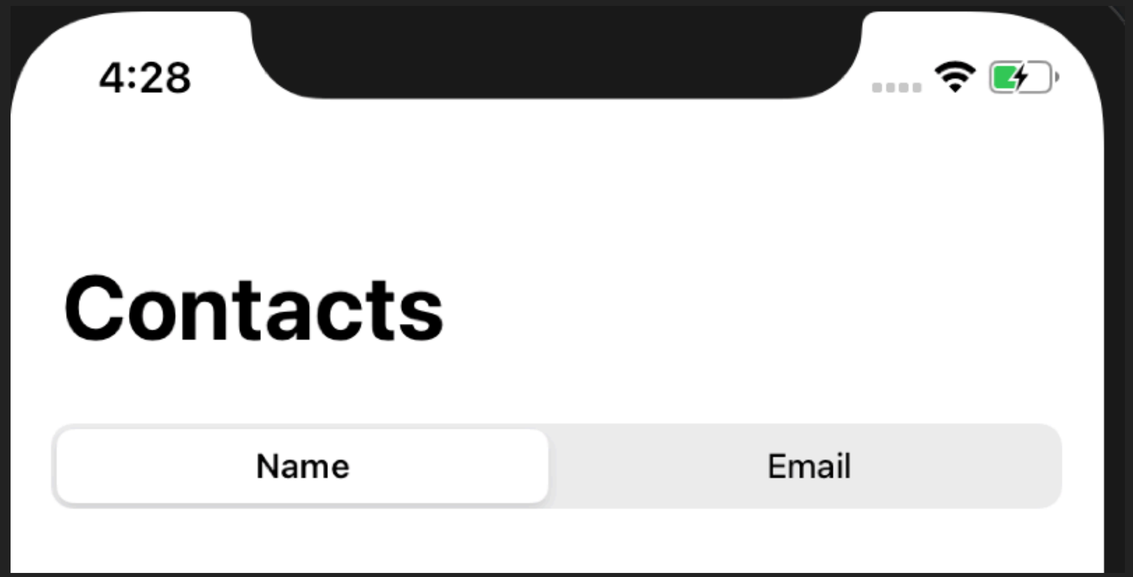
SWIFT UI 100 DAYS CHALLENGE

WEEK 4 | WORKING WITH SEGMENT CONTROL | STATE VARIABLES | NAVIGATION IN LISTS

CREATING A SEGMENT CONTROL

- Segment Control in SwiftUI is a special flavour of Picker itself.
- Picker is a control in SwiftUI which allows us to select a value from, a list of possible options. In order to properly use a picker, we need to back it with an array of possible options to choose from and a State variable storing the index of selected option in the array.
- More information on how to customise a picker can be found on https://developer.apple.com/documentation/swiftui/picker
- The code snippet is provided in the next slide.

```
Picker("Sort Contacts by", selection: $sortingIndex) {
    Text("Name").tag(0)
    Text("Email").tag(1)
}.pickerStyle(SegmentedPickerStyle())
.onReceive([self.sortingIndex].publisher.first(), perform: { (tag) in
    // Here you would want to write your custom logic to perform some operations on segment change
    switch tag {
    case 0:
        contacts.sort { $0.email < $1.email }
    case 1:
        contacts.sort { $0.name > $1.name }
    default:
        break
    }
})
```

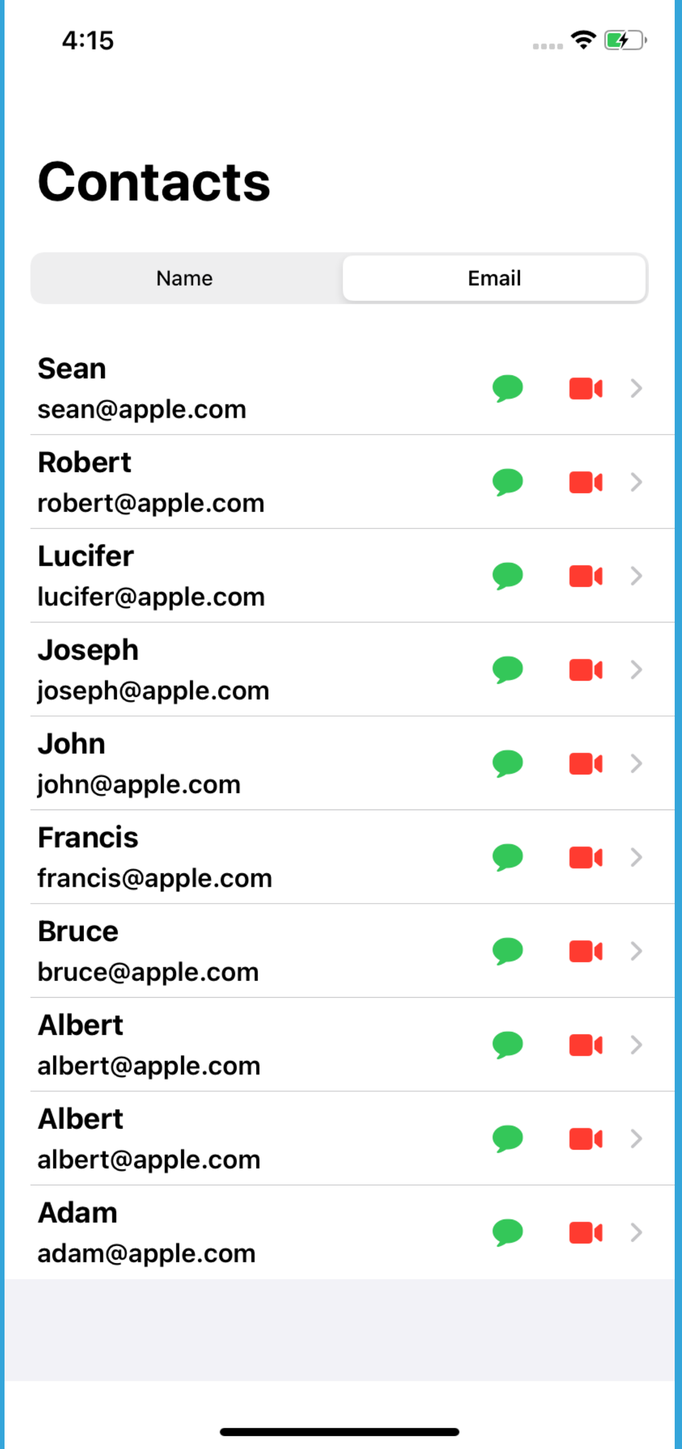


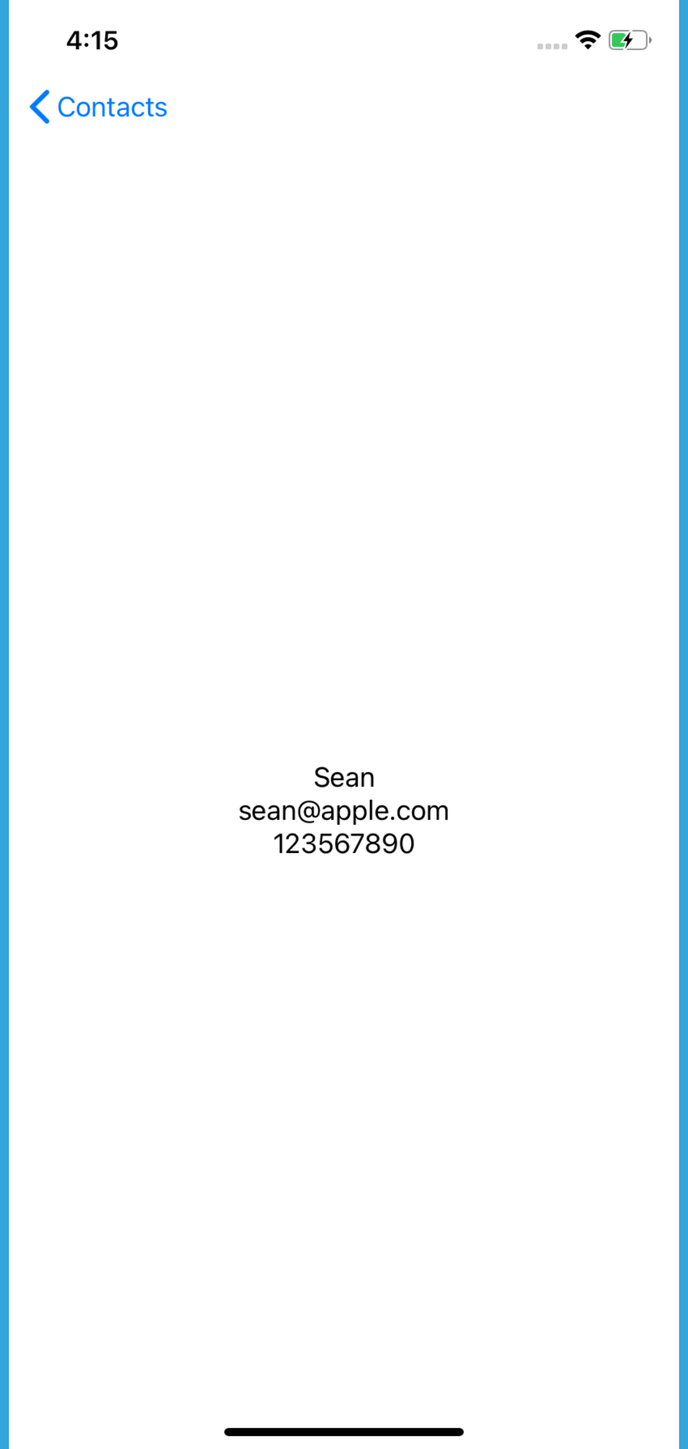
CODE SNIPPET TO CREATE A SEGMENT CONTROL

THIS CODE SNIPPET WILL CREATE A SEGMENT CONTROL AS SHOWN IN THE DIAGRAM

EXAMPLE – A LIST WITH A SEGMENT CONTROL WHICH PERFORMS A SPECIFIC OPERATION & NAVIGATION FROM THE LIST TO A DETAILED VIEW

- For our exercise, this week we will create a list view that will have a segment control, which will simply sort the list based on Name for Segment 0 and Email for Segment 1.
- Let’s look at the end product which has been demonstrated for this example.
- The end product contains a segment control, a list, two buttons (text & call) which shows an alert when tapped.
- Tapping on the list row, it will navigate to the next screen & clicking on the segment control options will sort the list.





FINAL PRODUCT

SWIFTUI WEEK 4

CODE SNIPPET THAT ENABLES NAVIGATION IN LIST

```
List {
    ForEach(contacts, id:\.self) { contact in
        NavigationLink(destination: ContactDetails(contactInfo: contact)) {
            ContactCell(contactInfo: contact)
        }
    }
}.listStyle(GroupedListStyle())
 .onAppear {
```

* Navigation Link takes in a parameter destination, which is a View itself, and when tapped on the cells gets called.

* Pass in reference to the details view which you have created and will be used as the details view.

* The data model that has been used for this demonstration is also attached for reference.

```
struct ContactInfo: Identifiable, Hashable {
    let id = UUID()
    let name: String
    let email: String
    let number: String
}
```

CODE SNIPPET THAT DEMONSTRATES BUTTON ACTION

In the last week challenge, we added a button, this week we will see how to handle the action. For simplicity, a simple alert is being displayed once the call or text button has been tapped. Code snippet is shared as shown in the image.

```
struct ContactCell: View {
    @State private var showingAlert = false

    var contactInfo: ContactInfo
    var body: some View {
        HStack {
            VStack(alignment: .leading, spacing: 5) {
                Text(contactInfo.name)
                    .fontWeight(.bold)
                    .font(.system(size: 18))
                Text(contactInfo.email)
                    .fontWeight(.semibold)
                    .font(.system(size: 16))
            }
            Spacer()
            Button(action: sendMessage) {
                Image(systemName: "message.fill")
                    .frame(width: 40, height: 40, alignment: .center)
                    .font(.body)
                    .foregroundColor(.green)
            }.alert(isPresented: $showingAlert) {
                Alert(title: Text("Still in Development"), message: Text("Once implementation is complete you will be able to send a
                    text message or make a video call"), dismissButton: .default(Text("Got it!")))
            }
            Button(action: makeVideoCall) {
                Image(systemName: "video.fill")
                    .frame(width: 40, height: 40, alignment: .center)
                    .font(.body)
                    .foregroundColor(.red)
            }.alert(isPresented: $showingAlert) {
                Alert(title: Text("Still in Development"), message: Text("Once implementation is complete you will be able to send a
                    text message or make a video call"), dismissButton: .default(Text("Got it!")))
            }
        }
    }

    func makeVideoCall() {
        self.showingAlert = true
    }

    func sendMessage() {
        self.showingAlert = true
    }
}
```

KEY POINTS ABOUT THIS EXERCISE

- Creation of LIST has not been explained here, as it was explained in Week 3.
- Some minute details might have been skipped (code not provided) in the snippets as they were part of earlier weeks.
- This is a reference example, we can customise and make it more interesting.
- Do the same exercise with ScrollView and get extra 5 points.


Thank you for participating in the challenge. That’s it for week 4