Desarrollo de Apps para iOS

SwiftUI

IWEB 2019-2020
Santiago Pavón

ver: 2019.11.25

Introducción

- Introducido con iOS 13 y Xcode 11.
- Desarrollo del UI de forma declarativa.
  - Declarar como es el UI.
  - No se implementa el UI proporcionando UIViews, no se usa Storyboard, no hay controladores, no hay autolayout
- Crear un proyecto: Seleccionar que el User Interface es SwiftUI.
- El canvas muestra un Preview del UI.
  - Editar en el código o en el canvas.
  - Actualizar el canvas: pulsar en Resume, vista en vivo o en un dispositivo real.
  - Command+Click sobre un elemento para ver el Inspector,
  - Usar el inspector de atributos: seleccionar vistas y modificadores

Repaso de Swift

- Inferencia de tipos.
- Tipos opacos: **some View**
- Struct es un tipo valor.
- Memberwise Initializator: Constructor de un Struct que asigna valores a las propiedades sin inicializar.

```
struct Persona {
    var nombre
    var edad
}
var p = Persona(nombre: "Ana", edad: 19)
```

- Trailing Closures: Si el último parámetro de una función es una closure, al invocar la función puede ponerse la expresión closure fuera de los paréntesis.

Demo 1

- Crear un proyecto son SwiftUI como tipo de interface.
- La vista creada es ContentView.
  - La propiedad body devuelve la vista a crear.
  - Solo devuelve una view.
  - Crear varias views y usar VStack para agruparlas.
- La clase ContentView_Previews implementa la(s) preview(s).
  - En la propiedad previews.
  - Cambiar por lo que se quiera.
  - Modificadores:

```
.previewLayout(.fixed(width: 100, height: 100))
.previewDevice(PreviewDevice(rawValue: "iPhone SE"))
.previewDisplayName(deviceName)
```

- El contenido de la UIWindow de la app se crea en SceneDelegate.
  - Cambiar en SceneDelegate la vista ContentView por la view que se quiera usar inicialmente.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, World!")
            Text("Adios")
            Button(action: {
                print("Hola")
            }) {
                Text("Demo")
            }
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ForEach(["iPhone SE", "iPhone XS Max"], id: \.self) { deviceName in
            ContentView()
                .previewLayout(.fixed(width: 250, height: 400))
                .previewDevice(PreviewDevice(rawValue: deviceName))
                .previewDisplayName(deviceName)
        }
    }
}
```

Mostrar el canvas en el editor y pulsar el botón Resume

Views

- Las views de SwiftUI son:
  - Text, Image
  - TextField, SecureField
  - Button, SegmentedControl, Stepper, Toggle, Slider
  - Picker, DatePicker
  - Spacer, Divider
  - VStack, HStack, ZStack, ScrollView
  - List, Form
  - NavigationView, TabView,
  - Alert, ActionSheet
  -

Modificadores

- Modificador: función que toma una view y devuelve otra view.
- De la clase base se heredan muchos modificadores comunes, y cada tipo de view también tiene sus propios modificadores.
- Ejemplos:
  - `.background(Color.blue), .foregroundColor(.red)`
  - `.font(.title) .fontWeight(.black)`
  - `.padding(),.padding(.bottom, 100)`
  - `.overlay(Circle().stroke(Color.gray, lineWidth: 2))`
  - `.clipShape(Circle())`
  - `.shadow(radius: 10)`
  - `.frame(width: 100),.frame(width: 100, height: 100)`
  - `.offset(x:10, y: -50)`
  - `.edgesIgnoringSafeArea(.top)`
  - `.rotationEffect(0.8, anchor: .bottom)`
  - `.onAppear { ... },.onDisappear { ... }`
  - `.disabled(true)`

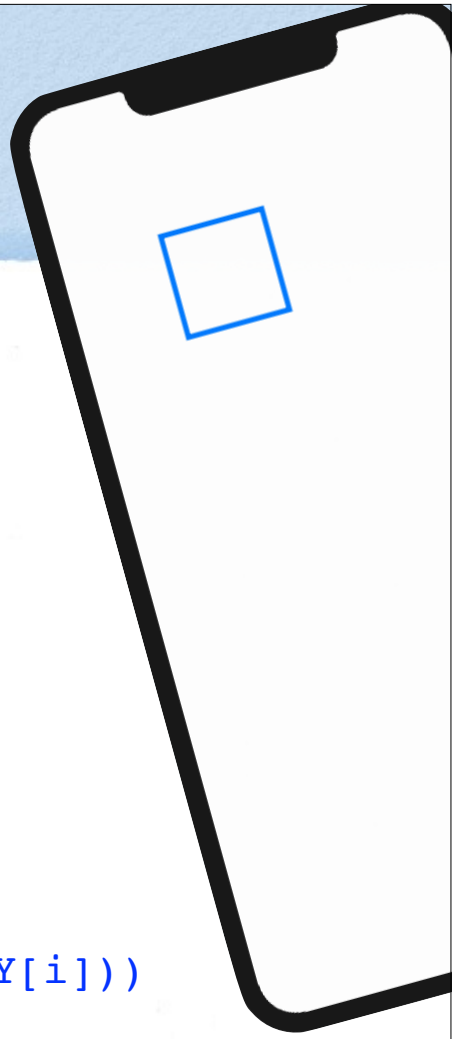
- El orden de aplicación de los modificadores importa.

```
Text("Hello, World!")
    .background(Color.blue)
    .padding()
Text("Hello, World!")
    .padding()
    .background(Color.blue)
```

- Imágenes:

```
Image("cara")
    .resizable()
    .scaledToFit()
    .background(Color.blue)
    .clipShape(Circle())
    .rotationEffect(Angle(degrees: 45))

Image(systemName: "square.and.arrow.up")
    .resizable()
```

Dibujar



```
struct ContentView: View {

    let pX = [ 100, 100, 200, 200]
    let pY = [ 100, 200, 200, 100]

    var body: some View {

        let path = Path { path in
            path.move(to: CGPoint(x: pX[0], y: pY[0]))
            for i in 1...3 {
               path.addLine(to: CGPoint(x: pX[i], y: pY[i]))
            }
            path.closeSubpath()
        }
        return path.stroke(Color.blue, lineWidth: 5)
    }
}
```

- Más:

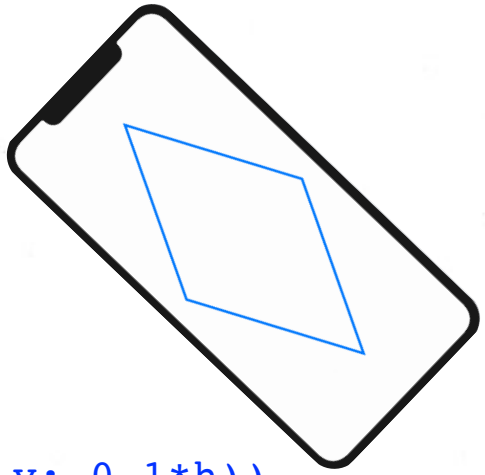
```
path.move(to: CGPoint)
path.addLine(to: CGPoint)
path.addLines([CGPoint])
path.addQuadCurve(to: CGPoint, control: CGPoint)
```

```
path
   .stroke(Color.blue, lineWidth: 5)
   .fill(Color.red)
   .overlay(path.stroke(Color.blue,
                        lineWidth: 5))
```

- Notas:
  - El modificador .stroke pinta el borde y devuelve un Path.
  - El modificador .fill rellena el interior, pero devuelve una View.
  - Hay que usar .stroke antes que .fill.

GeometryReader

- Wrapper de las views que permite acceder a la geometría de la view padre.



```
struct ContentView: View {

    var body: some View {
        GeometryReader { geometry in
            Path { path in
                let w = geometry.size.width
                let h = geometry.size.height

                path.move(to: CGPoint(x: 0.5*w, y: 0.1*h))
                path.addLine(to: CGPoint(x: 0.1*w, y: 0.5*h))
                path.addLine(to: CGPoint(x: 0.5*w, y: 0.9*h))
                path.addLine(to: CGPoint(x: 0.9*w, y: 0.5*h))

                path.closeSubpath()
            }
            .stroke(Color.blue, lineWidth: 5)
        }
    }
}
```

Shape

- Protocolo para crear un Path dentro de un rectángulo.

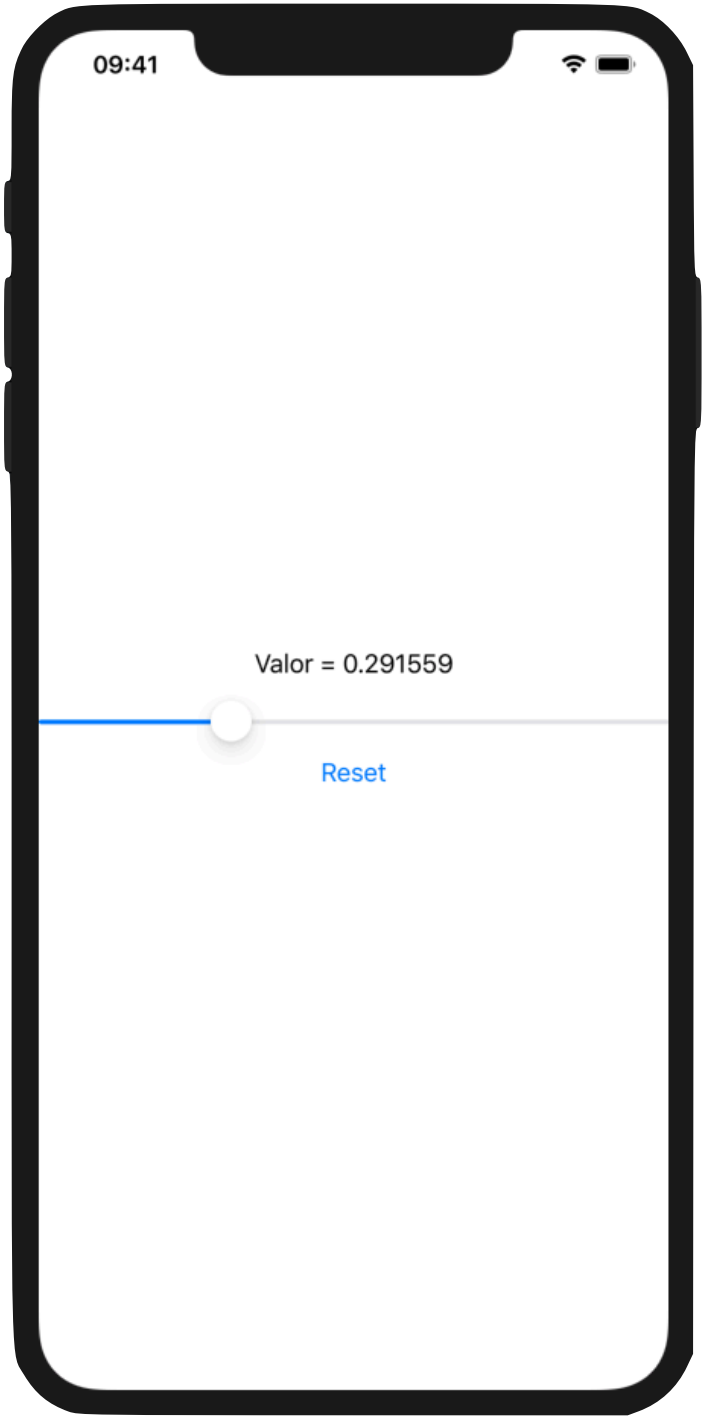
@State

- @State es un Property Wrapper que indica que una propiedad contiene parte del estado de la app.

```
@State var edad: Int = 0
```

- Si cambia el valor de la propiedad, entonces se refresca la UI.
- Los controles usan un binding a una propiedad de estado para actualizar su valor.
  - Usar el prefijo $ para crear un binding a la propiedad.
- Las propiedades de estado afectan al comportamiento de las views, a su contenido, su layout, ...

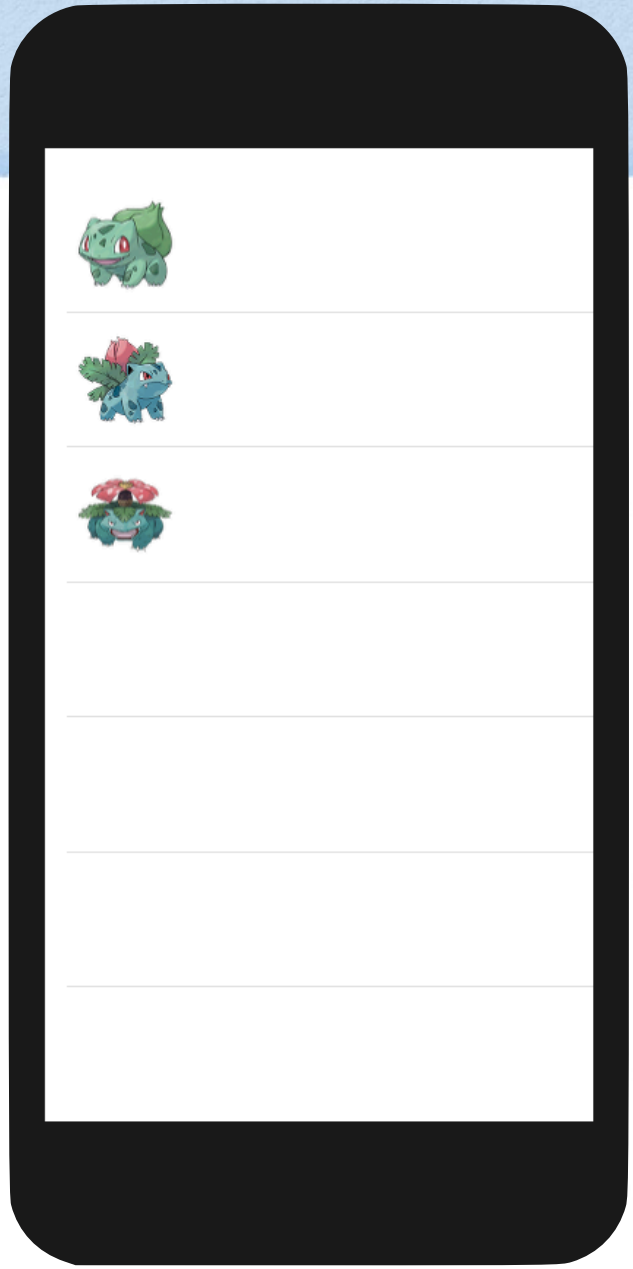
```
struct ContentView: View {

    @State var x = 0.5

    var body: some View {
        VStack {
            Text("Valor = \(x)")

            Slider(value: $x)

            Button(action: {
                self.x = 0
            }) {
                Text("Reset")
            }
        }
    }
}
```



List

- List es una lista de filas.
- Crearla pasando a List como parámetro una closure con las views.

```
struct ContentView: View {
    var body: some View {
        List {
            Image("001")
            Image("002")
            Image("003")
        }
    }
}
```



- Crear la lista pasando tres argumentos a List:
  - una colección de datos.
  - un KeyPath que identifique unívocamente cada dato de la colección.
  - una closure que devuelve una view para cada dato de la colección.

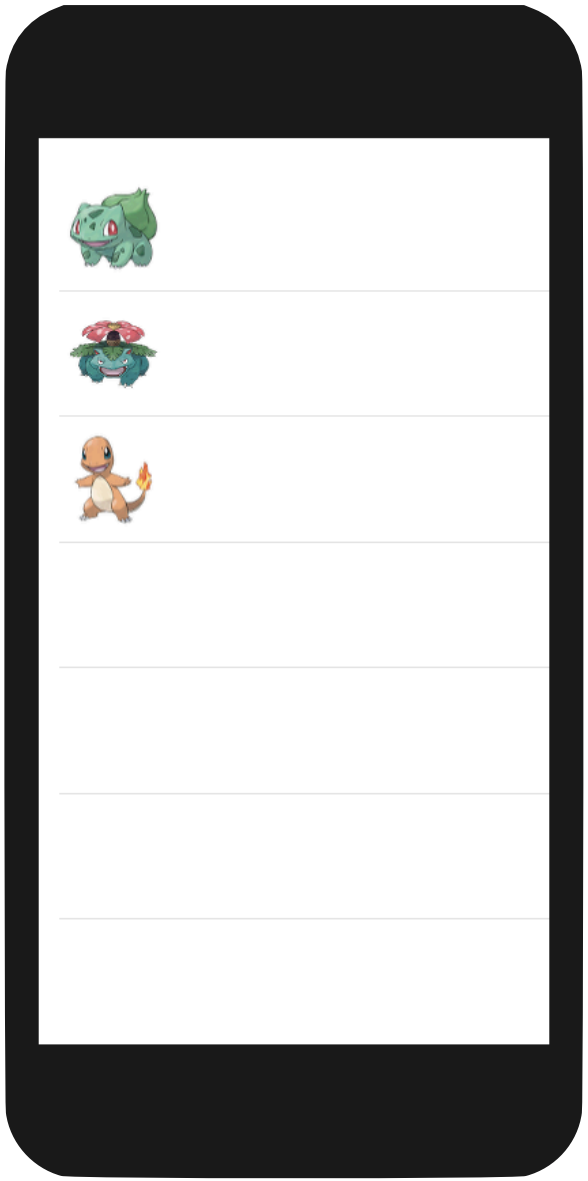
```
struct ContentView: View {

    var nombres = ["001", "002", "003"]

    var body: some View {
        List(nombres, id: \.self) { name in
            Image(name)
        }
    }
}
```

- Crear la lista pasando dos argumentos a List:
  - Si los elementos de la colección de datos son conformes con el protocolo **Identifiable**, puede omitirse el segundo argumento.
  - El protocolo **Identifiable** requiere que el tipo tenga una propiedad id que identifique las instancias.

```
struct Dato: Identifiable {
    var nombre: String
    var id: UUID
}

let datos = [
    Dato(nombre: "001" , id: UUID()),
    Dato(nombre: "002" , id: UUID()),
    Dato(nombre: "003" , id: UUID()),
]

struct ContentView: View {
    var body: some View {
        List(datos) { dato in
            Image(dato.nombre)
        }
    }
}
```

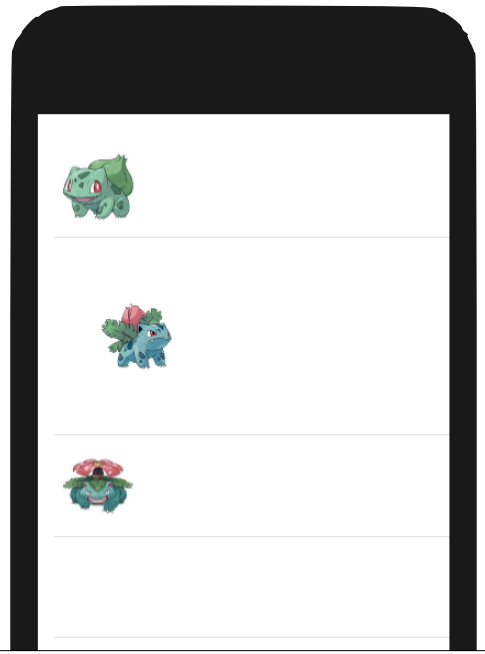
- ForEach
  - Se usa dentro de una List cuando las views a mostrar se eligen dinámicamente. o combinando diferentes tipos de views.
  - En el ejemplo se filtran algunos elementos de array.

```
struct Dato: Identifiable {
    var nombre: String
    var id: UUID
}

let datos = [
    Dato(nombre: "001" , id: UUID()),
    Dato(nombre: "002" , id: UUID()),
    Dato(nombre: "003" , id: UUID()),
    Dato(nombre: "004" , id: UUID()),
]

struct ContentView: View {

    var body: some View {
        List{
            ForEach(datos) { dato in
                if dato.nombre != "002" {
                    Image(dato.nombre)
                }
            }
        }
    }
}
```



- Insets
  - Puede asignarse un Insets a cada una de las views de la List.

```
struct ContentView: View {
    var body: some View {
        List {
            Image("001")
            Image("002")
                .listRowInsets(EdgeInsets(top: 50,
                                          leading: 50,
                                          bottom: 50,
                                          trailing: 50))
            Image("003")
        }
    }
}
```



- Secciones
  - Usar Section para crear secciones.
  - Usar el modificador .listStyle(GroupedListStyle()) para que el estilo de la lista sea de grupos.

```
List {
    ForEach(datos) {dato in

        Section(header: Text(dato.nombre)) {

            ForEach(dato.subdatos) { subdato in
               Text(subdato.nombre)
            }

        }
    }
}
.listStyle(GroupedListStyle())
```

NavigationView

- Navegación entre pantallas.
- Pasos:
  - Meter una List dentro de un NavigationView.
  - Añadir a la List el siguiente modificador para poner el título:

```
.navigationBarTitle(Text("Demo"))
```

  - Si se quiere que el titulo tenga letra pequeña:

```
.navigationBarTitle(Text("Demo"), displayMode: .inline)
```

  - Para poner views en la barra de navegación se añade el modificador a la List:

```
.navigationBarItems(trailing: unaView)
```

- Meter algunas (o todas) las filas de la List en un NavigationLink:

```
NavigationLink(destination: laViewDestino()) {
    Views_de_la_fila
}
```

  - Notas:
    - En un Navigation Link los textos se pintan con el color accent definido.
      - Para cambiarlo usar el modificador `.foregroundColor(.primary)`.
    - y las imágenes se pintan como una máscara.
      - Para pintarlas tal y como son originalmente, añadir el modificador `.renderingMode(.original)`.

```
struct ContentView: View {

    var pokedexModel = PokedexModel()

    var body: some View {
        NavigationView {
            List {
                ForEach(pokedexModel.types, id: \.name) {type in
                    Section(header: HStack {
                        Image(type.icon)
                            .resizable()
                            .frame(width: 50, height: 50)
                        Text(type.name)
                            .font(.largeTitle)
                    }) {
                        ForEach(type.races, id: \.code) { race in
                            NavigationLink(destination: RaceDetail(race: race)) {
                                ItemRace(race: race)
                            }
                        }
                    }
                }
            }
            .listStyle(GroupedListStyle())
            .navigationBarTitle("Pokedex")
        }
    }
}
```