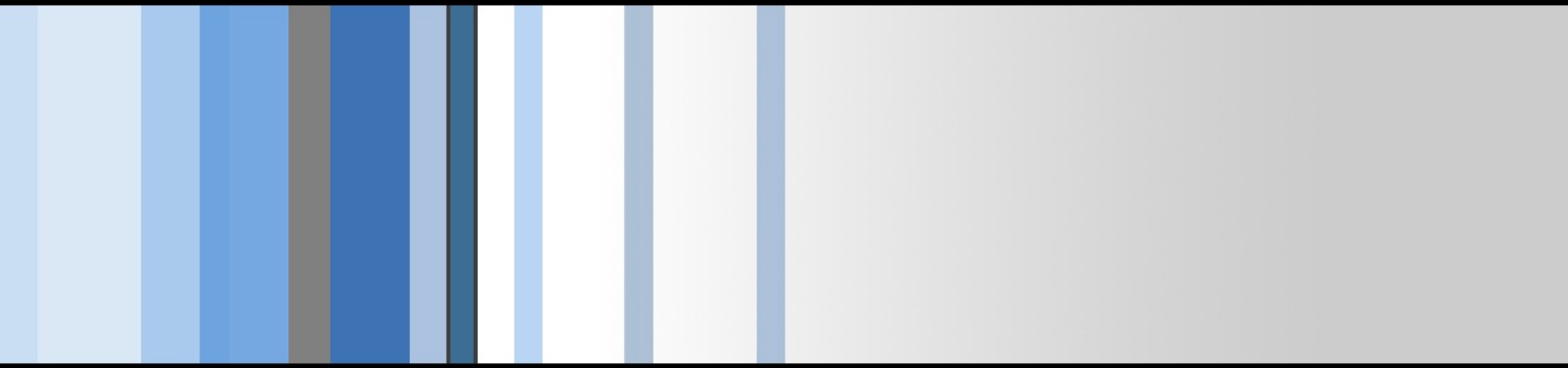
Class 17

CSC 496: iOS App Development
Views in SwiftUI

Si Chen (schen@wcupa.edu)



New SwiftUI View

- To create a new view in SwiftUI, you start by importing the SwiftUI package and then **defining a new struct that conforms to the View protocol**.
- Inside the struct, you'll implement the body computed property, which describes the view's content and layout.
- Here's a basic example to demonstrate creating a new view in SwiftUI:

```
import SwiftUI

struct MyNewView: View {
    var body: some View {
        Text("Hello, world!")
            .padding()
    }
}
```

New SwiftUI View

- **Define a New Struct**: Create a new Swift struct and make it conform to the View protocol.

```
import SwiftUI

struct MyNewView: View {
```

- **Implement the Body**: Within the struct, you'll need to implement a body computed property. This is where you define what your view looks like.

```
var body: some View {
    Text("Hello, world!")
        .padding()
}
```

- **To use MyNewView inside another view, you would do something like this:**

```
struct ContentView: View {
    var body: some View {
        MyNewView()
    }
}
```

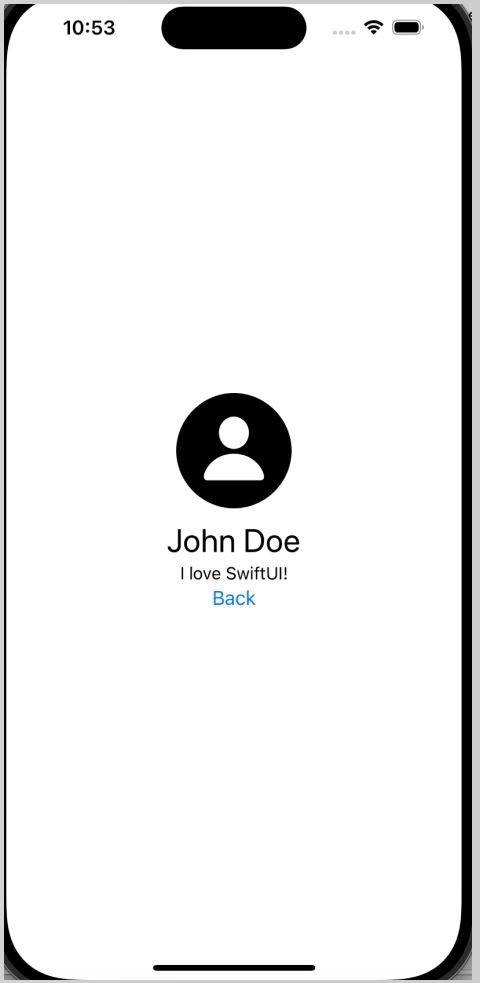
Structuring Views

- In SwiftUI, it's a good practice to structure your views by breaking them down into smaller, reusable components.
- For example, suppose you have a view that displays user information. This view could consist of smaller views: one for the user's avatar, one for the username, and one for the user's bio.

```
import SwiftUI

struct UserProfileView: View {
    var body: some View {
        VStack {
            UserAvatarView()
            UserNameView()
            UserBioView()
        }
    }
}

struct UserAvatarView: View {
    var body: some View {
        Image(systemName: "person.circle.fill")
            .resizable()
            .frame(width: 100, height: 100)
    }
}

struct UserNameView: View {
    var body: some View {
        Text("John Doe")
            .font(.title)
    }
}

struct UserBioView: View {
    var body: some View {
        Text("I love SwiftUI!")
            .font(.subheadline)
    }
}
```

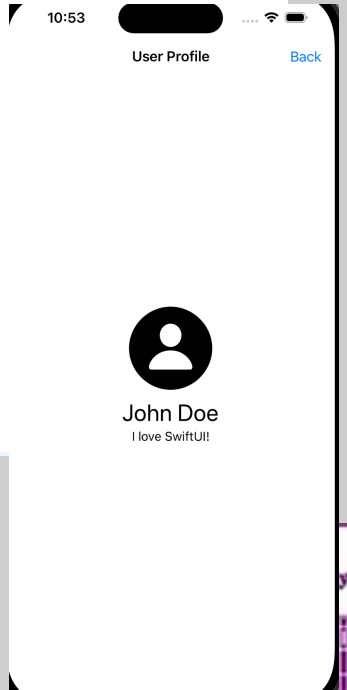
views

```swift
import SwiftUI

struct ContentView: View {
    @State private var showProfile = false

    var body: some View {
        if showProfile {
            UserProfileView()
            Button("Back") {
                showProfile = false
            }
        } else {
            Button("Profile") {
                showProfile = true
            }
        }
    }
}
```



NavigationView

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

    var body: some View {
        NavigationView {
            if showProfile {
                UserProfileView()
                    .navigationBarTitle("User Profile", displayMode: .inline)
                    .navigationBarItems(trailing: Button("Back") {
                        showProfile = false
                    })
            } else {
                Button("Profile") {
                    showProfile = true
                }
            }
        }
    }
}
```



Lab3: Build an app with API and third-party library （Group Project)

- **Assignment Requirements:**

1. **API Selection**: Choose an API for integration from https://github.com/public-apis/public-apis. Alternatively, you may opt for any other API that does not necessitate authentication.
2. **Data Fetching**: Develop an application capable of fetching data from the chosen API. Your application should retrieve at least one property or data field from the API.
3. **User Interface (UI)**: Construct a user interface to display the fetched data. Implement features that allow the user to initiate new data requests. For example, if you select the Weather API, the interface should enable the user to specify a state (like PA or NY) for which to fetch data.
4. **Dynamic Imagery**: Incorporate at least one changeable image in the application's UI. The image should update based on the data retrieved from the API. For instance, the image could turn red if the fetched daily temperatures exceed a certain threshold.
5. **Library Usage**: Utilize **at least one third-party library** in the development of your application.

- **Bonus Criteria:**

- **Asynchronous Programming**: Earn a 2 points bonus if you implement asynchronous programming using async/await for data fetching and handling.

e.g., changeable Image of Weather API

Q & A