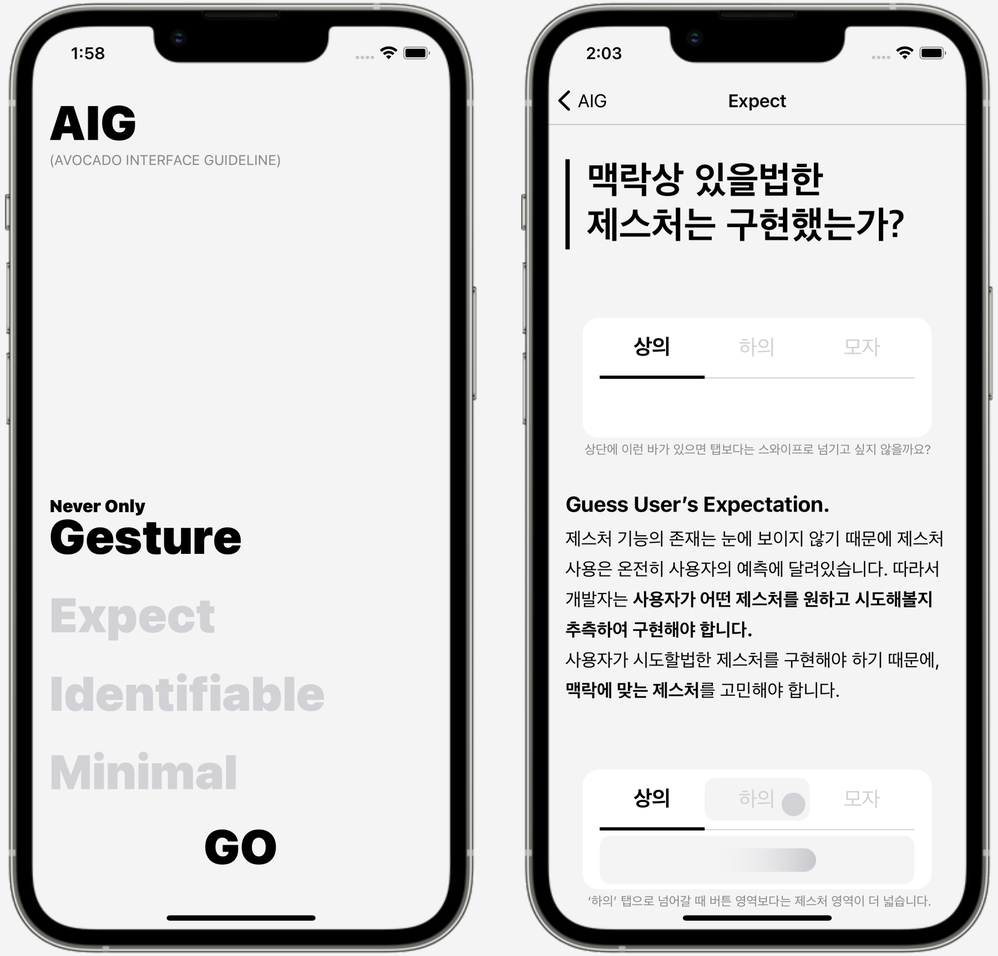
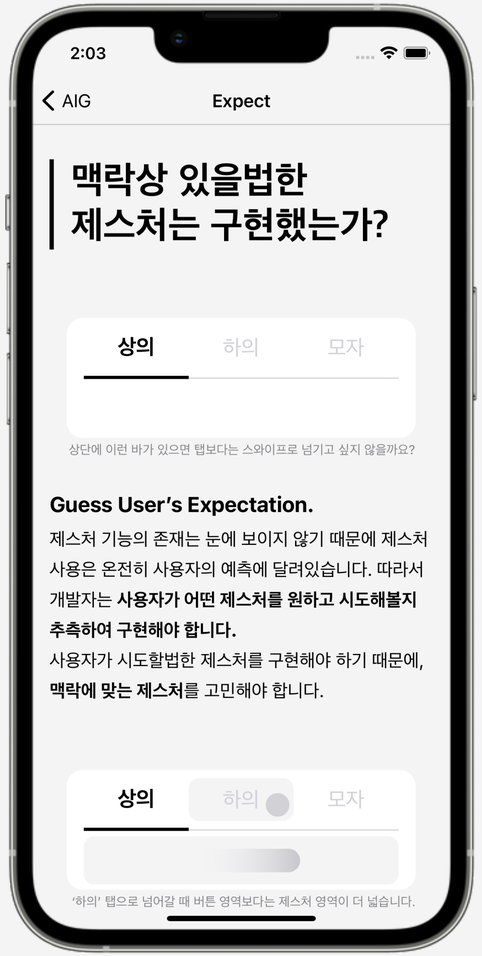
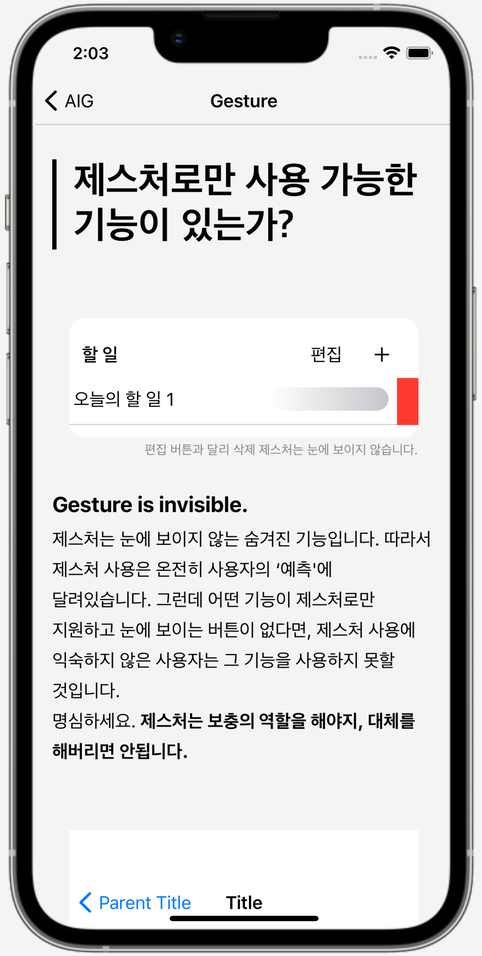
AIG Avocado Interface Guidelines.

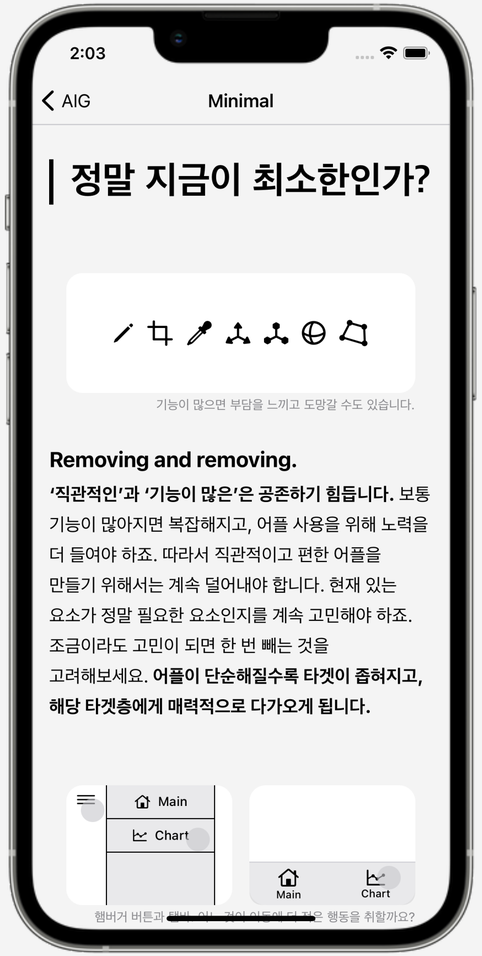


Avo (Chaeho Min, @chamin22)

My Essential Questions For Designing Apps.

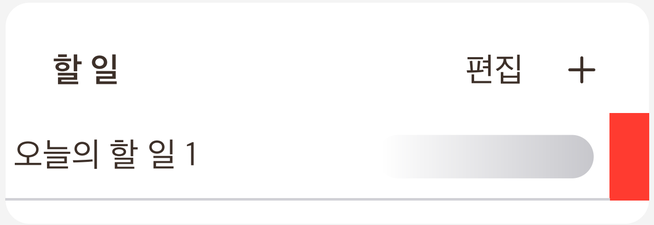






1. Never Only Gesture.

제스처로만 사용 가능한 기능이 있는가?



편집 버튼과 달리 삭제 제스처는 눈에 보이지 않습니다.

Gesture is invisible.

제스처는 눈에 보이지 않는 숨겨진 기능입니다. 따라서 제스처 사용은 온전히 사용자의 '예측'에 달려있습니다. 그런데 어떤 기능이 제스처로만 지원하고 눈에 보이는 버튼이 없다면, 제스처 사용에 익숙하지 않은 사용자는 그 기능을 사용하지 못할 것입니다.
명심하세요. **제스처는 보충의 역할을 해야지, 대체를 해버리면 안됩니다.**

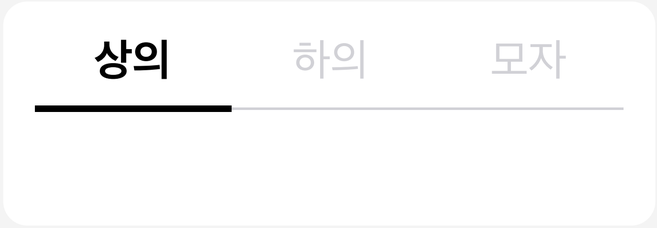
내비게이션바는 항상 제스처와 함께 뒤로가기 버튼을 제공합니다.

Always with visible.

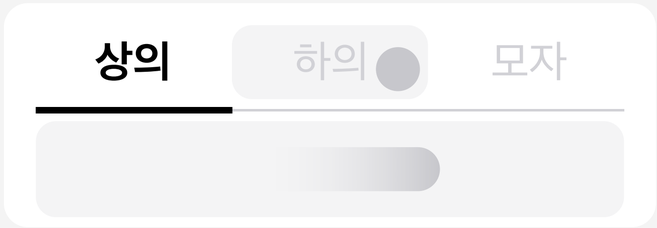
iOS 기본 요소는 모두 제스처와 동일한 액션을 하는 버튼을 제공합니다. 내비게이션바의 뒤로가기 버튼이나, 리스트의 편집 버튼 같이 말이죠. 생각해보세요, **리스트에 편집 버튼이 없다면 제스처를 모르는 사용자는 리스트를 어떻게 삭제할까요?**

2. Behaviors In Ways People Expect.

맥락상 있을법한 제스처는 구현했는가?



상단에 이런 바가 있으면 탭보다는 스와이프로 넘기고 싶지 않을까요?



‘하의’ 탭으로 넘어갈 때 버튼 영역보다는 제스처 영역이 더 넓습니다.

Guess User’s Expectation.

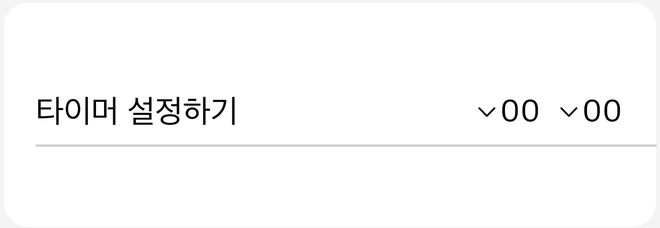
제스처 기능의 존재는 눈에 보이지 않기 때문에 제스처 사용은 온전히 사용자의 예측에 달려있습니다. 따라서 개발자는 **사용자가 어떤 제스처를 원하고 시도해볼지 추측하여 구현해야 합니다.**
사용자가 시도할법한 제스처를 구현해야 하기 때문에, **맥락에 맞는 제스처**를 고민해야 합니다.

Button vs Gesture.

그럼 버튼이 있는데 왜 제스처를 추가로 지원해야 할까요? **정확한 부분을 조준해서 탭해야 하는 버튼**과 달리 **제스처는 좀 더 넓은 영역을 조작**해도 되기 때문에 버튼과 제스처, 두 가지 방법을 제시하면 대부분의 사용자는 제스처를 택할 것입니다.
따라서 자주 탭하거나 누르기 어려운 위치에 있는 버튼은 **동일한 역할을 하는 제스처를 지원**하는 것을 추천합니다.

3. Features Must Be Identifiable.

요소의 역할을 명확히 인식할 수 있는가?



여기서 두 가지 설정은 '분/초'일까요 '시/분'일까요?

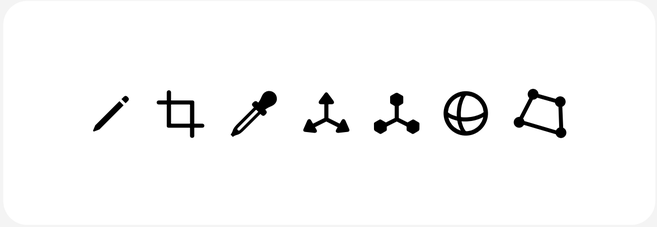
Be Intuitive. Or with description.

어플을 사용하려면 조작을 해야 합니다. 예를 들면 버튼을 눌러야 하거나, 설정값을 조정하거나, 직접 입력해야 하는 경우가 있죠. 그런데 **이 행위가 무엇을 위한 것인지 인지하기가 힘들다면 어떨까요?** 노력을 들여 예측을 하거나 시행착오를 겪어보는 등의 불편한 과정을 거쳐야 합니다. 이렇게 되면 사용자에게 어플이 불편하고 성의 없게 만들어졌다는 인상을 줄 수 있죠.

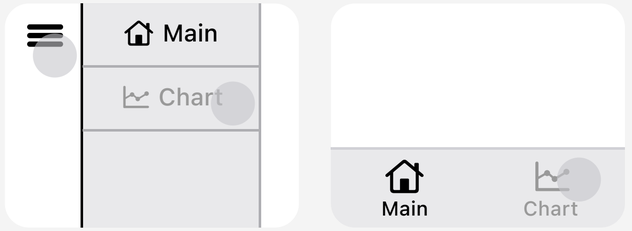
모든 사용자는 어플을 처음 사용하는 순간이 있다는 사실을 명심하세요. 나중에서야 어떤 기능을 사용하기 위해 특정 버튼을 누르는것이 당연할지라도, 처음 순간에는 그게 당연하지 않을 수 있습니다.
따라서 요소의 역할을 명확히 인식할 수 있도록 **직관적이거나, 최소한의 설명이 있어야 합니다.**

4. Keep It Minimal.

정말 지금이 최소한인가?



기능이 많으면 부담을 느끼고 도망갈 수도 있습니다.



햄버거 버튼과 탭바. 어느 것이 이동에 더 적은 행동을 취할까요?

Removing and removing.

‘직관적인’과 ‘기능이 많은’은 공존하기 힘듭니다. 보통 기능이 많아지면 복잡해지고, 어플 사용을 위해 노력을 더 들여야 하죠. 따라서 직관적이고 편한 어플을 만들기 위해서는 계속 덜어내야 합니다. 현재 있는 요소가 정말 필요한 요소인지를 계속 고민해야 하죠. 조금이라도 고민이 되면 한 번 빼는 것을 고려해보세요. **어플이 단순해질수록 타겟이 좁혀지고, 해당 타겟층에게 매력적으로 다가오게 됩니다.**

Minimize action steps.

기능의 개수 뿐만 아니라 기능을 사용하기 위해 행해야 하는 **행동의 횟수도 최소화**해야 합니다. 계속 탭해야 할 위치가 변하면, 정확히 조준해가며 탭하는게 매우 귀찮아집니다. 따라서 사용자를 최대한 덜 귀찮게 만들어야 합니다. 어플의 플로우를 쭉 따라가보면서 **‘이 행동이 굳이 필요한가? 더 줄일 수 있지 않을까?’**를 계속 고민해보는 것을 추천합니다.

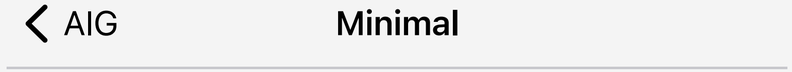
What I’ve Learned.

Clean Code…?

```swift
var body: some View {

    ZStack {

        Color.bgColor.ignoresSafeArea()

        VStack(spacing: 0) {

            title
            Spacer()
            categoryButtons
            categoryGOButton
        }
        .padding()
    }
}
```

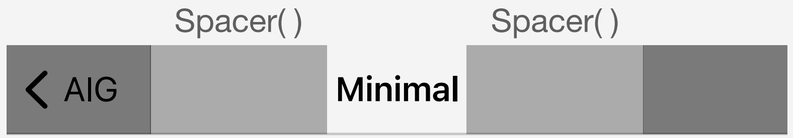
```swift
var body: some View {

    ZStack {

        Color.bgColor.ignoresSafeArea()

        VStack(spacing: 0) {
            CustomNavigationBar(title: guide.guideTitle)

            ScrollView(showsIndicators: false) {

                VStack(spacing: 64) {
                    question
                    content1
                    content2
                }
                .padding(.vertical, 32)
            }
            .ignoresSafeArea(edges: .bottom)
            .padding(.horizontal)
        }
    }
}
```

.frame + .infinity = greedy!!



.frame + .infinity = greedy!!

1. 똑같은 크기의 버튼을 비어있는 쪽에도 만들고 안 보이게 처리한다.



왼쪽 버튼과 똑같은 크기로 요소를 만들고 숨긴다.

버튼의 크기가 바뀌면 오른쪽에 숨겨둔 애도 같이 바꿔줘야 하는 번거로움이 있다!

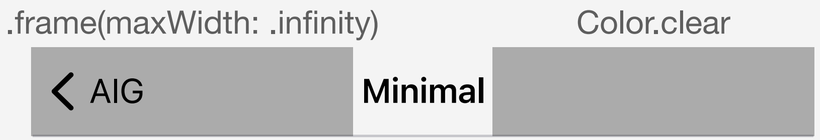
.frame + .infinity = greedy!!

2. ZStack으로 버튼과 타이틀을 나눠준다.

글자가 길어지거나 화면 너비가 좁아지면 글자가 겹칠 위험이 있다!

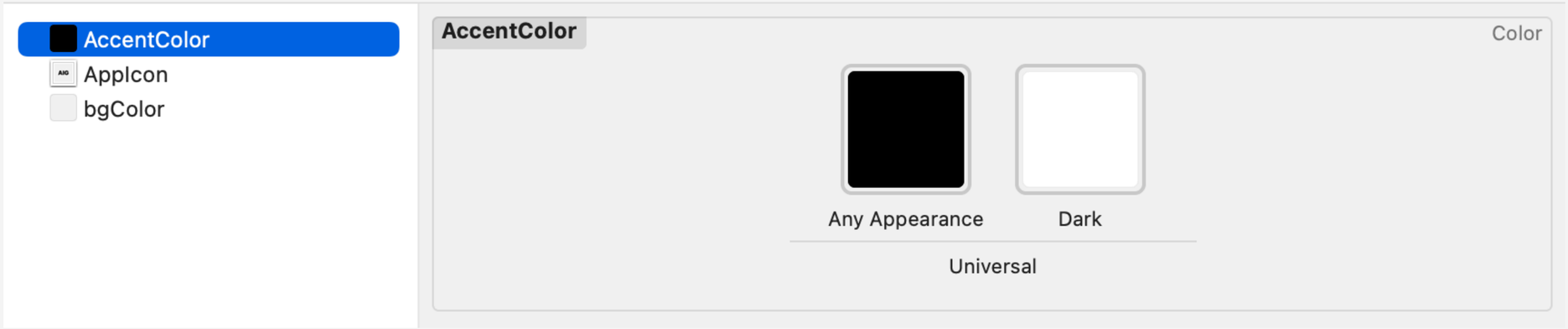
.frame + .infinity = greedy!!

3. .frame(maxWidth: .infinity)로 버튼 너비를 greedy로 만든다.



버튼 길이가 바뀌어도 그대로 타이틀은 가운데에 있고,
두 레이블이 겹칠 일도 없어졌다.

AccentColor in Assets



Thank You.