iOS Application Development

Lecture 11: SwiftUI Continued

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

WS '22/'23 • hci.rwth-aachen.de/ios
Layout Views
Layout Behavior

- Two behaviors of views

Text views pull in

Colors are Push-Out views
VStack, HStack & ZStack

• Pull-in container views

• Like all layout views, can only contain up to 10 items
  • Group{} to create larger lists

• LazyVStack & LazyHStack load items only if visible
  • Push-out containers

```swift
// Replace VStack with HStack or HStack here
VStack(alignment: .center, spacing: 5) {
    Text(" ")
        .frame(width: 150)
        .background(.blue)
    Text(" ")
        .frame(width: 75)
        .background(.green)
    Text(" ")
        .frame(width: 37)
        .background(.orange)
}
```
List

• Vertically scrolling list
• Very efficient
• Items can be different View types
• Multiple selection, Delete via swipe optional
• Group{} helps with more than 10 items
• Section{} creates a separated group with an optional title

```swift
struct ContentView: View {
    var body: some View {
        List {
            Text("Hello, Aachen!")
            Text("Hello, Bremen!")
            Text("Hello, Zurich!")
        }
    }
}
```
Form

• Similar to List

• Both use UITableView in iOS

• Use Form to group controls used for data entry, like in settings or inspectors

• Used, e.g., in the iOS Settings app
struct ContentView: View {
    var body: some View {
        HStack {
            Spacer()
            VStack(alignment: .leading) {
                Text("Name")
                    .font(.largeTitle)
                    .bold()
                Text("Isabelle")
                Text("Oscar")
                Text("Sara")
            }
            Spacer(minLength: 150)
            VStack(alignment: .leading) {
                Text("Age")
                    .font(.largeTitle)
                    .bold()
                Text("19")
                Text("22")
                Text("25")
            }
            Spacer()
        }
    }
}
Layers

- Add layers on top and behind views

```swift
struct ContentView: View {
    var body: some View {
        Image("pigeons")
            .frame(maxWidth: .infinity)
            .opacity(0.6) // Make image only 60% solid
            .background(Color.red.opacity(0.4)) // Layer behind image
            .background(Color.yellow.opacity(0.4)) // Layer behind red
            .background(Color.blue.opacity(0.4)) // Layer behind yellow
            .overlay(
                Text("Pigeons")
                    .foregroundColor(.white)
                    .font(.largeTitle)
            ) // Layer on top of image
            .clipped()
    }
}
```
MVVM: Model / View / ViewModel

- MVC Pros: Separation of concerns, wide adoption, clear data flow
- MVC Cons: "MVC = Massive ViewController", ViewControllers hard to test (UI)
- Solution: Presentation Model [Fowler 2004]: Store app state (ViewModel) independently of UI (View)
- But requires tons of glue code between ViewModel and View
- Solution: Bindings (like Cocoa Bindings in Mac OS X since 2003, but not iOS)
- Adopted by Microsoft's Windows Presentation Foundation in 2005 as "MVVM"
- Natural choice for SwiftUI, testability, smaller Views
"MVVM is the modern improvement of MVC for declarative programming"
Apps on iOS and macOS

Prof. Dr. Jan Borchers: iOS Application Development

View
AppKit

Controller
AppKit

Model
Foundation, CoreData

View
UIKit

Controller
UIKit
# SwiftUI View Names

<table>
<thead>
<tr>
<th>UIKit</th>
<th>SwiftUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>UITableViewCell</td>
<td>List</td>
</tr>
<tr>
<td>UICollectionView</td>
<td>No direct equivalent (alternatives: LazyVStack, LazyHStack,...)</td>
</tr>
<tr>
<td>UILabel</td>
<td>Text</td>
</tr>
<tr>
<td>UITextField</td>
<td>TextField</td>
</tr>
<tr>
<td>UITextView</td>
<td>TextEditor</td>
</tr>
<tr>
<td>UISwitch</td>
<td>Toggle</td>
</tr>
<tr>
<td>UISlider</td>
<td>Slider</td>
</tr>
<tr>
<td>UIButton</td>
<td>Button</td>
</tr>
<tr>
<td>UINavigationController</td>
<td>NavigationView (softly deprecated → NavigationStack)</td>
</tr>
<tr>
<td>UIAlertController with style .alert</td>
<td>Alert</td>
</tr>
<tr>
<td>UIAlertController with style .actionSheet</td>
<td>ActionSheet</td>
</tr>
<tr>
<td>UIStackView with horizontal axis</td>
<td>HStack</td>
</tr>
<tr>
<td>UIStackView with vertical axis</td>
<td>VStack</td>
</tr>
<tr>
<td>UIImageView</td>
<td>Image</td>
</tr>
<tr>
<td>UISegmentedControl</td>
<td>SegmentedControl</td>
</tr>
<tr>
<td>UIStepper</td>
<td>Stepper</td>
</tr>
<tr>
<td>UIDatePicker</td>
<td>DatePicker</td>
</tr>
<tr>
<td>NSAttributedString</td>
<td>Incompatible with SwiftUI; use Text instead.</td>
</tr>
</tbody>
</table>

The SwiftUI View Names table lists the UIKit views and their SwiftUI equivalents. Some UIKit views do not have a direct SwiftUI equivalent, while others have alternatives. Views that are deprecated in UIKit are not included in SwiftUI.
Variables & Loops in SwiftUI
Variables in SwiftUI

```swift
struct ContentView: View {
    var tapCount = 0

    var body: some View {
        VStack {
            Spacer()
            Spacer()
            Text("\(tapCount)")
                .font(.largeTitle)
            Spacer()
            Button("Tap here") {
                }
            Spacer()
        }
    }
}
```
struct ContentView: View {
    let peopleInIOS = ["René Schäfer", "Oliver Nowak", "Jan Borchers"]

    var body: some View {
        LazyVStack(spacing: 10) {
            ForEach(0..<3) { index in
                Text(peopleInIOS[index])
                    .font(.largeTitle)
            }
        }
    }
}

Prof. Dr. Jan Borchers: iOS Application Development
Variables in SwiftUI

```swift
struct ContentView: View {
    let peopleInIOS = [
        "René Schäfer",
        "Oliver Nowak",
        "Jan Borchers"
    ]

    var body: some View {
        LazyVStack(spacing: 10) {
            ForEach(peopleInIOS, id: \.self){ person in
                Text(person)
                    .font(.largeTitle)
            }
        }
    }
}
```
Customization
Custom Views

• Create complex views by splitting into smaller views
• Makes views reusable and simpler to exchange
• Keeps the code more readable and easier to understand
Custom Views

```swift
struct ContentView: View {
    let peopleIniOS = ["René Schäfer", "Oliver Nowak", "Jan Borchers"]

    var body: some View {
        LazyVStack(spacing: 10) {
            ForEach(peopleIniOS, id: \.self){ person in
                CustomRow(name: person)
            }
        }
    }
}

struct CustomRow: View {
    var name: String

    var body: some View {
        HStack {
            Image(systemName: "checkmark")
            Text(name)
                .font(.largeTitle)
            .padding()
        }
    }
}
```
Controls and Property Wrappers
Changing Variables

```swift
struct ContentView: View {
    var tapCount = 0

    var body: some View {
        VStack {
            Spacer()
            Spacer()
            Text("\(tapCount)"
                 .font(.largeTitle)
                 Spacer()
            Button("Tap here") {
                tapCount += 1
            }
            Spacer()
        }
    }
}
```
Property Wrapper: @State

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

    var body: some View {
        VStack {
            Spacer()
            Spacer()
            Text("\((tapCount)"
                .font(.largeTitle)
            Spacer()
            Button("Tap here") {
                tapCount += 1
            }
            Spacer()
        }
    }
}
```
Property Wrapper: @State

- **@State** lets you modify simple properties stored inside a single view.
- Recommended to use as **private** variable.
- Changing **@State** properties will reinvoke the body property.
- Note: **Button**'s action is specified as trailing closure.
Two-Way Bindings

```swift
struct ContentView: View {
    @State private var name = ""

    var body: some View {
        Form {
            Text("Hello \\
(name)")
            TextField("Enter your name", text: $name)
        }
    }
}
```

- `$` makes `name` a property with two-way binding: The TextField View displays its current value, but can also change it.
Property Wrapper: `@Binding`

CustomMediaControl { }

Silence

Play

Enjoy the silence

Photo by Shalagh Murphy from Pexels
Property Wrapper: `@Binding`

```swift
struct ContentView: View {
    @State private var isPlaying = false

    var body: some View {
        VStack {
            Spacer()
            if isPlaying {
                VStack {
                    Image("playImage")
                        .resizable()
                        .scaledToFit()
                    Text("Photo by Shelagh Murphy from Pexels")
                        .font(.caption)
                }else {
                    Text("Silence")
                        .font(.largeTitle)
                        .fontWeight(.thin)
                }
                Spacer()
                CustomMediaControl(isPlaying: $isPlaying)
            }
        }
        .frame(alignment: .bottom)
    }
}
```
# Property Wrapper: @Binding

```swift
struct CustomMediaControl: View {
    @State var isPlaying: Bool

    var body: some View {
        VStack{
            Button(isPlaying ? "⏹" : "▶") {
                isPlaying.toggle()
            }
            .font(.largeTitle)
            Text(isPlaying ? "Enjoy the silence" : "Play")
        }
    }
}
```
Property Wrapper: @Binding

```swift
struct CustomMediaControl: View {
    @Binding var isPlaying: Bool

    var body: some View {
        VStack{
            Button(isPlaying ? "⏸" : "▶") {
                isPlaying.toggle()
            }
            .font(.largeTitle)
            Text(isPlaying ? "Enjoy the silence" : "Play")
        }
    }
}
```
Other Property Wrappers

• @StateObject
  A property wrapper type that instantiates an observable object.

• @ObservedObject
  A property wrapper type that subscribes to an observable object and invalidates a view whenever the observable object changes.

• @EnvironmentObject
  A property wrapper type for an observable object supplied by a parent or ancestor view.

• @Environment
  A property wrapper that reads a value from a view’s environment.
Navigation
Drill-Down Navigation Example

1. The Offer
   - Let me tell you a secret.
   - "Sure!"
   - "Hmm?"

2. The Secret
   - SwiftUI is awesome! 🎉

3. A Missed Opportunity
   - Okay, so don’t.
   - “Could you repeat what you said?”
Using NavigationStack

```swift
struct ContentView: View {
    var body: some View {
        NavigationStack {
            VStack(spacing: 20) {
                Text("🤫 Let me tell you a secret.")
                    .padding(.bottom, 30.0)
                    .font(.largeTitle).multilineTextAlignment(.center)
                NavigationLink(""Sure!"", destination: PositiveAnswerView())
                NavigationLink(destination: NegativeAnswerView()) {
                    HStack {
                        Text("😡")
                        .font(.title)
                        Text(""Hmm?""")
                    }
                }
            }
            .navigationTitle(Text("The Offer"))
            .toolbar{
                ToolbarItem(placement: .navigationBarTrailing){
                    Button("Credits"){}
                }
            }
        }
    }
}
```

Added to the VStack!
NavigationStack: Grouping Toolbar Items

```swift
struct ContentView: View {
    var body: some View {
        NavigationStack {
            VStack(spacing: 20) {
                Text("🤫\nLet me tell you a secret.")
                    .padding(.bottom, 30.0)
                    .font(.largeTitle).multilineTextAlignment(.center)
            NavigationLink("\nSure!\n", destination: PositiveAnswerView())
            NavigationLink(destination: NegativeAnswerView()) {
                HStack {
                    Text("🥱")
                        .font(.title)
                    Text("\nHmm?\n")
                }
            }
            .navigationBarTitle(Text("The Offer"))
            .toolbar{
                ToolbarItemGroup(placement: .navigationBarTrailing){
                    Button("Credits"){}
                    Button("Help"){}
                }
            }
        }
    }
}
```

Prof. Dr. Jan Borchers: iOS Application Development
struct ContentView: View {
    var body: some View {
        NavigationStack {
            VStack(spacing: 20) {
                Text("🤫\nLet me tell you a secret."")
                    .padding(.bottom, 30.0)
                    .font(.largeTitle).multilineTextAlignment(.center)
                NavigationLink(""Sure!"", destination: PositiveAnswerView())
                NavigationLink(destination: NegativeAnswerView()) {
                    HStack {
                        Text("🥱")
                            .font(.title)
                        Text(""Hmm?""")
                    }
                }
            }
            .navigationTitle(Text("The Offer"))
            .toolbar{
                ToolbarItemGroup(placement: .navigationBarTrailing){
                    Button("Credits"){}
                    Button("Help"){}
                }
            }
        }
    }
}
struct PositiveAnswerView: View {
    var body: some View {
        VStack() {
            Text("SwiftUI is awesome! 🎉")
                .font(.largeTitle)
        }
        .navigationTitle(Text("The Secret"))
    }
}
struct NegativeAnswerView: View {
    @Environment(
        .presentationMode
    ) var presentationMode

    var body: some View {
        VStack(spacing: 40) {
            Text("Okay, so don't.")
                .font(.largeTitle)
            Button(""Could you repeat what you said?"")
                .action {
                    presentationMode.wrappedValue.dismiss()
                }
        }
        .navigationBarTitle(Text("A Missed Opportunity"))
        .navigationBarBackButtonHidden(true)
    }
}
struct ContentView: View {

    @State var showSecret = false

    var body: some View {
        VStack(spacing: 20) {
            Text("扫一\nLet me tell you a secret.")
                .padding(.bottom, 30.0)
                .font(.largeTitle).multilineTextAlignment(.center)
            Button(""Sure!""){
                showSecret.toggle()
            }
            .sheet(isPresented: $showSecret){
                PositiveAnswerView(showSecret: $showSecret)
            }
        }
    }
}

Also works this way with NavigationLinks
struct PositiveAnswerView: View {
    @Binding var showSecret: Bool

    var body: some View {
        VStack(spacing: 100) {
            Text("SwiftUI is awesome! 🎉")
                .font(.largeTitle)
            Button(""Could you tell my friend this secret, too?"""){
                showSecret = false
            }
        }
    }
}
NavigationLinks: NavigationDestination

```swift
struct ContentView: View {

    var body: some View {
        NavigationStack {
            List(people, id: \.self){ person in
                NavigationLink(person, value: person)
            }
            .navigationDestination(for: String.self){ person in
                ProfilePage(person: person)
            }
        }
    }
}
```
SwiftUI: Further Reading

- Apple's SwiftUI Tutorials, and WWDC presentations
- Paul Hegarty, Stanford University: CS193p — Developing Applications for iOS
- Paul Hudson: 100 Days of SwiftUI (free), Swift Design Patterns ($25–50)
- Mark Moeykens: SwiftUI View Mastery
- Next release of Apple's free Develop in Swift books (currently still UIKit only)
SwiftUI: the Big Messages

1. Object-Oriented Programming is Dead, Long Live **Declarative Programming**!

2. **MVVM** is the corresponding modern improvement over MVC

3. Modern universal languages can describe UIs like **domain-specific languages**

4. You can design a UI **graphically and in code simultaneously**

5. The best app languages must **evolve together** with a UI library and IDE

6. Declarative Programming simplifies development across **mobile and desktop**

7. SwiftUI is a current **case study of a paradigm shift** across a major OS family