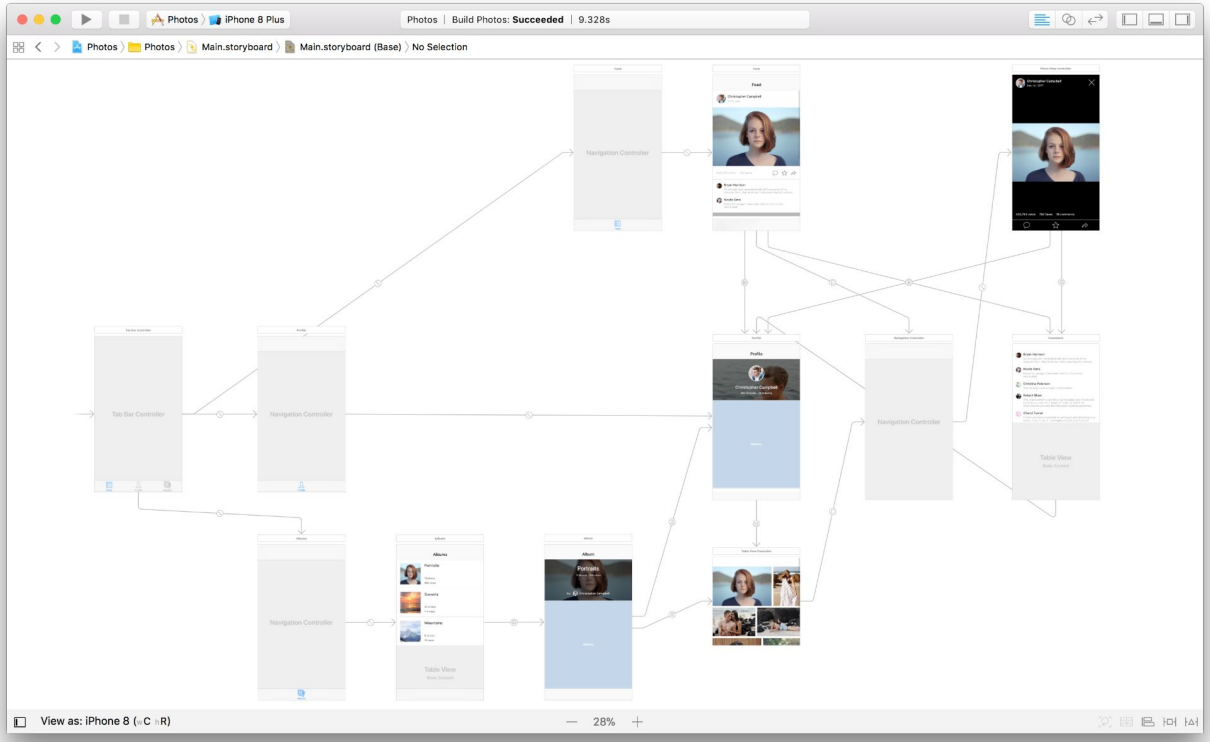
SwiftUI

iOS Developers

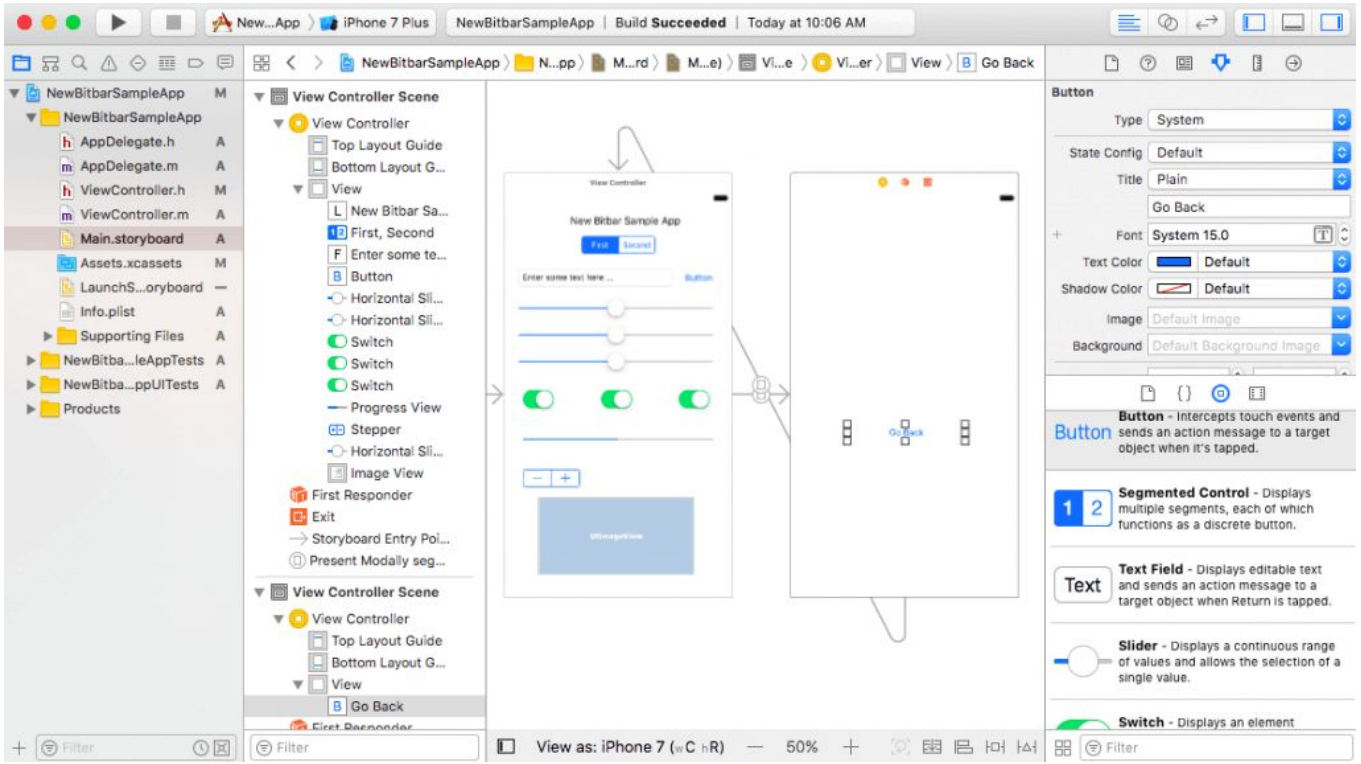
Contenido

- ¿Qué teníamos hasta el 2019?
- Programación declarativa.
- ¿Qué es SwiftUI?
- States y bindings
- Animaciones y Transiciones
- Listas
- Navegación
- Usar uikit en swift ui

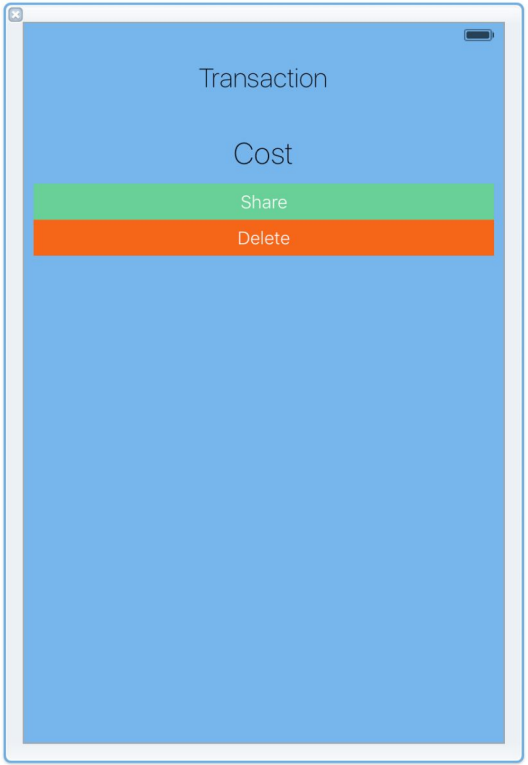
Hasta el 2019 - Storyboards



Hasta el 2019 - Storyboards (2)



Hasta el 2019 - Custom Views




```swift
//  ReturnView.swift
//  Manager
//
//
//
//

import UIKit

class ReturnView: UIView {

    override init(frame: CGRect) {
        super.init(frame: frame)
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    @IBOutlet var displayTransaction: UILabel!

    @IBOutlet var displayCost: UILabel!

    @IBOutlet var shareBtn: UIButton!

    @IBOutlet var deleteBtn: UIButton!

    @IBAction func shareItem(sender: AnyObject) {

        print("share")

    }

    @IBAction func deleteItem(sender: AnyObject) {

        print("delete")

    }

}
```

UIKit

“UIKit provides a variety of features for building apps, including components you can use to construct the core infrastructure of your iOS, iPadOS, or tvOS apps. The framework provides the window and view architecture for implementing your UI, the event-handling infrastructure for delivering Multi-Touch and other types of input to your app, and the main run loop for managing interactions between the user, the system, and your app. ”

Programación declarativa

“*La programación declarativa se enfoca en el resultado final, mientras que la programación imperativa se enfoca en cómo llegar allí*. *Por ejemplo, cuando te subes a un taxi, le declaras al conductor a dónde quieres ir. No le dices cómo llegar allí brindándole indicaciones paso a paso.*”

Programación declarativa

Imperativa

```
func filter(array: [Person], name: String) -> [Person] {
  var result = [Person]()
  for item in array {
    if item.name == name {
      result.append(item)
    }
  }
  return result
}
```

Programación declarativa

Declarativo

```
func filter(array: [Person], name: String) -> [Person] {
  return array.filter({ $0.name == name })
}
```

¿Qué es SwiftUI?

“SwiftUI helps you build great-looking apps across all Apple platforms with the power of Swift — and surprisingly little code. You can bring even better experiences to everyone, on any Apple device, using just one set of tools and APIs.”

SwiftUI

Better apps. Less code.

UIKit es imperativo

```swift
class ContentViewController: UIViewController, UITableViewDataSource {

  private var tableView: UITableView!

  var persons: [Person] = [
    Person(name: "SwiftUI"),
    Person(name: "UIKit")
  ]

  override func viewDidLoad() {
    super.viewDidLoad()
    setupTableView()
  }

  private func setupTableView() {
    tableView = UITableView()
    tableView.dataSource = self
    tableView.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(tableView)

    let topAndBottomMargens = CGFloat(16)
    NSLayoutConstraint.activate([
      tableView.topAnchor.constraint(equalTo: view.bottomAnchor, constant: topAndBottomMargens),
      tableView.centerXAnchor.constraint(equalTo: view.centerXAnchor),
      tableView.leadingAnchor.constraint(equalTo: view.leadingAnchor)
    ])
  }

  func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return persons.count
  }

  func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let personCell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath) as! PersonTableViewCell
    personCell.textLabel?.textColor = .gray
    personCell.textLabel?.text = persons[indexPath.row].name
    return personCell
  }
}
```

SwiftUI es declarativo

```
struct ContentView: View {

  var persons: [Person] = [
    Person(name: "SwiftUI"),
    Person(name: "UIKit")
  ]

  var body: some View {
    List(persons) { person in
      Text(person.name)
      .font(.  )
      .foregroundColor(.gray)
    }
  }

}
```

States

El estado es inevitable en cualquier aplicación moderna, pero con SwiftUI es importante recordar que todas nuestras vistas son simplemente funciones de su estado: no cambiamos las vistas directamente, sino que manipulamos el estado y dejamos que dicte el resultado.

SwiftUI nos brinda varias formas de almacenar el estado en nuestra aplicación, pero son sutilmente diferentes y es importante comprender cómo son diferentes para usar el marco correctamente.

SwiftUI administra el almacenamiento de una propiedad que declaras como estado. Cuando el valor cambia, SwiftUI actualiza las partes de la jerarquía de vistas que dependen del valor.

States (2)

```
struct ContentView: View {
    @State private var tapCount = 0

    var body: some View {
        Button("Tap count: \(tapCount)") {
            tapCount += 1
        }
    }
}
```

State Objects

Es una property wrapper para referenciar objetos complejos dentro de una view.

```
class User: ObservableObject {
    var username = "@user"
}
```

```
struct ContentView: View {
    @StateObject var user = User()

    var body: some View {
        Text("Username: \(user.username)")
    }
}
```

Binding

@Binding, nos permite conectar una propiedad @State de una vista a algunos datos del modelo subyacente.

Si pasa una propiedad de estado a una vista secundaria, SwiftUI actualiza la vista secundaria cada vez que cambia el valor en la principal, pero la secundaria no puede modificar el valor. Para permitir que la vista secundaria modifique el valor almacenado, pase un Binding en su lugar. Puede obtener un enlace a un valor de estado accediendo al valor proyectado del estado, que obtiene anteponiendo el nombre de la propiedad con un signo de dólar ($).

Binding (2)

```
@State private var rememberMe = false

var body: some View {
    Toggle("Remember Me", isOn: $rememberMe)
}
```

Sources of truth

Los property wrappers que poseen sus datos son fuentes de la verdad porque crean y administran el valor,

- @AppStorage
- @FetchRequest
- @GestureState
- @Namespace
- @NSApplicationDelegateAdaptor
- @Published
- @ScaledMetric
- @SceneStorage
- @State
- @StateObject
- @UIApplicationDelegateAdaptor

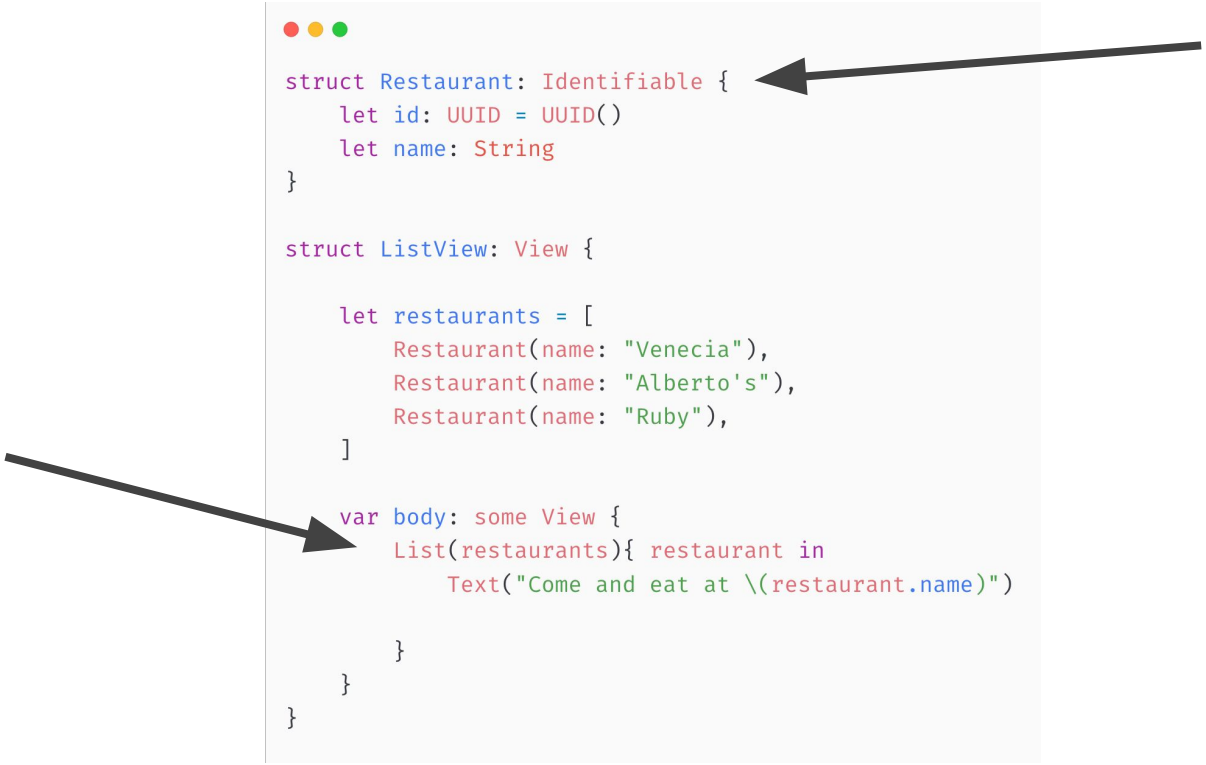
Property wrappers que no son sources of truth

- @Binding
- @Environment
- @EnvironmentObject
- @FocusedBinding
- @FocusedValue
- @ObservedObject

Animaciones y Transiciones

Example here

Listas



```
struct Restaurant: Identifiable {
    let id: UUID = UUID()
    let name: String
}

struct ListView: View {

    let restaurants = [
        Restaurant(name: "Venecia"),
        Restaurant(name: "Alberto's"),
        Restaurant(name: "Ruby"),
    ]

    var body: some View {
        List(restaurants){ restaurant in
            Text("Come and eat at \(restaurant.name)")

        }
    }
}
```

Navegación

```
struct BasicNavigationView: View {
    var body: some View {
        NavigationView {
            NavigationLink(destination: Text("Second View")) {
                Text("Hello, World!")
            }
            .navigationTitle("Navigation")
        }
    }
}
```

Usar UIKit en SwiftUI

Aunque SwiftUI hace un buen trabajo al proporcionar muchas de las subclases UIView de UIKit, pero todavía no las tiene todas en este momento. Afortunadamente, no es difícil crear contenedores personalizados para una UIView que desee.

Como ejemplo, vamos a crear un contenedor SwiftUI simple para **UITextView** como base de un editor de texto enriquecido. Esto toma cuatro pasos:

1. Crear una estructura que se ajuste a **UIViewRepresentable**.
2. Definición de una propiedad que almacena la cadena de texto con la que estamos trabajando.
3. Dándole un método **makeUIView()** que devolverá nuestra vista.
4. Agregar un método **updateUIView()** que se llamará cada vez que cambien los datos de la vista.

Usar UIKit en SwiftUI (2)

```swift
struct TextView: UIViewRepresentable {
    @Binding var text: NSMutableAttributedString

    func makeUIView(context: Context) -> UITextView {
        UITextView()
    }

    func updateUIView(_ uiView: UITextView, context: Context) {
        uiView.attributedText = text
    }
}
```

Usar UIKit en SwiftUI (2)

```
struct ContentView: View {
    @State var text = NSMutableAttributedString(string: "")

    var body: some View {
        TextView(text: $text)
            .frame(minWidth: 0, maxWidth: .infinity, minHeight: 0, maxHeight: .infinity)
    }
}
```

Gracias

SwiftUI

Better apps. Less code.

Repositorio de Ejemplos