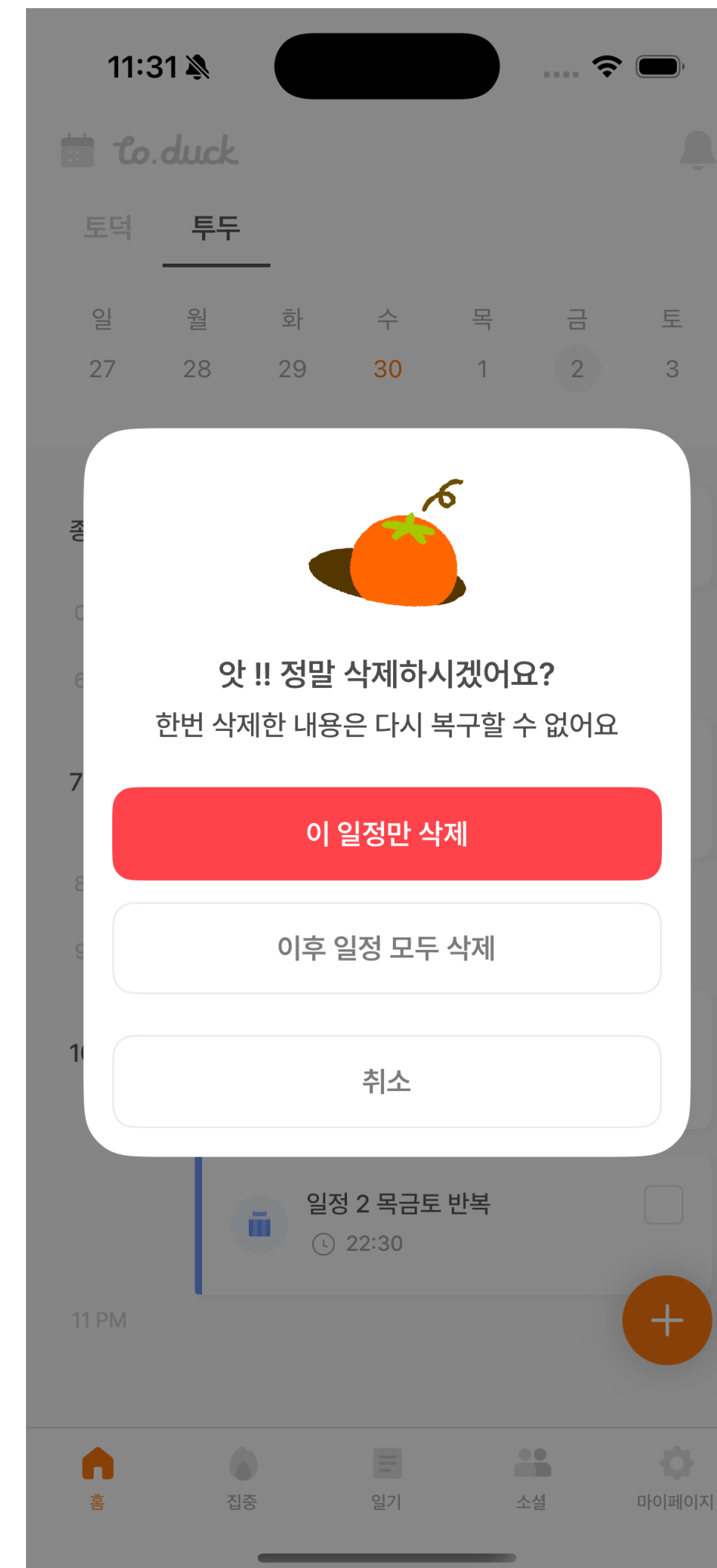
3. 10시에 대한 뷰를 그린다면,
[일정]에서 10시인 것들 꺼내고,
[루틴]에서 10시인 것들 꺼내도 되지만 뷰 코드가 복잡해짐

- 뷰는 하나인데, 배열은 일정 루틴 각각 존재 = 가독성 저하
- 상세보기, 삭제하기, 수정하기 등 매개변수로 넘겨줄 때
일정도 있어야 하고, 루틴도 있어야 함 = 불필요한 코드 생산



```
final class DeleteEventViewController:
    init(
        schedule: Schedule?,
        routine: Routine?
    ) {
        self.schedule = schedule
        self.routine = routine
        super.init()
    }
}
```

구현체를 받아줘야 함



문제를 느끼고 모델 간의 분리를 해야함을 인지했음

* 주의: 여기부터 필자의 뇌피셜이 가득할 수 있음

```
public struct Schedule {  
    public let title: String  
    public let category: String?  
    public let time: Date?  
    public let memo: String?  
    public let isFinish: Bool  
  
    public let place: String?  
}
```

```
public struct Routine {  
    public let title: String  
    public let category: String?  
    public let time: Date?  
    public let memo: String?  
    public let isFinish: Bool  
  
    public let isPublic: Bool?  
}
```

일정과 루틴은 모델이 비슷함

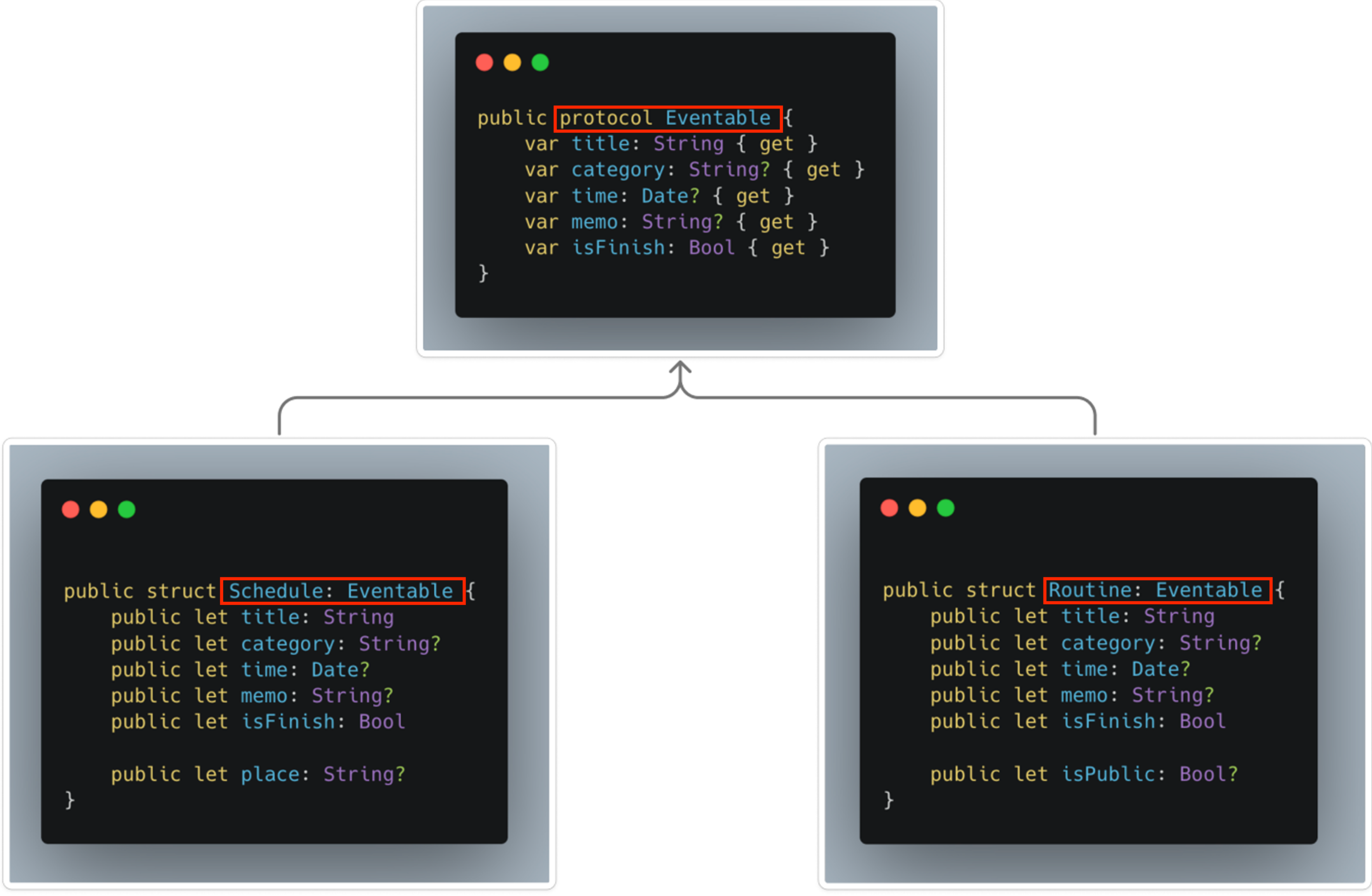
공통되는 부분만 묶어서 프로토콜(= 인터페이스)로 추상화하면 어떨까 ?

Eventable 프로토콜

(= 인터페이스)

- 일정과 루틴에서 공통되는 부분만 묶음
- 일정과 루틴이 해당 프로토콜을 채택함
- 이제 일정과 루틴은 Eventable이라 할 수 있음
- 일정의 장소, 루틴의 공개여부는 뒤에서 설명

```
public protocol Eventable {  
    var title: String { get }  
    var category: String? { get }  
    var time: Date? { get }  
    var memo: String? { get }  
    var isFinish: Bool { get }  
}
```



```
public protocol Eventable {  
    var title: String { get }  
    var category: String? { get }  
    var time: Date? { get }  
    var memo: String? { get }  
    var isFinish: Bool { get }  
}
```

```
public struct Schedule: Eventable {  
    public let title: String  
    public let category: String?  
    public let time: Date?  
    public let memo: String?  
    public let isFinish: Bool  
  
    public let place: String?  
}
```

```
public struct Routine: Eventable {  
    public let title: String  
    public let category: String?  
    public let time: Date?  
    public let memo: String?  
    public let isFinish: Bool  
  
    public let isPublic: Bool?  
}
```

EventDisplayItem 모델

- Eventable을 채택함
- 일정과 루틴을 모두 포용할 수 있음
- 뷰를 그리는 곳에서 사용할 모델
- Presentation Layer 모듈에서 사용

```
struct EventDisplayItem: Eventable {  
    let title: String  
    let category: String  
    let time: String?  
    let memo: String?  
    let isFinish: Bool  
  
    let place: String?  
    let isPublic: Bool?  
  
    init(  
        from event: Eventable,  
        place: String? = nil,  
        isPublic: Bool = false  
    ) {  
        ...  
    }  
}
```


Eventable 프로토콜

(= 인터페이스)



```
extension Eventable {  
    var categoryColor: UIColor {  
        return category.convertToUIColor() ?? .clear  
    }  
  
    var categoryIcon: UIImage? {  
        return UIImage.categoryDictionary[category]  
    }  
}
```

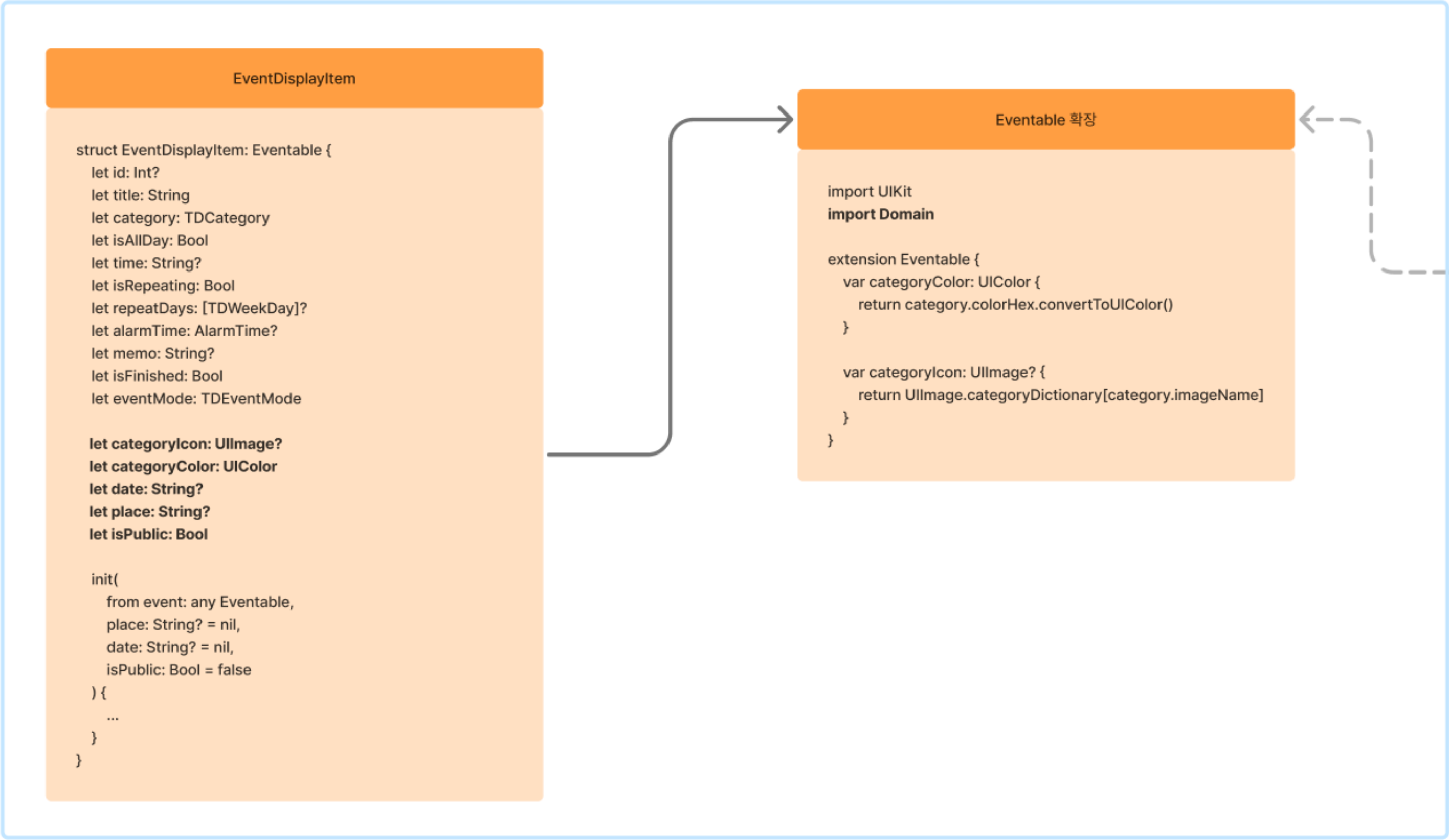
해당 프로토콜에 확장을 통해 Domain Layer가 아닌,

뷰를 그리는 Presentation Layer 모듈에서는 UI Color와 Image까지 디폴트로 갖게 함

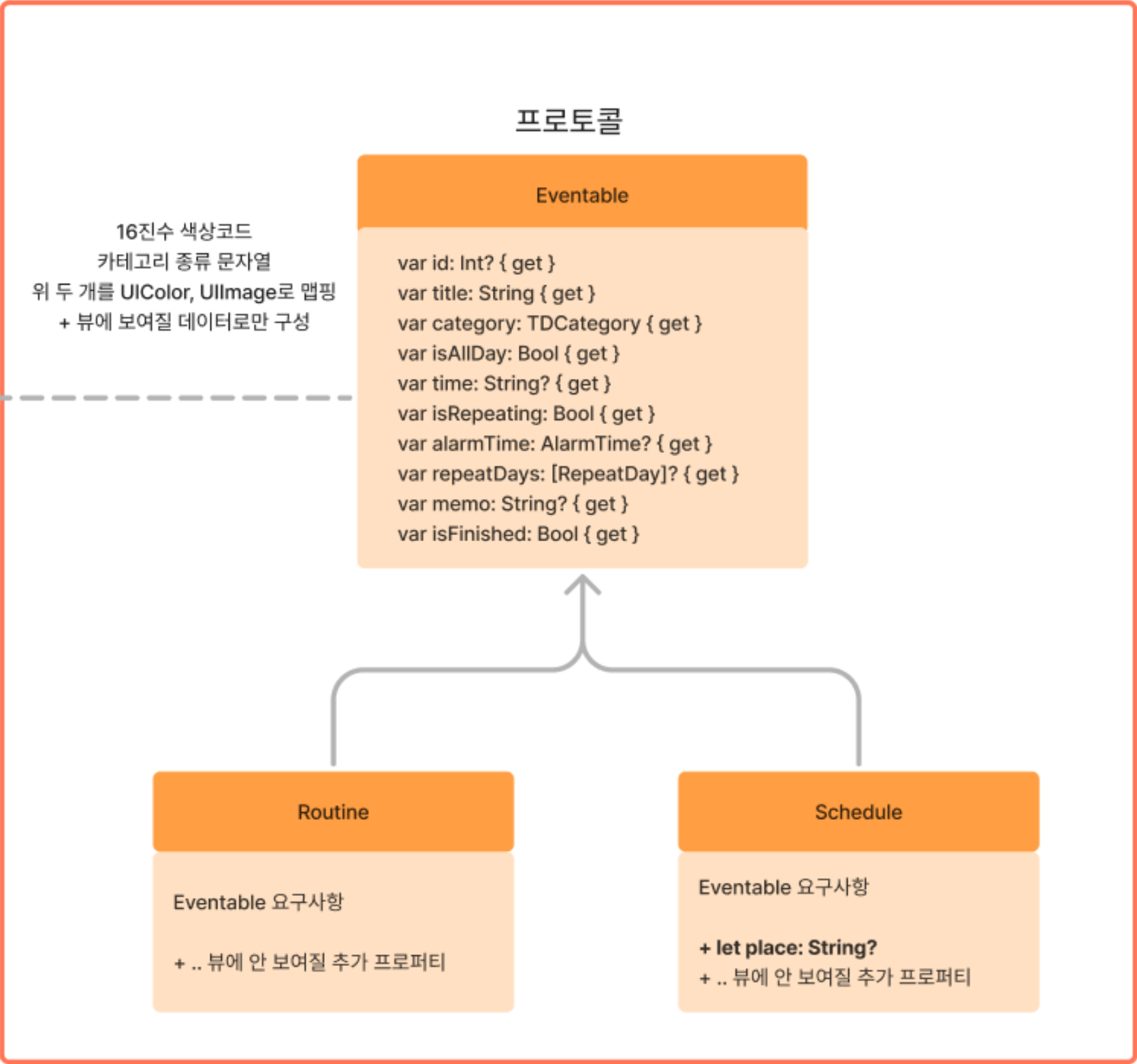
Eventable 프로토콜

(= 인터페이스)

Presentation

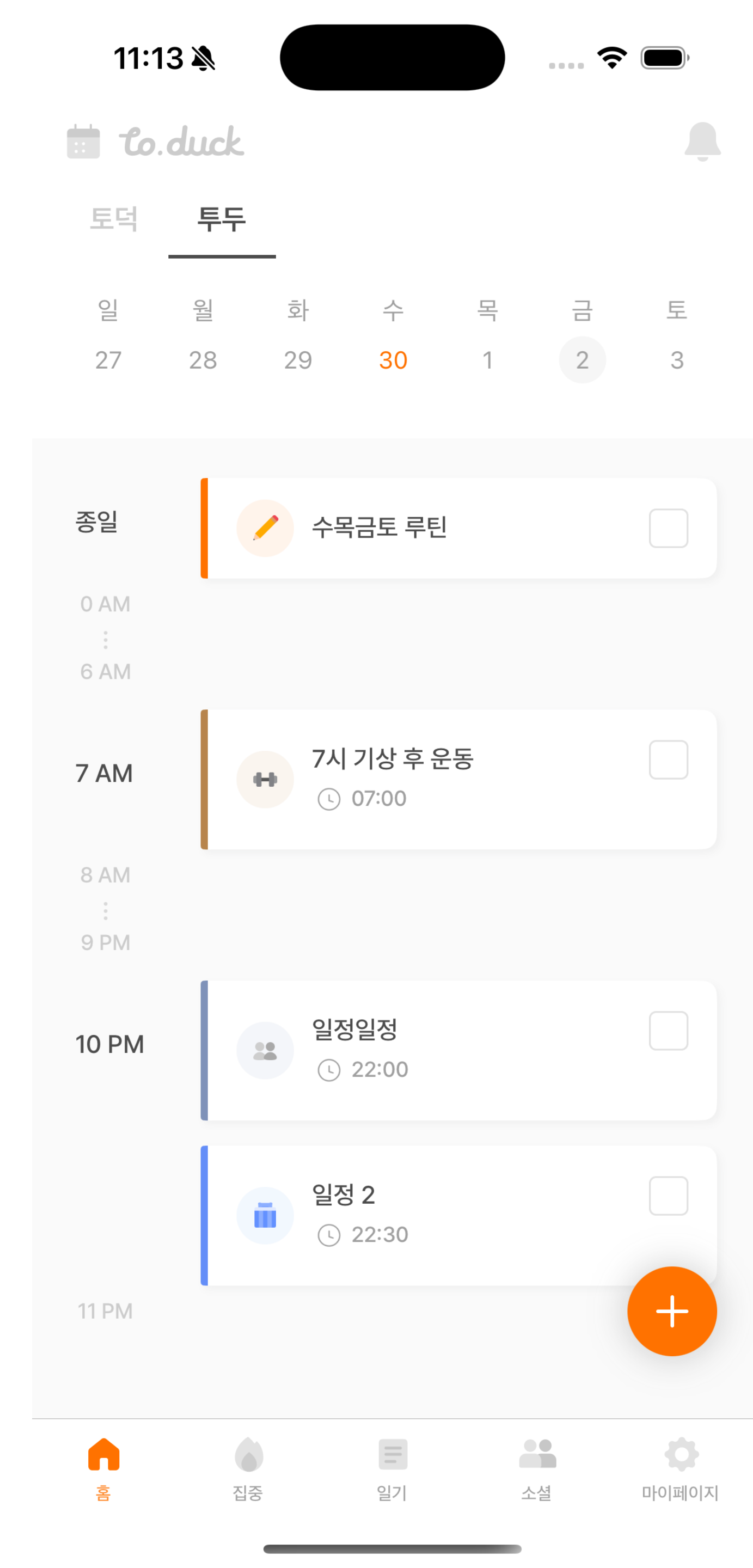


Domain



EventDisplayItem는 투두 UI 그리는 데에 충분한 모델임

그럼 UI를 그리기 위해 참조할 모델이
[EventDisplayItem]이면 될까 ?



ㄴ ㄴ

서버에는 ‘일정 조회’, ‘루틴 조회’가 각각 따로 있으며
이들은 각각 구현체가 넘어올 거임

용어정리 “투두” == 일정과 루틴을 아우르는 말

// MARK: - 투두 리스트 가져오기

```
private func fetchWeeklyTodoList(startDate: String, endDate: String) async {  
    do {  
        let fetchedWeeklyScheduleList = try await fetchScheduleListUseCase.execute(  
            startDate: startDate,  
            endDate: endDate  
        )  
        let fetchedWeeklyRoutineList = try await fetchRoutineListForDatesUseCase.execute(  
            startDate: startDate,  
            endDate: endDate  
        )  
        self.weeklyScheduleList = fetchedWeeklyScheduleList  
        self.weeklyRoutineList = fetchedWeeklyRoutineList  
        output.send(.fetchedTodoList)  
    } catch {  
        output.send(.failure(error: "투두를 불러오는데 실패했습니다."))  
    }  
}
```

fetchWeeklyScheduleList

```
let fetchedWeeklyScheduleList: [Date : [Schedule]]
```

TodoViewModel.swift

```
do {  
    let fetchedWeeklyScheduleList = try await fetchScheduleListUseCase.execute(  
        startDate: startDate,  
        endDate: endDate  
    )  
}
```

서버로부터 받은 일정들

fetchWeeklyRoutineList

```
let fetchedWeeklyRoutineList: [Date : [Routine]]
```

TodoViewModel.swift

```
let fetchedWeeklyRoutineList = try await fetchRoutineListForDatesUseCase.execute(  
    startDate: startDate,  
    endDate: endDate  
)
```

서버로부터 받은 루틴들

‘투두’ UI에 사용할 모델 결정

- [Eventable] 배열을 사용하자, 업 캐스팅 사용
- `private var todoList: [any Eventable] = []`
- 구체 타입에 의존하지 않고, 추상 타입(프로토콜)에 의존하는 설계
- Routine, Schedule, EventDisplayItem 모두 담을 수 있음

```
private func unionTodoListForSelectedDate(selectedDate: Date) {  
    let selectedDateScheduleList = weeklyScheduleList[selectedDate] ?? []  
    let selectedDateRoutineList = weeklyRoutineList[selectedDate] ?? []  
  
    let todoList: [any Eventable] = (selectedDateScheduleList as [any Eventable]) + (selectedDateRoutineList as [any Eventable])  
  
    self.todoList = todoList  
}
```

모델 추상화 적용 - 투두 삭제하기

- todoList에는 Schedule 혹은 Routine이 업캐스팅 되어 담겨있음
- 삭제의 경우, id만 있으면 되므로 todoList의 추상타입 그대로 id만 꺼내서 서버에 보내면 됨
- 추상타입 or 구체타입 상관 없음

```
let todoList: [any Eventable]
```

일정 삭제하기

- id
- 일정인지, 루틴인지 Bool 전달

루틴or일정 따라 서버에 id 전달하여 삭제

모델 추상화 적용 - 투두 상세보기

- todoList에는 Schedule 혹은 Routine이 업캐스팅 되어 담겨있음
- 투두 상세보기는 일정 혹은 루틴을 보여주므로 도메인 모델이 아닌 UI 모델이 필요함
- todoList에 담겨있는 추상 타입을 EventDisplayItem 으로 매핑해서 투두 상세보기 클래스에 전달함
- EventDisplayItem을 전달받은 클래스는 일정인 경우 place, 루틴인 경우 isPublic을 포함하여 UI에 그려줌

```
let todoList: [any Eventable]
```

일정 상세보기

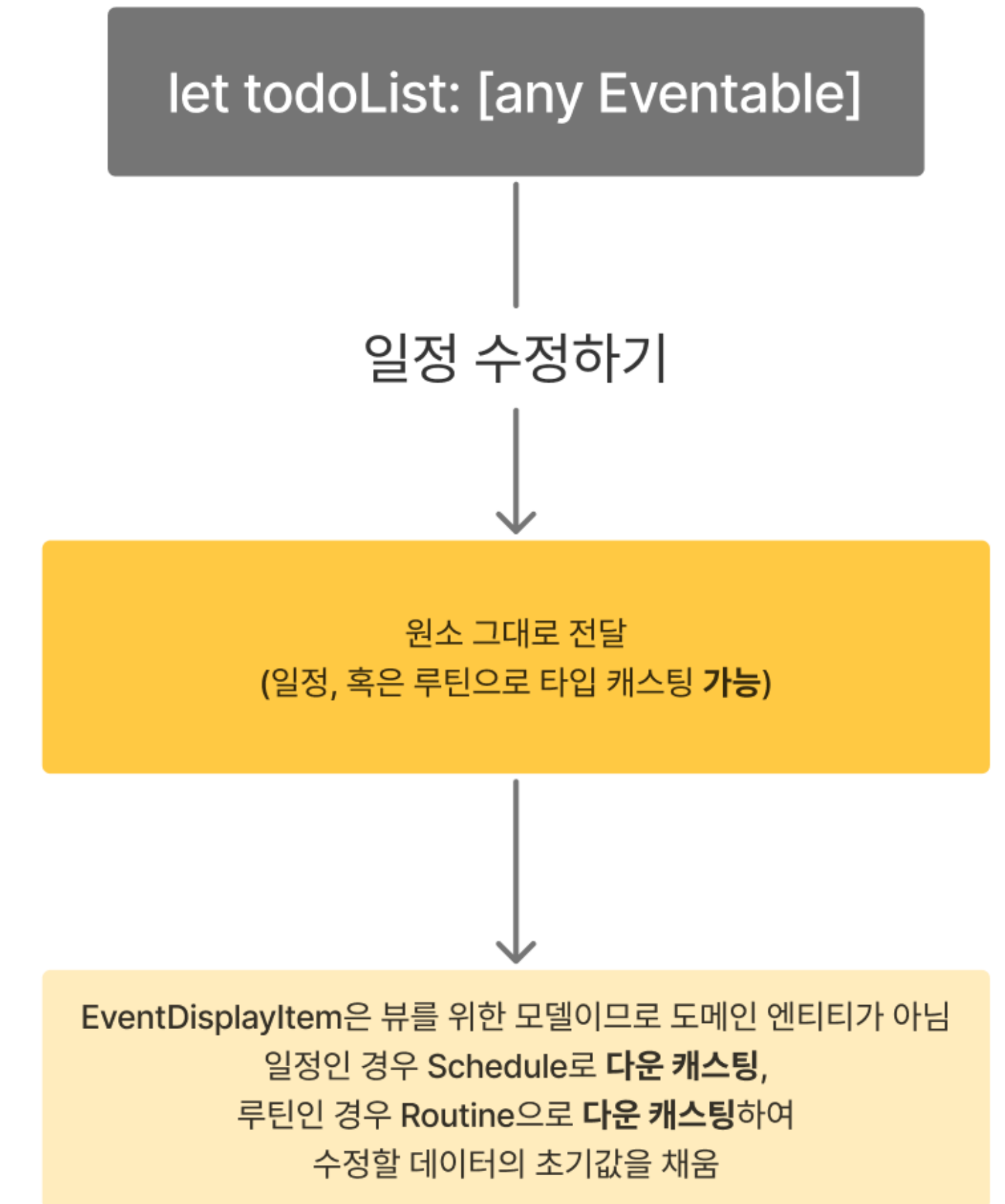
EventDisplayItem으로 매핑함

뷰를 나타내는 모델을 받아서
제목, 카테고리, 메모는 채우고
일정인 경우 place, 루틴인 경우 isPublic까지 채워줌

(일정, 혹은 루틴으로 타입 캐스팅 불가능)

모델 추상화 적용 - 투두 수정하기

- todoList에는 Schedule 혹은 Routine이 업캐스팅 되어 담겨있음
- 수정을 위해선 이전 데이터의 모든 내용을 알아야 함
(수정은 새로 만드는 것과 유사함)
- 일정의 경우 place 뿐만 아니라 start, end 날짜 등등 필요
- 루틴의 경우 isPublic 뿐만 아니라 반복 날짜 등등 필요
- 따라서 수정하기 클래스에서 any Eventable로 수정할 투두를 받고, 내부에서 타입 캐스팅(Down)을 통해 수정할 투두의 초기값을 채움



수정하기 클래스에서 any Eventable로 수정할 투두를 받고,
내부에서 타입 캐스팅(Down)을 통해 수정할 투두의 초기값을 채움

```
6 final class EventMakorViewModel: BaseViewModel {
60 // MARK: - Initializer
61 init(
62     mode: EventMakorViewController.Mode,
63     preEvent: (any Eventable)?,
64     selectedDate: Date? = nil
65 ) {
66     self.mode = mode
67     self.preEvent = preEvent
68     self.selectedDate = selectedDate
69     initialValueSetupForEditMode()
70 }
71
72 private func initialValueSetupForEditMode() {
73     guard let preEvent else { return }
74
75     if let schedule = preEvent as? Schedule {
76         self.title = schedule.title
77         self.selectedCategory = schedule.category
78         self.startDate = schedule.startDate
79         self.endDate = schedule.endDate
80         self.isAllDay = schedule.isAllDay
81         self.time = schedule.time
82         self.repeatDays = schedule.repeatDays
83         self.alarm = schedule.alarmTime
84         self.location = schedule.place
85         self.memo = schedule.memo
86     } else if let routine = preEvent as? Routine {
87         self.title = routine.title
88         self.selectedCategory = routine.category
89         self.isAllDay = routine.isAllDay
90         self.time = routine.time
91         self.isPublic = routine.isPublic
92         self.repeatDays = routine.repeatDays
93         self.alarm = routine.alarmTime
94         self.memo = routine.memo
95     }
96 }
```

일정

루틴

let todoList: [any Eventable]

일정 상세보기

일정 수정하기

일정 삭제하기

EventDisplayItem으로 맵핑함

뷰를 나타내는 모델을 받아서
제목, 카테고리, 메모는 채우고
일정인 경우 place, 루틴인 경우 isPublic까지 채워줌

(일정, 혹은 루틴으로 타입 캐스팅 불가능)

원소 그대로 전달
(일정, 혹은 루틴으로 타입 캐스팅 가능)

EventDisplayItem은 뷰를 위한 모델이므로 도메인 엔티티가 아님
일정인 경우 Schedule로 **다운 캐스팅**,
루틴인 경우 Routine으로 **다운 캐스팅**하여
수정할 데이터의 초기값을 채움

- id
- 일정인지, 루틴인지 Bool 전달

루틴or일정 따라 서버에 id 전달하여 삭제

마무리 정리

- 기획과 디자인에 맞는 **도메인 모델을 설계했음**
- 공통된 모델의 프로퍼티를 추상화하여 프로토콜(인터페이스)로 정의했음
- 프로토콜 타입, 즉 추상 타입의 프로퍼티를 두어 구체 타입이 아닌 추상 타입에 의존하게 하여 결합도를 낮춤
- 값 타입인 Schedule, Routine과 EventDisplayItem에 프로토콜을 채택하게 하여 클래스가 아닌 값 타입간의 OOP 특징을 적용함
- 위 내용들을 종합하여 **프로토콜 지향 프로그래밍**을 코드 레벨에서 경험함

Q & A