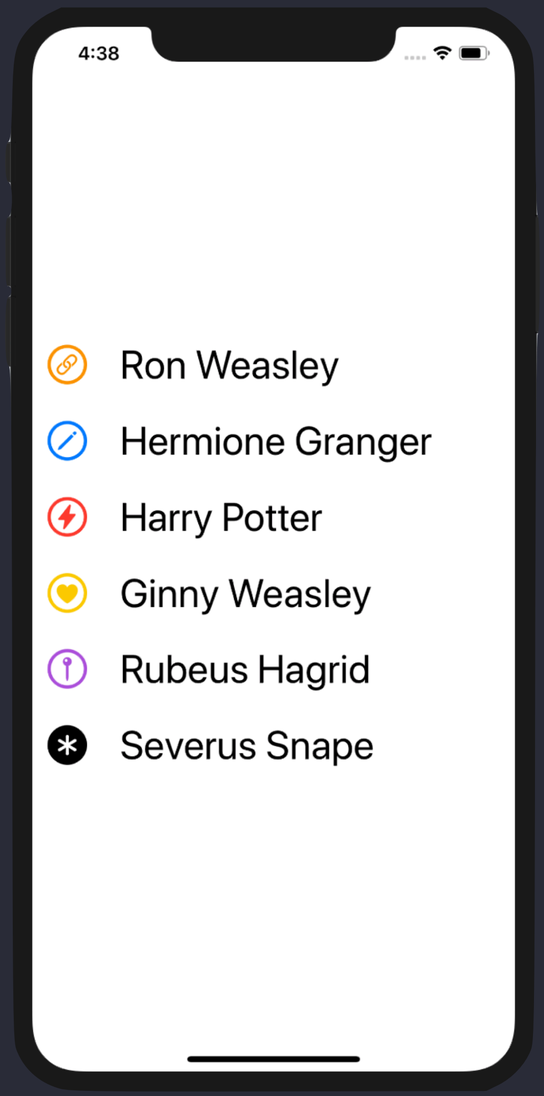
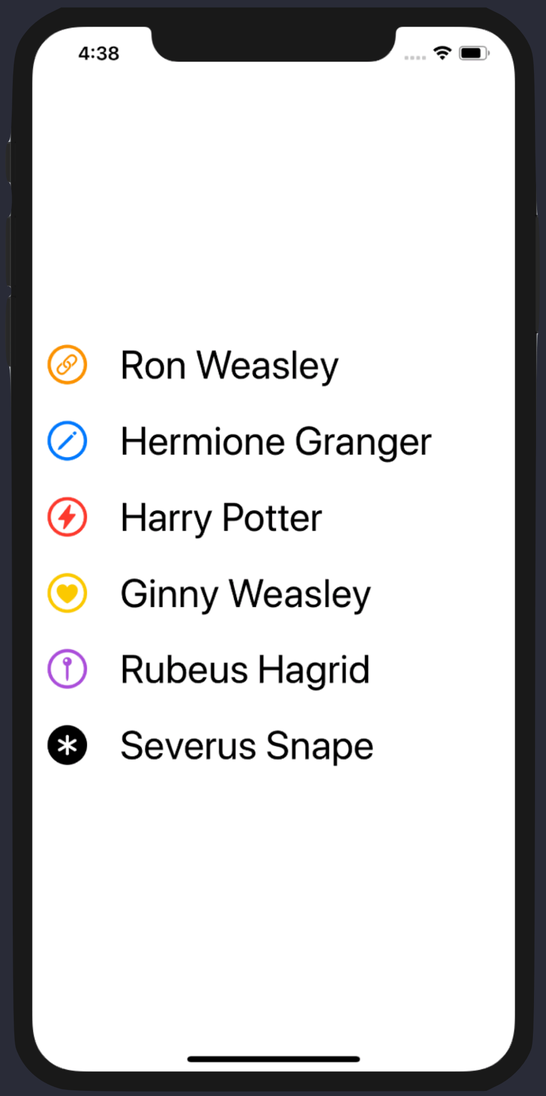
Extraction de vues

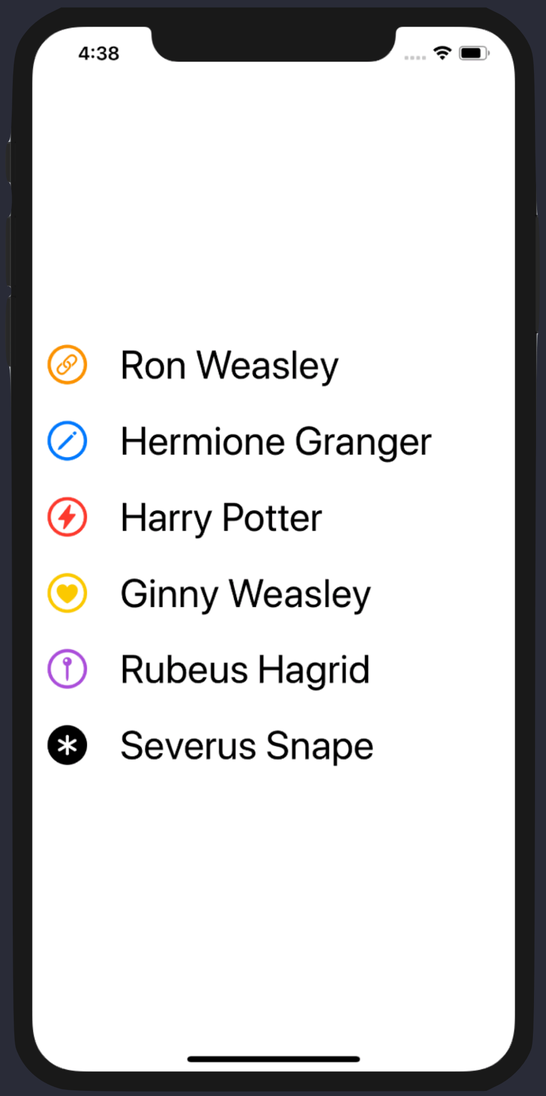
```
struct HeroList: View {
  var body: some View {
    VStack (alignment: .leading) {
      HStack {
        Image(systemName: "link.circle")
          .foregroundColor( .orange )
          .frame(width: 60)
        Text(" Ron Weasley")
      }.font(.largeTitle)
      HStack {
        Image(systemName: "pencil.circle")
          .foregroundColor( .blue )
          .frame(width: 60)
        Text(" Hermione Granger")
      }.font(.largeTitle)
      HStack {
        Image(systemName: "bolt.circle")
          .foregroundColor( .red )
          .frame(width: 60)
        Text(" Harry Potter")
      }.font(.largeTitle)
      HStack {
        Image(systemName: "heart.circle")
          .foregroundColor( .yellow )
          .frame(width: 60)
        Text(" Ginny Weasley")
      }.font(.largeTitle)
      HStack {
        Image(systemName: "mappin.circle")
          .foregroundColor( .purple )
          .frame(width: 60)
        Text(" Rubeus Hagrid")
      }.font(.largeTitle)
```



```swift
struct HeroList: View {
  var body: some View {
    VStack (alignment: .leading){
      HeroRow(icon: "link.circle", name: " Ron Weasley", color: .orange)
      HeroRow(icon: "pencil.circle", name: " Hermione Granger", color: .blue)
      HeroRow(icon: "bolt.circle", name: " Harry Potter", color: .red)
      HeroRow(icon: "heart.circle", name: " Ginny Weasley", color: .yellow)
      HeroRow(icon: "mappin.circle", name: " Rubeus Hagrid", color: .purple)
      HeroRow(icon: "asterisk.circle.fill", name: " Severus Snape", color: .black)
    }
  }
}

struct HeroRow: View {
  let icon: String
  let name: String
  let color: Color
  var body: some View {
      HStack {
        Image(systemName: icon)
          .foregroundColor(color)
          .frame(width: 60)
        Text(name)
      }.font(.largeTitle)
  }
}
```



```swift
struct HeroList: View {
  var body: some View {
    VStack (alignment: .leading){
      HeroRow(icon: "link.circle", name: " Ron Weasley", color: .orange)
      HeroRow(icon: "pencil.circle", name: " Hermione Granger", color: .blue)
      HeroRow(icon: "bolt.circle", name: " Harry Potter", color: .red)
      HeroRow(icon: "heart.circle", name: " Ginny Weasley", color: .yellow)
      HeroRow(icon: "mappin.circle", name: " Rubeus Hagrid", color: .purple)
      HeroRow(icon: "asterisk.circle.fill", name: " Severus Snape", color: .black)
    }
  }
}

struct HeroRow: View {
  let icon: String
  let name: String
  let color: Color
  var body: some View {
      HStack {
        Image(systemName: icon)
          .foregroundColor(color)
          .frame(width: 60)
        Text(name)
      }.font(.largeTitle)
  }
}
```

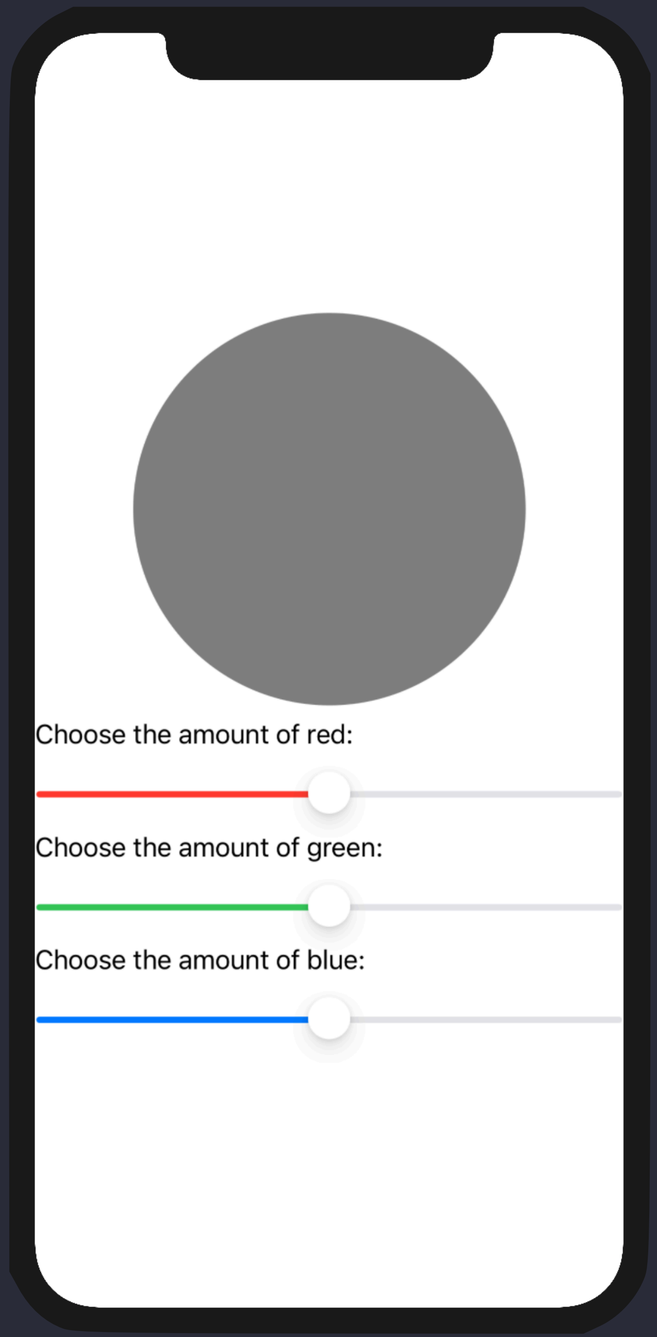


Pourquoi ? 🤔

- Éviter de dupliquer du code
- Plus facile à lire
- Plus facile à maintenir
- Moins de risque de bugs
- Réutilisabilité
  - Un composant complexe personnalisable avec quelques paramètres !

Two way binding

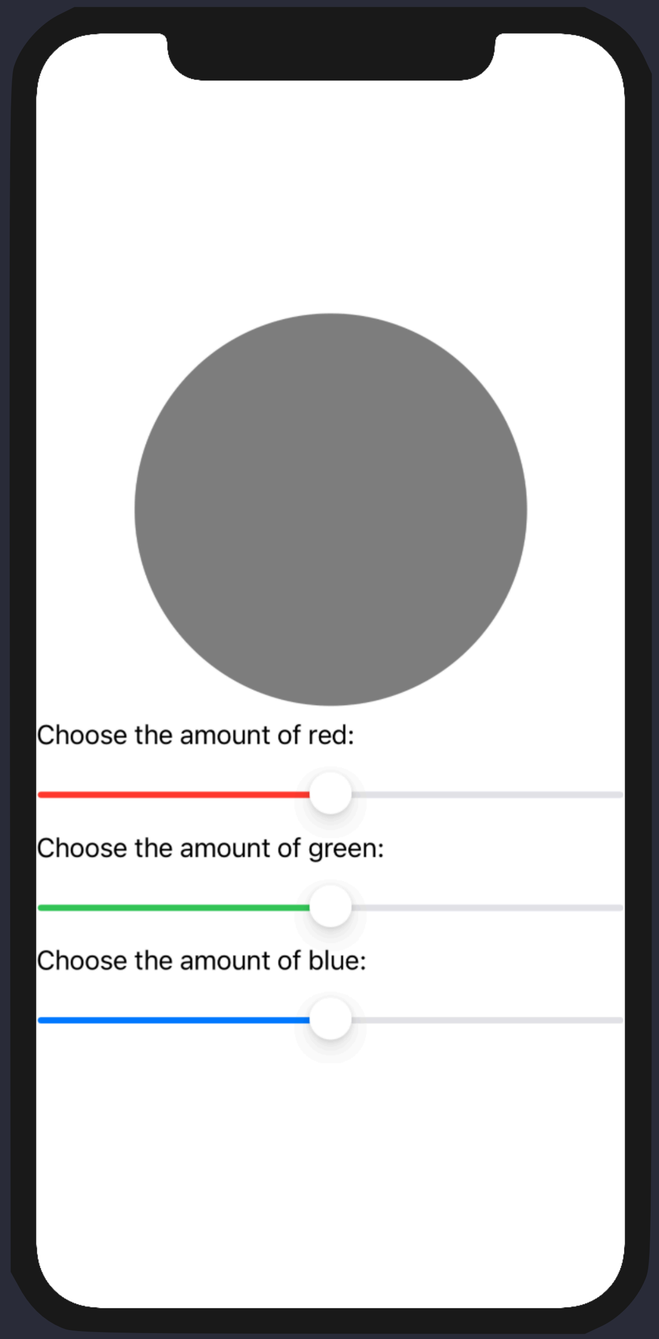
```swift
struct ColorPickerView: View {

    @State var red: Double = 0.5
    @State var green: Double = 0.5
    @State var blue: Double = 0.5

    var body: some View {
        VStack {
            Color(red: red, green: green, blue: blue, opacity: 1)
                .frame(width: 250, height: 250)
                .mask(Circle())
            VStack(alignment: .leading) {
                Text("Choose the amount of red:")
                Slider(value: $red, in: 0...1) {
                    Text("Red Slider")
                }
                .accentColor(.red)
            }
            VStack(alignment: .leading) {
                Text("Choose the amount of green:")
                Slider(value: $green, in: 0...1) {
                    Text("Green Slider")
                }
                .accentColor(.green)

            }
            VStack(alignment: .leading) {
                Text("Choose the amount of blue:")
                Slider(value: $blue, in: 0...1) {
                    Text("Blue Slider")
                }
                .accentColor(.blue)
            }
        }
    }
}
```



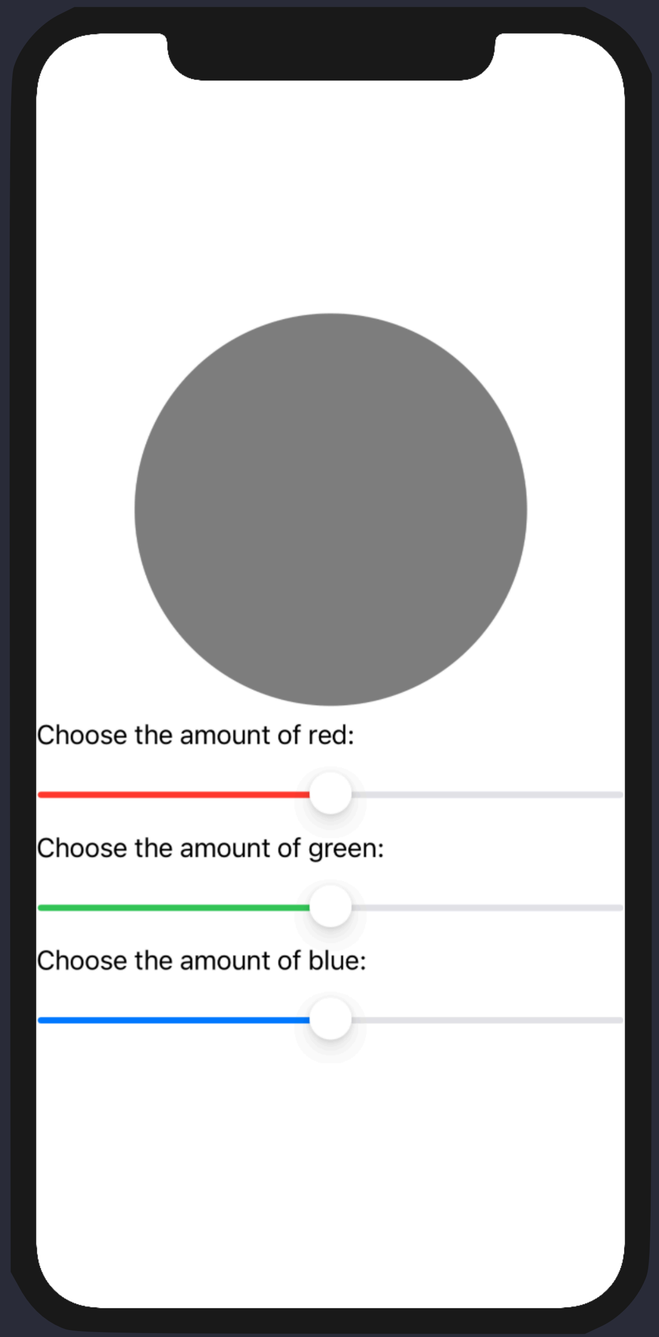
```swift
struct ColorPickerView: View {

    @State private var red: Double = 0.5
    @State private var green: Double = 0.5
    @State private var blue: Double = 0.5

    var body: some View {
        VStack {
            Color(red: red, green: green, blue: blue, opacity: 1)
                .frame(width: 250, height: 250)
                .mask(Circle())
            ColorPickerSlider(colorValue: red, colorName: "red", color: .red)
            ColorPickerSlider(colorValue: green, colorName: "green", color: .green)
            ColorPickerSlider(colorValue: blue, colorName: "blue", color: .blue)
        }
    }
}

struct ColorPickerSlider: View {

    var colorValue: Double
    let colorName: String
    let color: Color

    var body: some View {
        VStack(alignment: .leading) {
            Text("Choose the amount of \(colorName):")
            Slider(value: colorValue, in: 0...1) {
                Text("\(colorName.capitalized) Slider")
            }
            .accentColor(color)
        }
    }
}
```

Cannot convert value of type 'Double' to expected argument type 'Binding<Double>'

```swift
struct ColorPickerView: View {

    @State private var red: Double = 0.5
    @State private var green: Double = 0.5
    @State private var blue: Double = 0.5

    var body: some View {
        VStack {
            Color(red: red, green: green, blue: blue, opacity: 1)
                .frame(width: 250, height: 250)
                .mask(Circle())
            ColorPickerSlider(colorValue: $red, colorName: "red", color: .red)
            ColorPickerSlider(colorValue: $green, colorName: "green", color: .green)
            ColorPickerSlider(colorValue: $blue, colorName: "blue", color: .blue)
        }
    }
}

struct ColorPickerSlider: View {

    @Binding var colorValue: Double
    let colorName: String
    let color: Color

    var body: some View {
        VStack(alignment: .leading) {
            Text("Choose the amount of \(colorName):")
            Slider(value: $colorValue, in: 0...1) {
                Text("\(colorName.capitalized) Slider")
            }
            .accentColor(color)
        }
    }
}
```



```swift
struct ColorPickerView: View {

    @State private var red: Double = 0.5
    @State private var green: Double = 0.5
    @State private var blue: Double = 0.5

    var body: some View {
        VStack {
            Color(red: red, green: green, blue: blue, opacity: 1)
                .frame(width: 250, height: 250)
                .mask(Circle())
            ColorPickerSlider(colorValue: $red, colorName: "red", color: .red)
            ColorPickerSlider(colorValue: $green, colorName: "green", color: .green)
            ColorPickerSlider(colorValue: $blue, colorName: "blue", color: .blue)
        }
    }
}

struct ColorPickerSlider: View {

    @Binding var colorValue: Double
    let colorName: String
    let color: Color

    var body: some View {
        VStack(alignment: .leading) {
            Text("Choose the amount of \(colorName):")
            Slider(value: $colorValue, in: 0...1) {
                Text("\(colorName.capitalized) Slider")
            }
            .accentColor(color)
        }
    }
}
```



Résumé

- Ne pas avoir un gros paquet de code répétitif, mais extraire les composants
  - S’il y a un état qui se retrouve d’un côté et de l’autre, utiliser `@Binding`
- Extraire aussi les modifiers selon le même principe
- Permet d’avoir un code
  - plus propre
  - plus lisible
  - plus facile à modifier
  - moins de bugs