iOS Seminar 3

Wafflestudio 2024

과제 2 공통 피드백

Enumeration (관련 문서)

- path, parameter 등... 매 Request마다 지정해서 보내야 하는 값
  - 매번 새 변수로 선언하게 되면, `let parameters: Parameters = [...]`의 반복
- 여러 case에서 공통되는 Type의 Variable을 사용해야 할 때
  - Enumeration+Switch문의 활용 : 코드 반복을 줄일 수 있다!

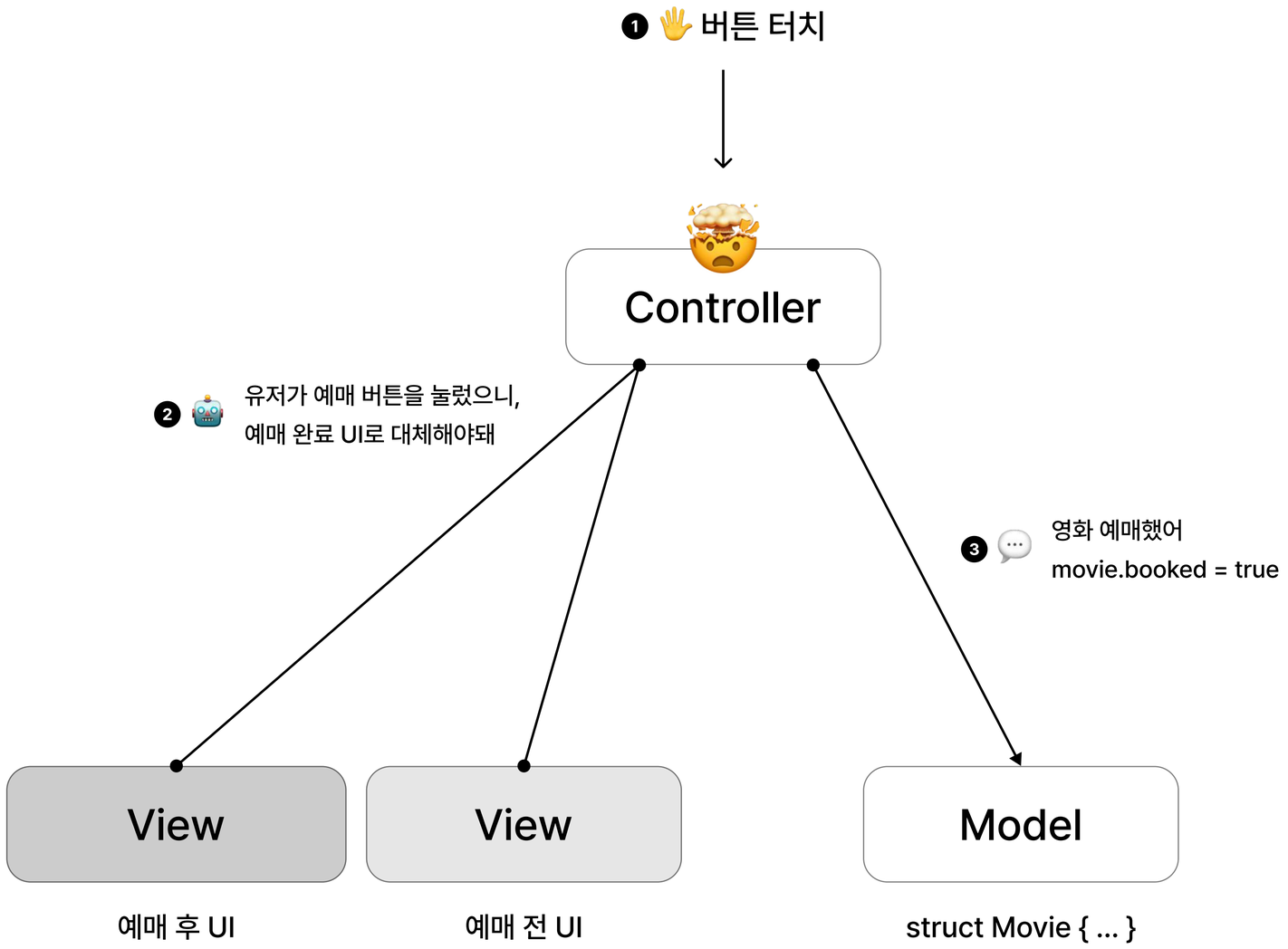
Design Pattern

Design Pattern

- MVC
- MVVM
- 그 외 MVP 등...

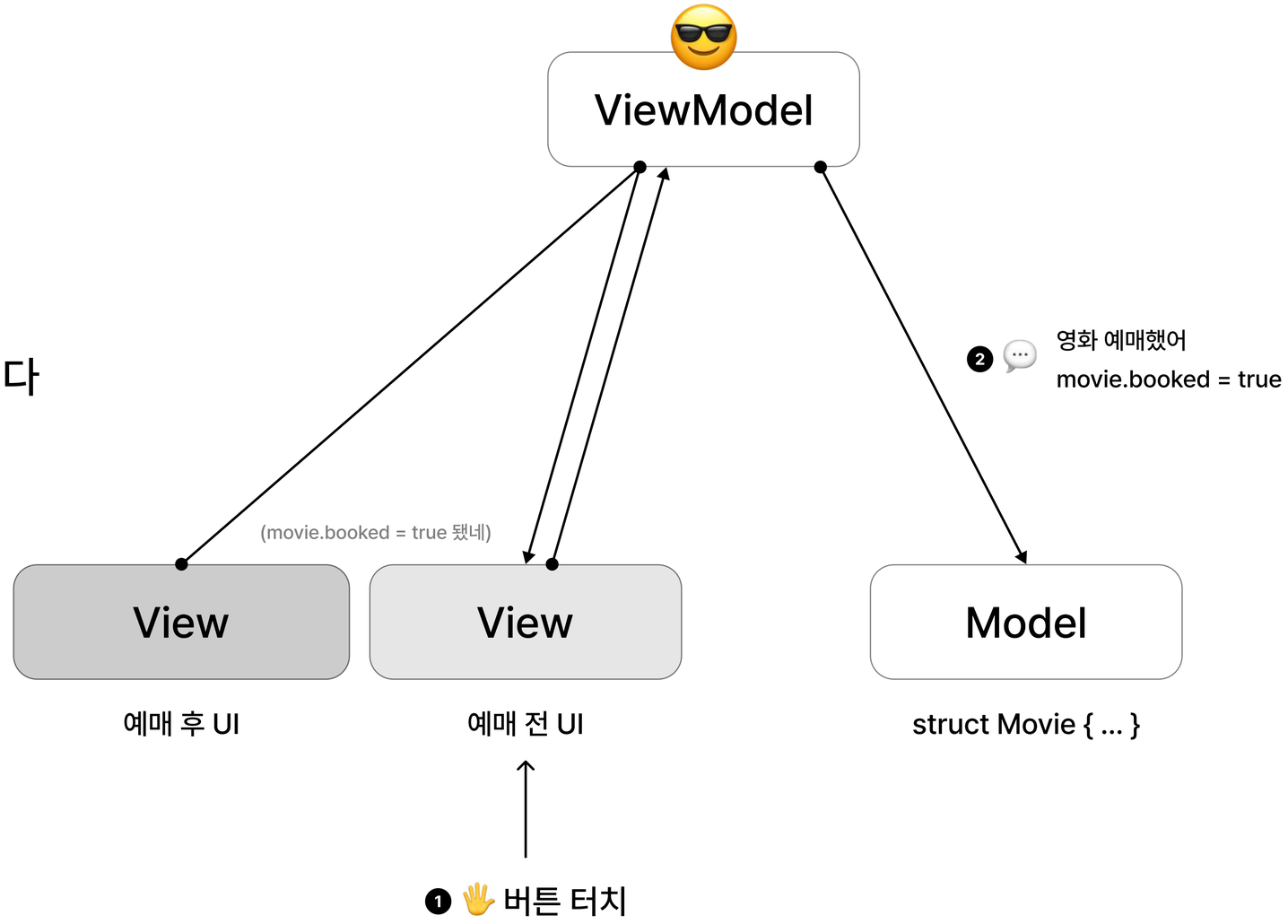
MVC

- Model - View - Controller
- 이제까지 진행했던 과제와 유사한 패턴
- Massive - View - Controller
  - 규모가 큰 프로젝트일수록 잘...



MVVM

- Model - View - ViewModel
- View와 Model 간의 결합도를 없앰
  - ViewModel이 바뀌면 View도 바뀐다



SwiftUI

SwiftUI

- WWDC 2019에서 발표한 프레임워크
- Declarative syntax
  - (UIKit은 Imperative)
- Data Driven
- HStack, VStack, ZStack, Spacer 등을 이용

```swift
import SwiftUI

struct MainView: View {

    @State private var id: String = ""
    @State private var password: String = ""
    @State private var pushToAfterLoginView: Bool = false

    var body: some View {
        VStack {
            VStack(alignment: .leading, spacing: 0) {
                Text("iOS Seminar")
                    .font(.system(size: 28, weight: .semibold))

                Spacer().frame(height: 48)

                VStack(spacing: 32) {
                    TextField("아이디", text: $id, prompt: Text("아이디(8자 이하)"))
                        .font(.system(size: 17))
                        .textInputAutocapitalization(.never)

                    TextField("비밀번호", text: $password, prompt: Text("비밀번호"))
                        .font(.system(size: 17))
                        .textContentType(.password)
                        .textInputAutocapitalization(.never)
                }
            }
            .padding(.horizontal, 24)

            /// ...
```

UIKit보다 좋은 점?

- 과제1의 TodoTableViewCell을 SwiftUI로 구현한다면...
- ex) 좌우 간격 24, 상하 간격 16을 설정하고자 했을 때
  - UIKit

```
98      private func setLayoutConstraint() {
99          addSubview(labelStackView)
100         labelStackView.translatesAutoresizingMaskIntoConstraints = false
101         NSLayoutConstraint.activate([
102             labelStackView.leadingAnchor.constraint(equalTo: checkImageView.trailingAnchor, constant: 24),
103             labelStackView.trailingAnchor.constraint(equalTo: safeAreaLayoutGuide.trailingAnchor, constant: -24),
104             labelStackView.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor, constant: 16),
105             labelStackView.bottomAnchor.constraint(equalTo: safeAreaLayoutGuide.bottomAnchor, constant: -16),
106         ])
107     }
```

  - SwiftUI

```
42          .padding(.horizontal, 24)
43          .padding(.vertical, 16)
```

그럼에도 UIKit을 배우는 이유

- 복잡한 custom UI를 구현하고자 하는 경우
  - UIKit + SwiftUI 혼용하는 프로젝트도 다수 존재
- iOS 15까지 pure SwiftUI로는 Alert에 TextField를 넣을 수 없었다...

SwiftUI Life cycle

UIKit → SwiftUI Life cycle (공식문서)

- 기존 UIKit을 사용했을 때 앱의 entry point는 `@main` annotation이 붙어있던 AppDelegate
  - `@main`: 특정 structure, class, 혹은 enumeration이 top-level entry point임을 나타낸다
  - C의 `int main(int argc, char *argv[])`같은 느낌
- SwiftUI에서는 `<MyAppName>App.swift`
- 관련 문서 (SwiftUI Tutorial)

```swift
import SwiftUI

@main
struct Seminar3DemoApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

SwiftUI+Data Managing

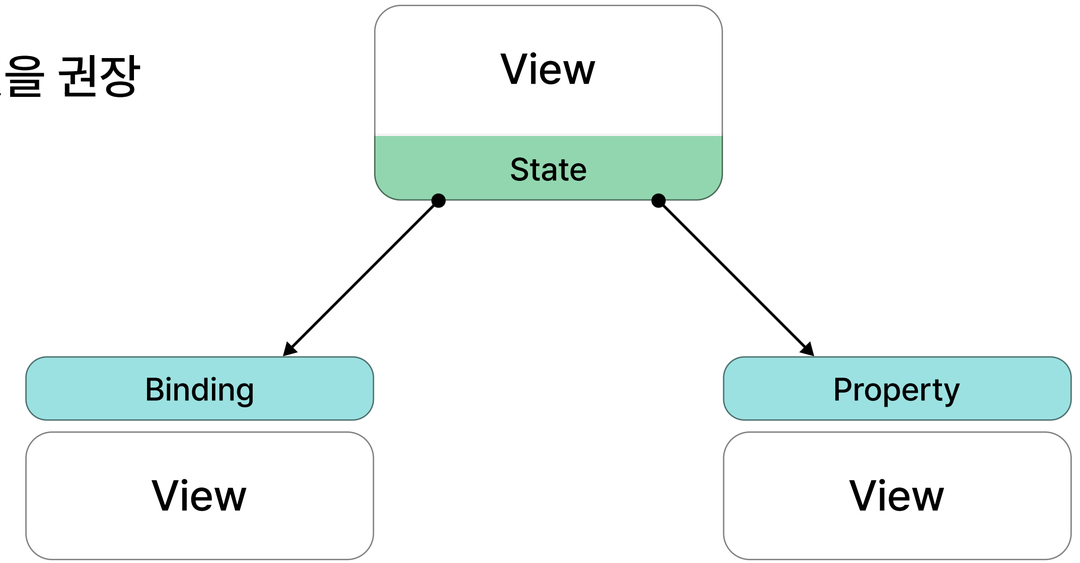
SwiftUI가 View를 업데이트하는 방식

- 관련 문서
- SwiftUI는 Data Driven
  a. View state 공유
  b. Model data의 변경사항 관찰 (이건 다음 세미나에서)
- View에 의존성이 있는 데이터가 업데이트되면, 그 데이터에 연관된 인터페이스만 자동으로 업데이트

View State Sharing

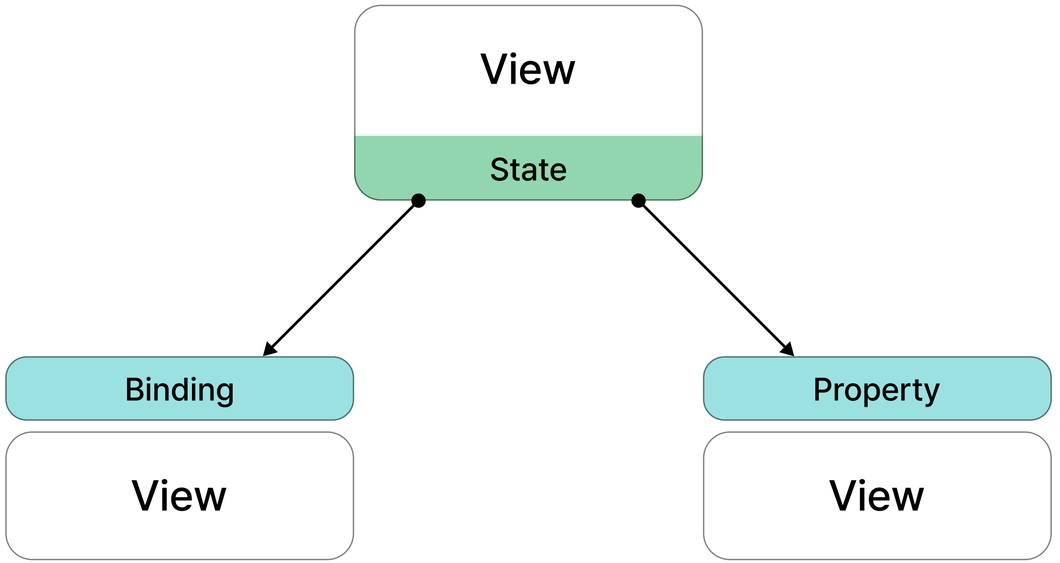
- 관련문서
- Store data as state in the least common ancestor of the views that need the data to establish a single source of truth that’s shared across views.
- 다만 이러한 state는 persistent한 데이터보다는,

  User Interface에만 영향을 미치는 데이터에만 적용할 것을 권장



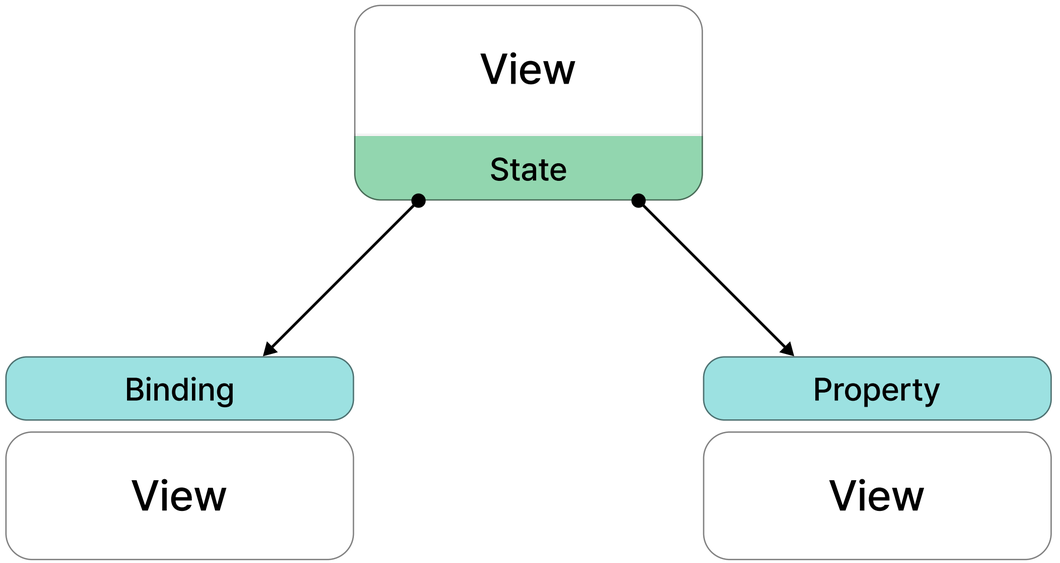
@State (공식문서)

- User Interface state를 지역적으로 관리하는 경우
  - ex) 화면의 counter 값을 1씩 올리기
- 하위 View에게 read-only로 공유하거나, binding을 이용해 read-write로 공유
  - binding으로 공유하는 경우 `$`
- 어느 스레드에서든 안전하게 값 변경 가능



@Binding (공식문서)

- 상위 View의 state를 받아 사용하면서도, 현재 View(하위 View)에서의 변경사항이 상위 View에 영향을 미치도록 하고 싶은 경우
- ex) 상세 페이지에서 Toggle을 on/off는 경우에 따라 메인 페이지를 다르게 보여주기



과제 3 안내

SwiftUI를 이용한 Todo List 앱 만들기

- 과제 3 디자인 링크(Figma)
- 과제 목표
  - SwiftUI를 이용한 state 관리 및 UI 구현
- 관련 키워드
  - @State, @Binding, @Environment
  - NavigationView / NavigationLink
  - UserDefaults

과제 상세 스펙 및 주의사항 (1)

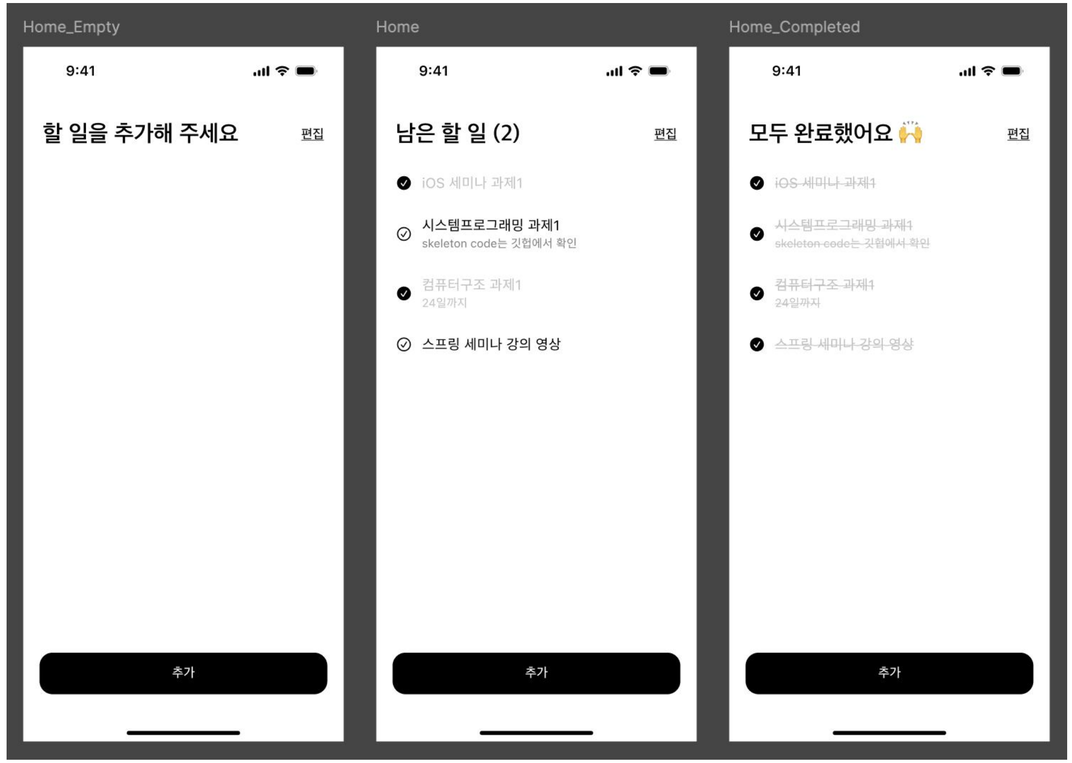
- 전체
  - UI 구현은 SwiftUI만 사용합니다
  - 아직 강의 자료 15페이지에서의 “Model data”의 변경사항 관찰에 대한 내용을 다루지 않아 완전한 MVVM 구조를 짜기 어렵습니다.

    미리 관련한 내용을 공부하고 MVVM 구조로 짜도 좋습니다

    (관련 키워드 : @ObservableObject @ObservedObject @StateObject @Published)
  - Text, Color, Constraint, Corner Radius 등은 피그마에 주어진 값을 사용하며, 누락된 부분이 있다면 슬랙에서 질문 부탁드립니다
- Font
  - SF Pro / Apple SD Gothic Neo는 Apple의 시스템 폰트이므로 과제 진행 시 size, weight, color만 고려합니다
- Alert
  - 별도의 디자인이 필요하지 않습니다. 문구와 동작만 알맞게 설정합니다

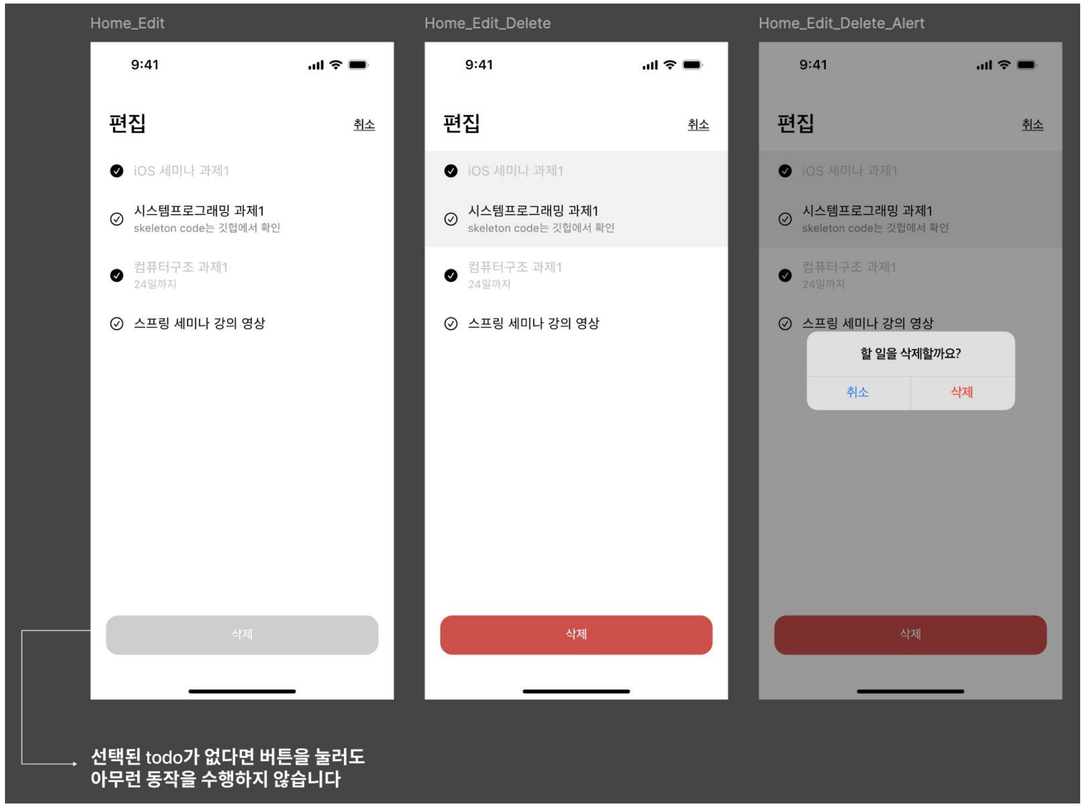
과제 상세 스펙 및 주의사항 (2)

- 홈 화면 (기본 모드)
  - 앱에 처음 진입하는 경우에는 언제나 기본 모드를 보여줍니다
  - Todo 목록의 상태에 따라 상단 문구를 알맞게 보여줍니다
    - NavigationBar Title이 아닌, Custom View입니다
  - 체크 아이콘을 탭하면 Todo의 완료 여부를 전환합니다
    - 구현에 따라 Cell 영역 어디든 탭해도 Todo의 완료 여부가 전환되도록 해도 괜찮습니다
    - 완료된 Todo는 취소선을 보여줍니다. 달라지는 Text 색상에 유의합니다



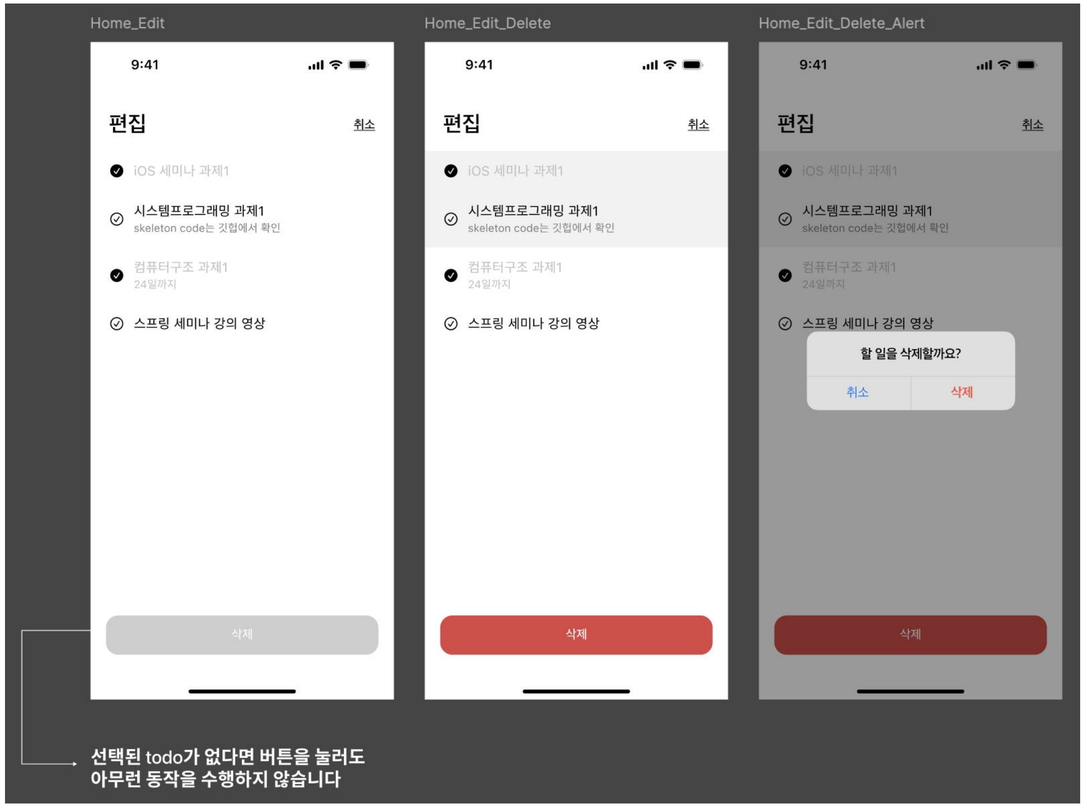
과제 상세 스펙 및 주의사항 (3)

- 홈 화면 (편집 모드)
  - 기본 모드에서 “편집” 버튼을 누르면 편집 모드로 전환됩니다
  - Cell을 탭하면 배경 색상이 바뀌며 Cell이 선택됩니다
    - 이미 선택된 Cell을 다시 탭하면 배경이 투명해지며 미선택 상태로 돌아갑니다
  - 삭제 버튼
    - 선택된 Cell이 없는 경우에는 삭제 버튼을 탭해도 아무런 동작을 수행하지 않습니다
    - 선택된 Cell이 하나 이상인 경우에는 Alert를 띄웁니다
    - 두 상태에 따라 다른 배경 색상에 유의합니다
  - 취소 버튼을 누르면 선택 내역을 초기화하고 기본 모드로 돌아갑니다



과제 상세 스펙 및 주의사항 (4)

- Alert
  - “삭제” 누르면 선택했던 Todo를 삭제합니다
  - “취소”를 누르면 편집 화면으로 되돌아갑니다. 선택했던 Todo 목록이 초기화되지 않습니다



과제 상세 스펙 및 주의사항 (5)

- Todo 추가 화면
  - 저장하지 않고 화면을 나가는 경우에는 작성 중이던 Todo가 초기화됩니다
  - 입력 중에는 X 버튼이 뜨도록 합니다
  - 자동으로 첫 글자를 대문자로 만드는 기능을 비활성화합니다
- Alert
  - “저장” 버튼을 눌렀을 때, Todo의 제목이 비어있는 경우에는 Alert를 띄웁니다
    - Todo의 제목이 비어있지 않다면, 새 Todo를 만들고 이전 화면으로 되돌아갑니다
    - 홈 화면에는 새로 추가된 Todo가 바로 보여야 합니다