

SwiftUI: The basics

The shortest path to a great UI

- Présentation
- Composants
- Modificateurs
- Gestion des états

SwiftUI: The basics



SwiftUI: The basics



SwiftUI

- Framework UI déclaratif
- Disponible sur toutes les plateformes Apple
- Remplace ou complète UIKit / AppKit / WatchKit
- 100% natif

SwiftUI

- Compatible iOS 13+
- Encore en évolution, la connaissance de UIKit parfois nécessaire
- SwiftUI et UIKit sont compatibles entre eux
- Sûrement le futur de la conception d'UI

Présentation

```
import SwiftUI

struct Content : View {

    @State var model = Themes.listModel

    var body: some View {
        List(model.items, action: model.selectItem) { item in
            Image(item.image)
            VStack(alignment: .leading) {
                Text(item.title)
                Text(item.subtitle)
                    .color(.gray)
            }
        }
    }
}
```

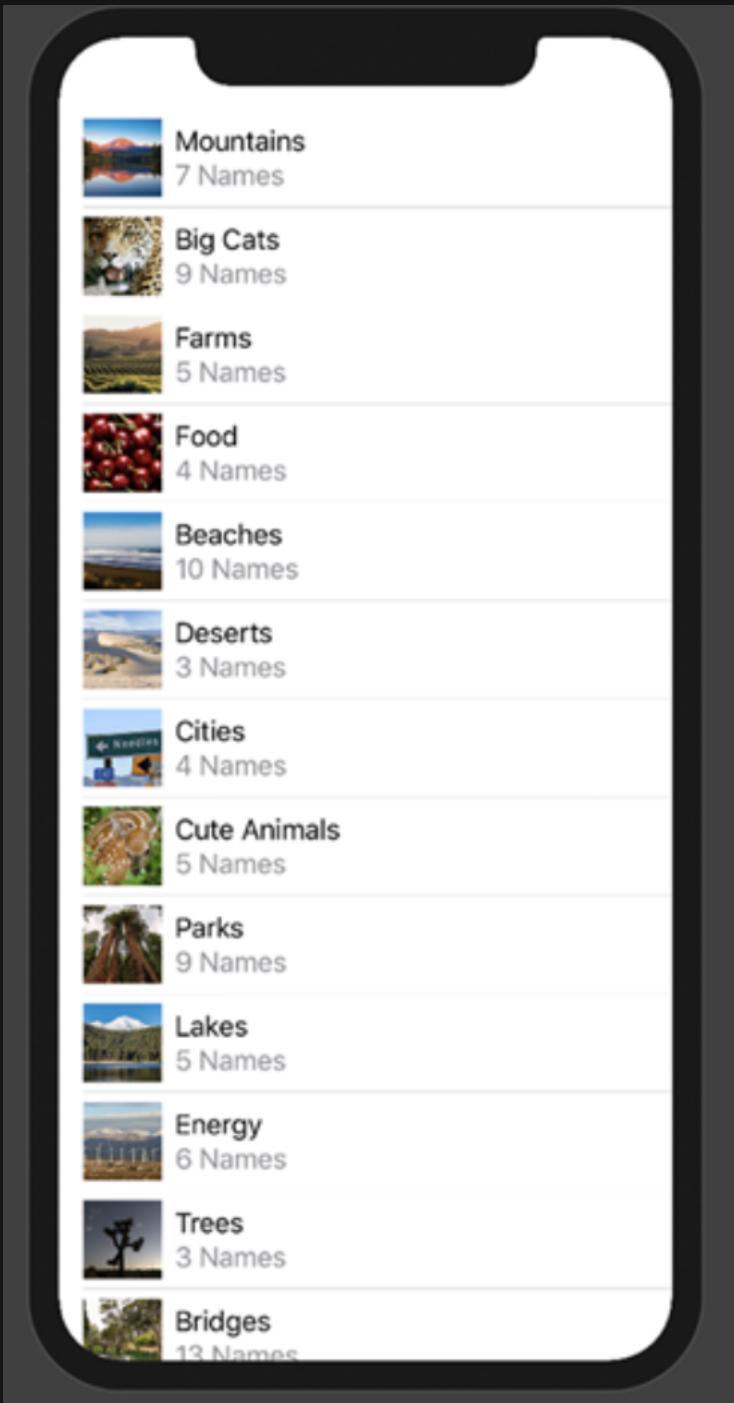
Présentation

```
import SwiftUI

struct Content : View {

    @State var model = Themes.listModel

    var body: some View {
        List(model.items, action: model.selectItem) { item in
            Image(item.image)
            VStack(alignment: .leading) {
                Text(item.title)
                Text(item.subtitle)
                    .color(.gray)
            }
        }
    }
}
```



Présentation

Write once, use everywhere

Présentation

Write once, use everywhere

Présentation

Learn once, use everywhere

Composants

- Les composants SwiftUI sont des structures
- Conformes au protocole View
- Beaucoup de composants existent en standard

Composants

- Les vues sont les briques de construction de votre UI
- En les combinant, on construit notre interface
- Avec SwiftUI, tout est "View"
 - Vues, contrôles, couleurs, formes

Composants

HStack

Image

ScrollView

Slider

ActionSheet

Section

NavigationView

Group

Alert

Divider

Stepper

Picker

SecureTextField

TextField

VStack

VSplitView

Form

List

ZStack

Text

Button

HSplitView

Toggle

GroupBox

NavigationLink

Spacer

DatePicker

Composants

NavLink
DatePicker
Slider
Button
Image
TextField
Text
Picker
SecureTextField
Stepper
Toggle

ScrollView
List
HStack
VStack
ZStack

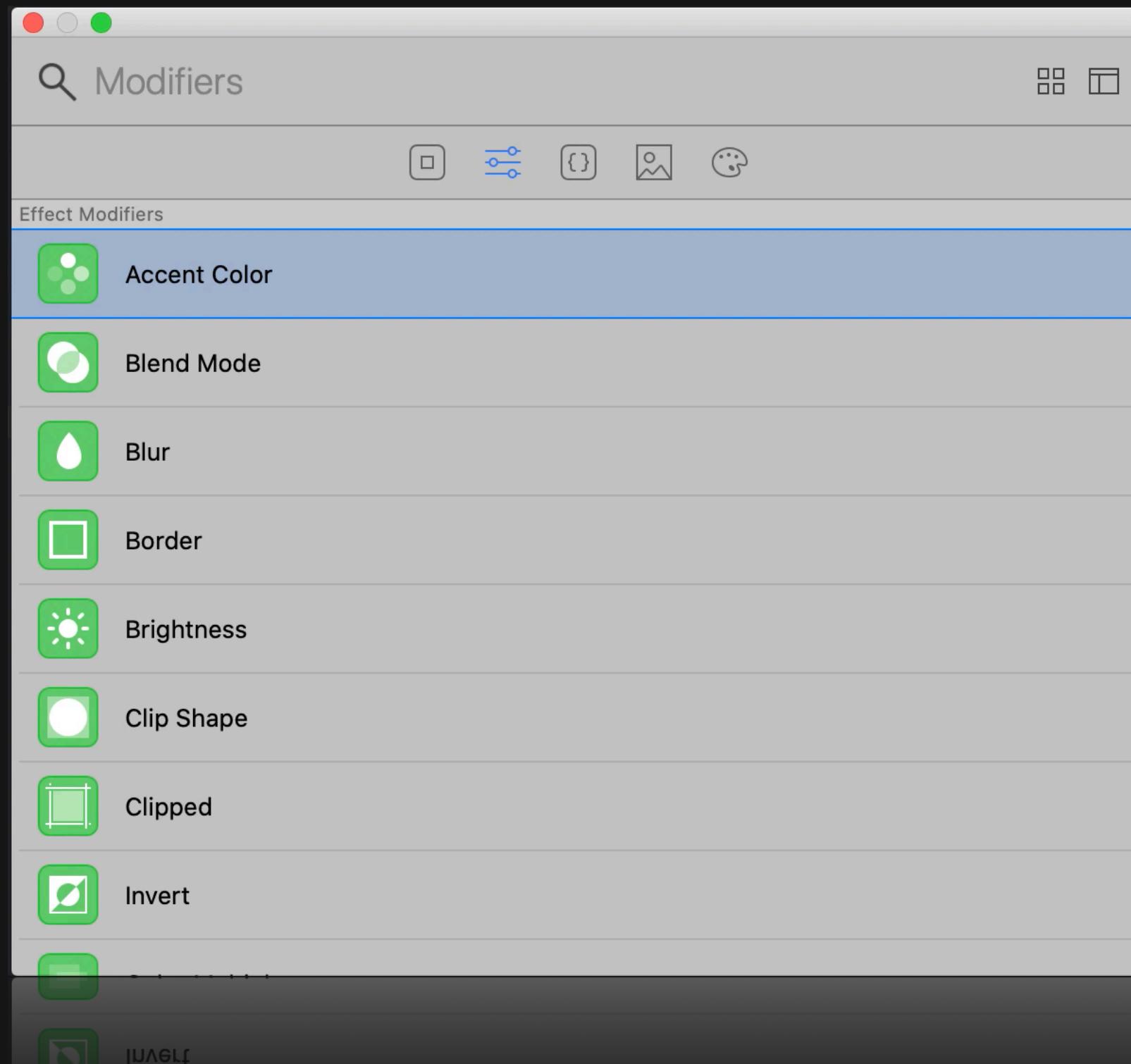
GroupBox
Group
Form
Section

TabView
NavigationView
VSplitView
HSplitView

ActionSheet
Alert

Divider
Spacer

Modificateurs

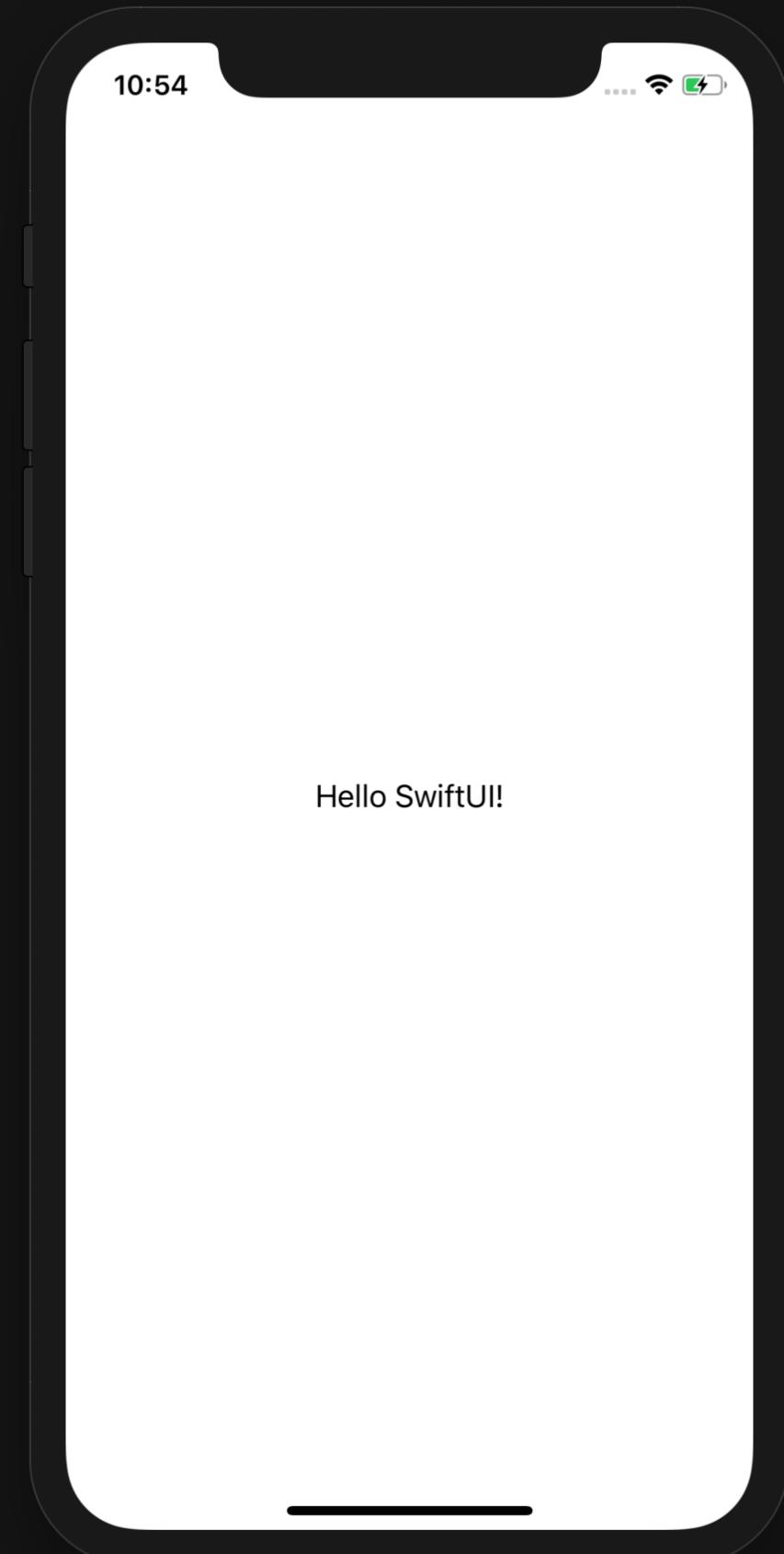


Modificateurs

- Les modificateurs sont des fonctions conçues pour modifier les vues
- La modification s'applique sur la vue sur laquelle elle est appelée, et retourne une nouvelle vue modifiée.
- L'ordre des modificateurs a un impact !

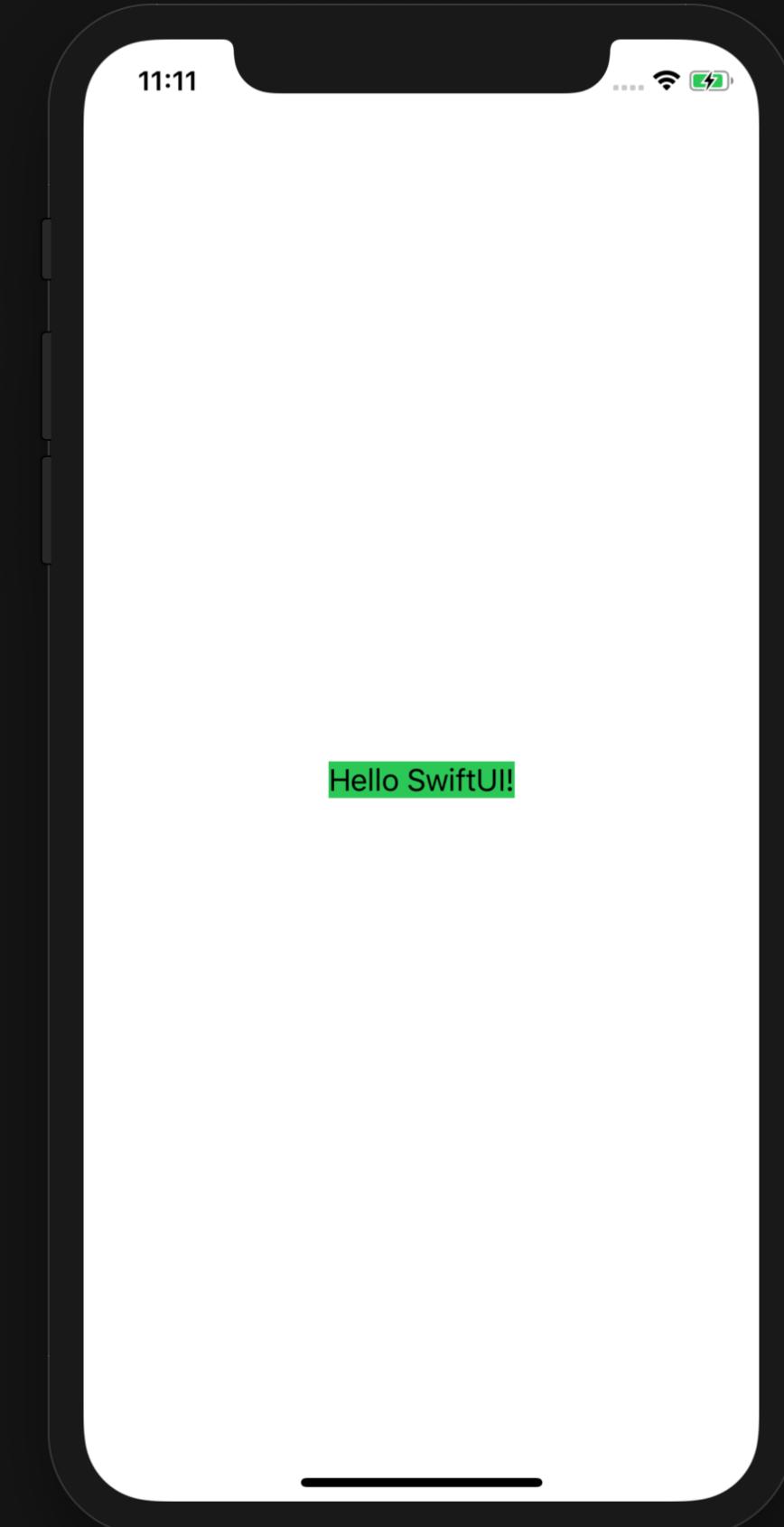
Modificateurs

```
struct ContentView: View {  
    var body: some View {  
        Text("Hello SwiftUI!")  
    }  
}
```



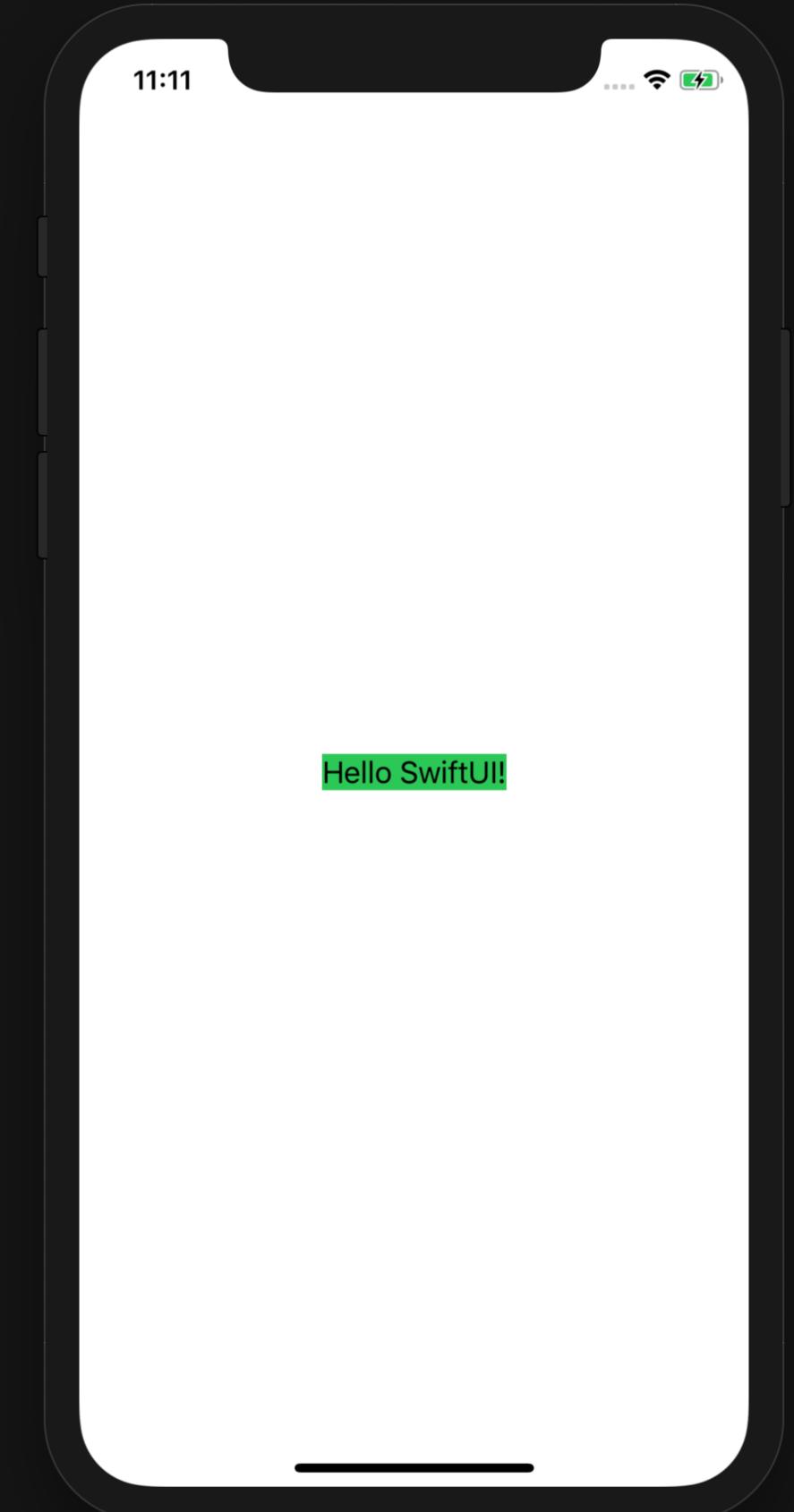
Modificateurs

```
struct ContentView: View {  
    var body: some View {  
        Text("Hello SwiftUI!")  
            .background(Color.green)  
    }  
}
```



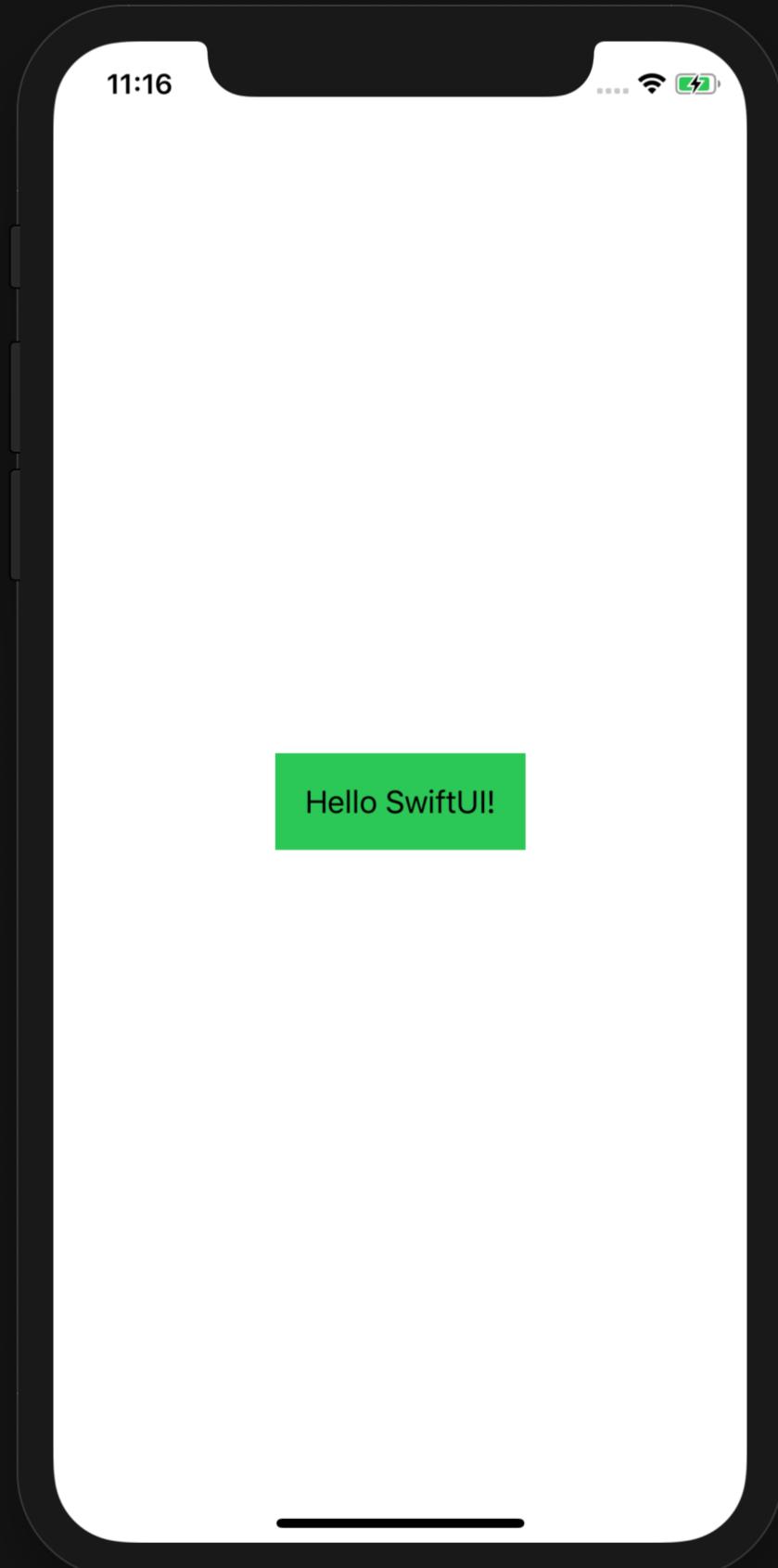
Modificateurs

```
struct ContentView: View {  
    var body: some View {  
        Text("Hello SwiftUI!")  
            .background(Color.green)  
            .padding()  
    }  
}
```



Modificateurs

```
struct ContentView: View {  
    var body: some View {  
        Text("Hello SwiftUI!")  
            .padding()  
            .background(Color.green)  
    }  
}
```



Gestion des états

- La gestion des états est inévitable dans les apps modernes
- Avec SwiftUI, les vues représentent un état
- On ne manipule pas la vue directement, on change son état
- Il existe plusieurs façon de définir l'état, et de passer des données suivant ce que l'on souhaite faire

Gestion des états

- **@State**

- Déclare une propriété "value type" qui participe à l'état d'une vue
- Met à jour la vue si la propriété change
- Généralement propre à la vue actuelle, donc **private**

```
struct ContentView: View {  
    @State private var name = "SwiftUI"  
  
    var body: some View {  
        Text(name)  
    }  
}
```

Gestion des états

- **@Binding**

- Utilisé pour déclarer une variable d'état qui est transmise d'un parent
- Permet de mettre à jour la variable dans le parent automatiquement

```
struct EditableTextView: View {  
    @State private var displayedText = "Hello, SwiftUI!"  
  
    var body: some View {  
        NavigationView {  
            VStack {  
                Text(displayedText)  
                NavigationLink("ChangeText",  
                    destination: TextEditingView(textToChange: $displayedText))  
            }  
        }  
    }  
  
    struct TextEditingView: View {  
        @Binding var textToChange: String  
  
        var body: some View {  
            TextField("Change text", text: $textToChange)  
        }  
    }  
}
```

Gestion des états

- ✿ **@StateObject**

- ✿ Déclare une propriété "complexe" qui participe à l'état d'une vue
- ✿ Met à jour la vue si la propriété change
- ✿ Généralement propre à la vue actuelle, donc **private**
- ✿ La propriété doit être conforme au protocole **ObservableObject**

```
struct ContentView: View {  
    @StateObject private var data = UserData()  
  
    var body: some View {  
        Text(data.name)  
    }  
}
```

Gestion des états

▀ @ObservedObject

- ❖ Déclare une propriété "complexe" qui participe à l'état
- ❖ Met à jour la vue si un élément important de la propriété change
- ❖ La propriété doit être conforme au protocole ObservableObject
- ❖ L'objet est en général reçu en paramètre

```
struct ContentView: View {  
    @ObservedObject var data: UserData  
  
    var body: some View {  
        VStack {  
            Text("\(data.score)")  
            if data.success {  
                Image(systemName: "checkmark.seal")  
            } else {  
                Image(systemName: "xmark.seal")  
            }  
        }  
    }  
}
```

Gestion des états

▪ ObservableObject

- Déclare un type observable
- Il faut informer quand un élément participant à l'état change
- Le moyen le plus simple :
@Published

```
class UserData: ObservableObject {  
  
    @Published var success = false  
    @Published var score = 0  
    @Published var name = "SwiftUI"  
}
```

Gestion des états

• `@EnvironmentObject`

- Déclare une variable qui se trouve dans l'environnement, transmises automatiquement aux vues enfants
- La variable d'environnement doit avoir été définie par une vue précédente, sinon crash
- Le type doit être conforme au protocole `@ObservableObject`

```
let contentView = ContentView().environmentObject(UserData())
```

```
struct ContentView: View {  
    @EnvironmentObject var data: UserData  
  
    var body: some View {  
        VStack {  
            Text("\(data.score)")  
            if data.success {  
                Image(systemName: "checkmark.seal")  
            } else {  
                Image(systemName: "xmark.seal")  
            }  
        }  
    }  
}
```

