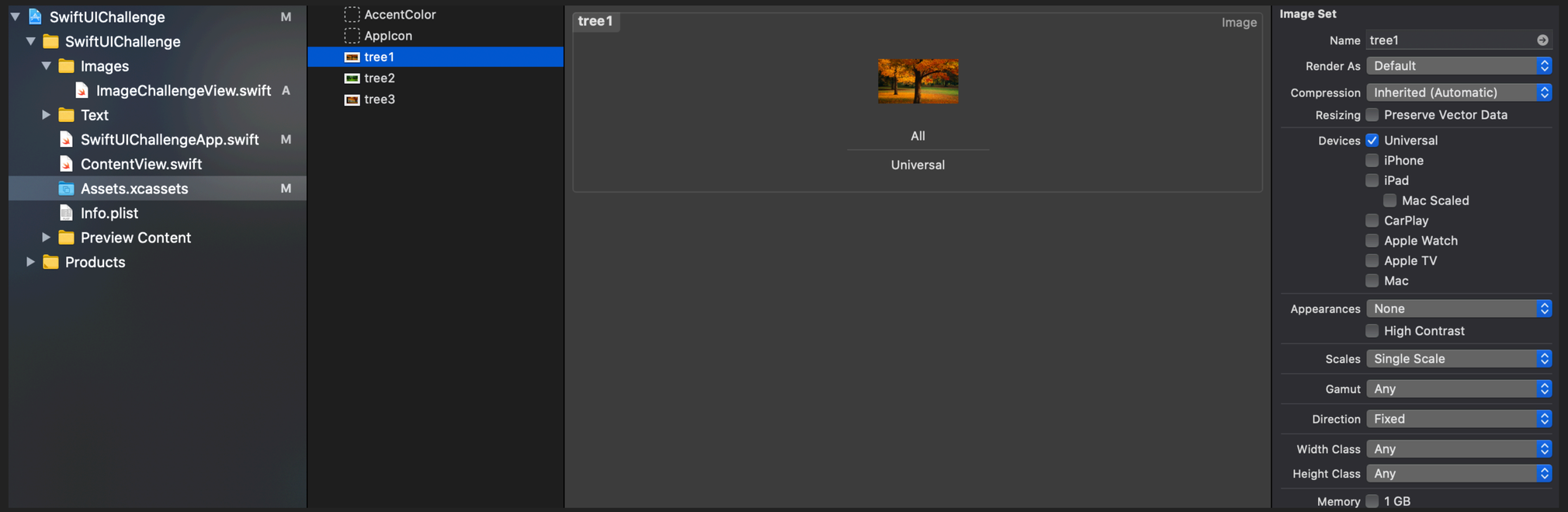
SWIFT UI 100 DAYS CHALLENGE

WEEK 2 | WORKING WITH IMAGES | CUSTOMISATION OF PROPERTIES | WORKING WITH SYSTEM IMAGES | COMBINING TEXT & IMAGES

DISPLAY IMPORTED IMAGE AT ITS FULL SIZE

- Include an image or two in the Assets as shown in the image



DISPLAY IMPORTED IMAGE AT ITS FULL SIZE

- In your SwiftUIView file. “ImageChallengeView”in my case. Include the following code.

```
struct ImageChallengeView: View {
    var body: some View {
        Image("tree1")
    }
}
```

- It can be noticed that image extends beyond the screen size and only a portion of it is visible:
- An enlarged image can be observed as shown in the next slide.

WITH A SIMPLE BLOCK OF CODE WE WERE ABLE TO RENDER AN IMAGE

A FULL SCREEN IMAGE

LETS RENDER THE FULL IMAGE PROPERLY

- With the help of the last code snippet, we were able to render the image, but we observed only a portion of the image was rendered, while the rest of the image was way beyond the visible bounds.
- Let’s fix it by adding an attribute that will resize the image itself.
- .resizable() modifier will make it fit in the entire available screen area.
- Let’s observe the changes in the next slide.

OBSERVATIONS

- It looks much better in a sense that it fits within the screen edges however it’s stretched out now.
- More help at : https://developer.apple.com/documentation/swiftui/image/resizable(capinsets:resizingmode:)

LETS RESIZE THE IMAGE PROPERLY

- Let’s maintain the aspect ratio of the image so that the image renders properly.
- We can use .scaledToFit() and .aspectRatio(contentMode: .fit), both gives the same result.

```
struct ImageChallengeView: View {
    var body: some View {
        Image("tree1")
            .resizable()
            .scaledToFit()
    }
}
```

```
struct ImageChallengeView: View {
    var body: some View {
        Image("tree1")
            .resizable()
            .aspectRatio(contentMode: .fit)
    }
}
```

- Lets observe the changes in the next slide

IN LANDSCAPE ORIENTATION

IN POTRAIT ORIENTATION

OBSERVATIONS

THE IMAGE LOOKS MUCH BETTER WITH THE CONTENT MODE SET

ADDING A CUSTOM FRAME TO THE IMAGE

- The frame can be treated as a rectangle which describes the view's location in the superview's coordinate system.
- We can provide a custom frame to an Image as well

```
struct ImageChallengeView: View {
    var body: some View {
        Image("tree1")
           .resizable()
           .scaledToFill()
           .frame(width: 300, height: 300, alignment: .center)

    }
}
```

- Where width, height species the dimension of the frame, and alignment will identify where the image will be placed in the rectangle.

OBSERVATIONS

IMAGE WITH A CUSTOM FRAME

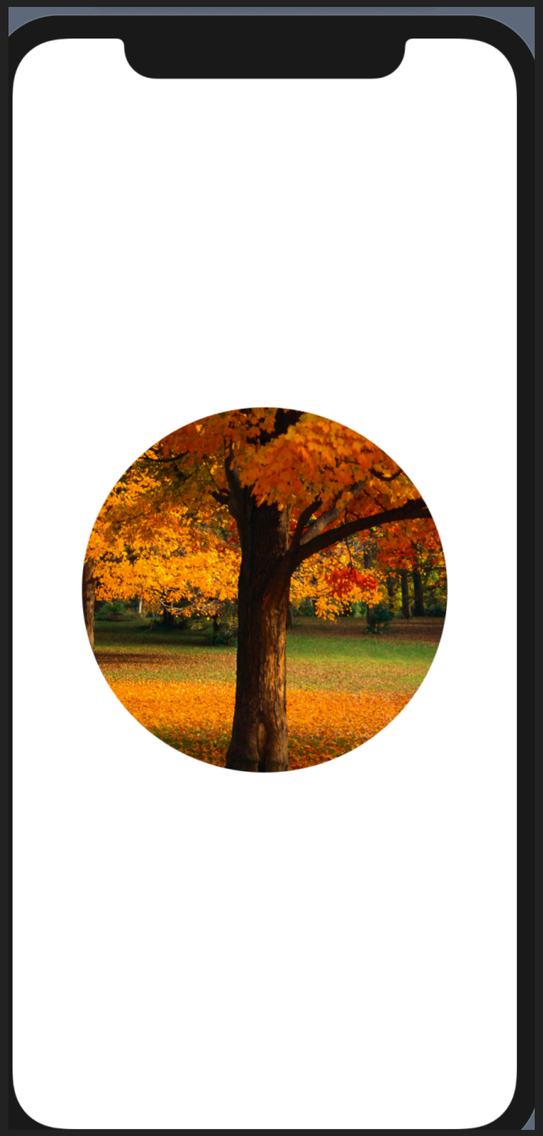
CLIP THE IMAGE TO A CIRCLE AND DISPLAY ONLY A PORTION OF THE IMAGE COVERED BY THE CIRCLE

- Let’s look at the source code

```
struct ImageChallengeView: View {
    var body: some View {
        Image("tree1")
           .resizable()
           .scaledToFill()
           .frame(width: 300, height: 300, alignment: .center)
           .clipShape(Circle())

    }
}
```

- By applying the .resizable() modifier we’re allowing to set the aspect ration of the image.
- Use the .aspectRatio() modifier to fill the entire content of the parent container.
- Restrict the size of the image to a custom frame.
- Apply the .clipShape() modifier using a Circle().
- Let’s see the observations in the next slide.



OBSERVATIONS

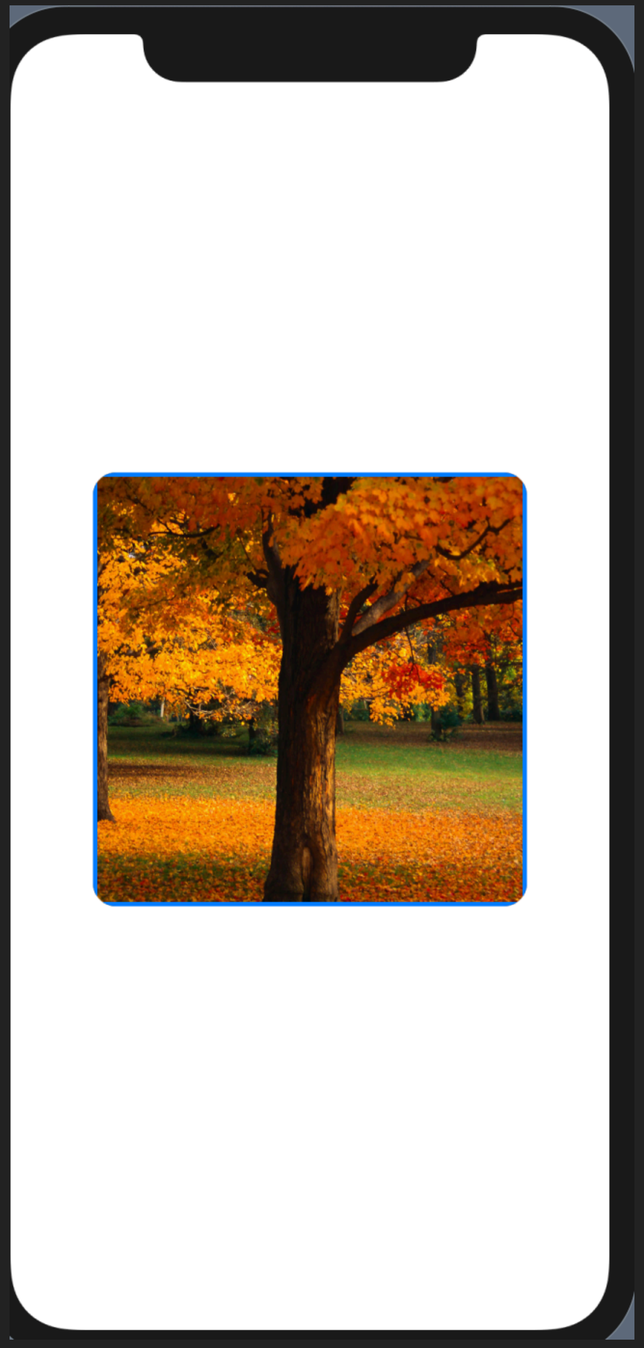
A NICE ROUNDED IMAGE

ADDING SOME BORDER COLOUR, CORNER RADIUS TO THE IMAGE

- Lets look at the source code

```
struct ImageChallengeView: View {
    var body: some View {
        Image("tree1")
           .resizable()
           .scaledToFill()
           .frame(width: 300, height: 300, alignment: .center)
           .border(Color.blue, width: 3.0)
           .cornerRadius(16.0)
           .clipped()
    }
}
```

- The observations can be seen in the next slide

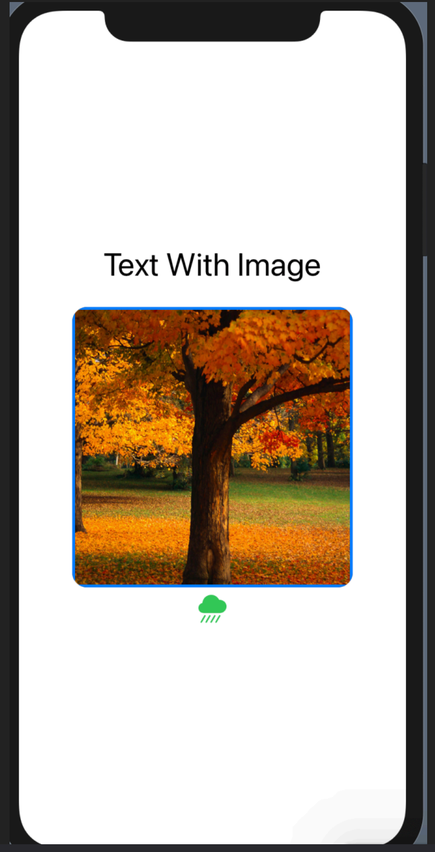


OBSERVATIONS

IMAGE WITH A BORDER COLOUR, CLIPPED & WITH A CORNER RADIUS

WORKING WITH SYSTEM IMAGES & COMBINING IT WITH OUR FAVOURITE TEXT

- Just like Font, Color Image is also a **late-binding token**, which means that the system resolves its actual value only when it’s about to use the image in a given environment. It takes into consideration what environment is currently set to dark or light mode. Then the system resolves its actual value.
- The next source code will contain a System Image, a Custom Image and a Text all aligned in a VSTACK. Let’s only focus on Image & Text here. VSTACK is another point of discussion.
- Lets look at the observations in the next slide.




```swift
struct ImageChallengeView: View {
    var body: some View {
        VStack(content: {
            Text("Text With Image")
                .font(.largeTitle)
            Image("tree1")
               .resizable()
               .scaledToFill()
               .frame(width: 300, height: 300, alignment: .center)
               .border(Color.blue, width: 3.0)
               .cornerRadius(16.0)
               .clipped()
            Image(systemName: "cloud.heavyrain.fill")
                .resizable()
                .font(.largeTitle)
                .foregroundColor(Color.green)
                .frame(width: 30, height: 30, alignment: .center)
        })
    }
}
```

OBSERVATION

- SYSTEM IMAGE THAT CAN BE RESIZED TO ANY DIMENSION & CUSTOMISED LIKE ANY OTHER VIEW


Thank you for participating in the challenge. That’s it for week 2