Catoptric Surface Control Interface

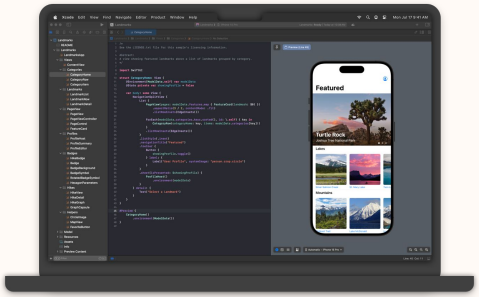
—Master's Project Defense, Logan Farrow

Table of Contents

01

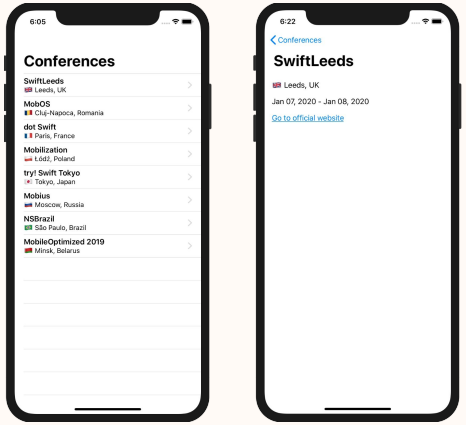
Overview

My Trajectory Throughout the Semester



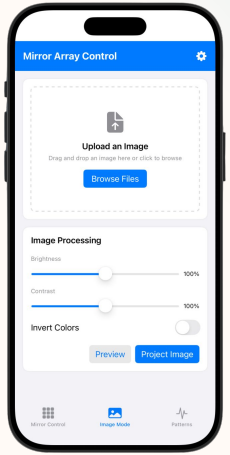
Learning SwiftUI

Watching videos weekly, learning new features as Apple creates new updates.



Building with SwiftUI

Building sample apps, testing different approaches, and researching common wireframes.



Applying SwiftUI

Using the SwiftUI principles to build a “remote control” interface for the mirror array.

What is Swift?

- Swift is a general-purpose, compiled programming language developed by Apple in 2014.
- Replaced Objective-C for Apple development, but still has interoperability
- Technically meant to be general purpose, but rarely used for non “Apple” projects.
- Swift took language ideas "from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list” – Chris Lattner (Swift Creator)

What is SwiftUI?

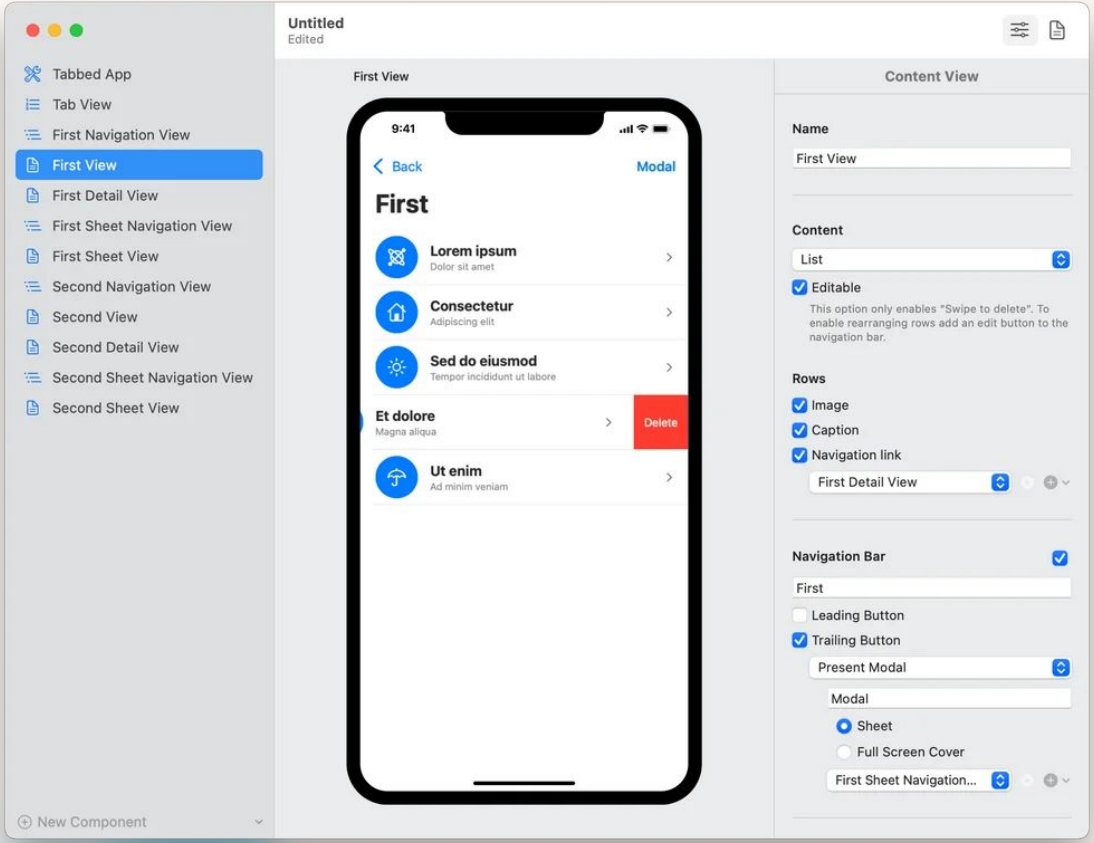
- SwiftUI is a relatively new (2019) user interface framework for Swift.

- Created by Apple for a new approach to UI development.
    - Focuses on multi platform development.
    - First ever “real time” previews of your app as you develop it.

- Similar to popular web frameworks like React, SwiftUI can automatically update the UI as “states” change.

Why SwiftUI?

- Declarative UI syntax
  - UI is “code” instead of being forced to use an interface to build an interface
- Cross Compatibility
  - code sharing across iOS, macOS, watchOS, and tvOS
- Apple’s “Future” Framework
  - UiKit is dubbed to be the main framework for Apple ecosystem development
- Automatic & Speedy Live Previews



02

UIKit v.s. SwiftUI

What is UiKit?

- UIKit is much more longstanding and was the go to for iOS development for many years
- Uses a storyboard to allow developers to place items like buttons then have them appear in code
- Code shares a space with the UIKit storyboard area inside of your development IDE
- Allows you map out every screen in your app and connect buttons via navigation “roads” to get a great overview of your entire structure

App Development Today

- Ever since SwiftUI's release Apple has been strongly pushing the full adoption the new framework.
- Apple still makes updates to UIKit regularly, but not as often as SwiftUI
- Most apps on the AppStore today use SwiftUI, but a major drawback is that it does not support older iOS versions
- SwiftUI has support for including some elements of UIKit for more advanced functionality

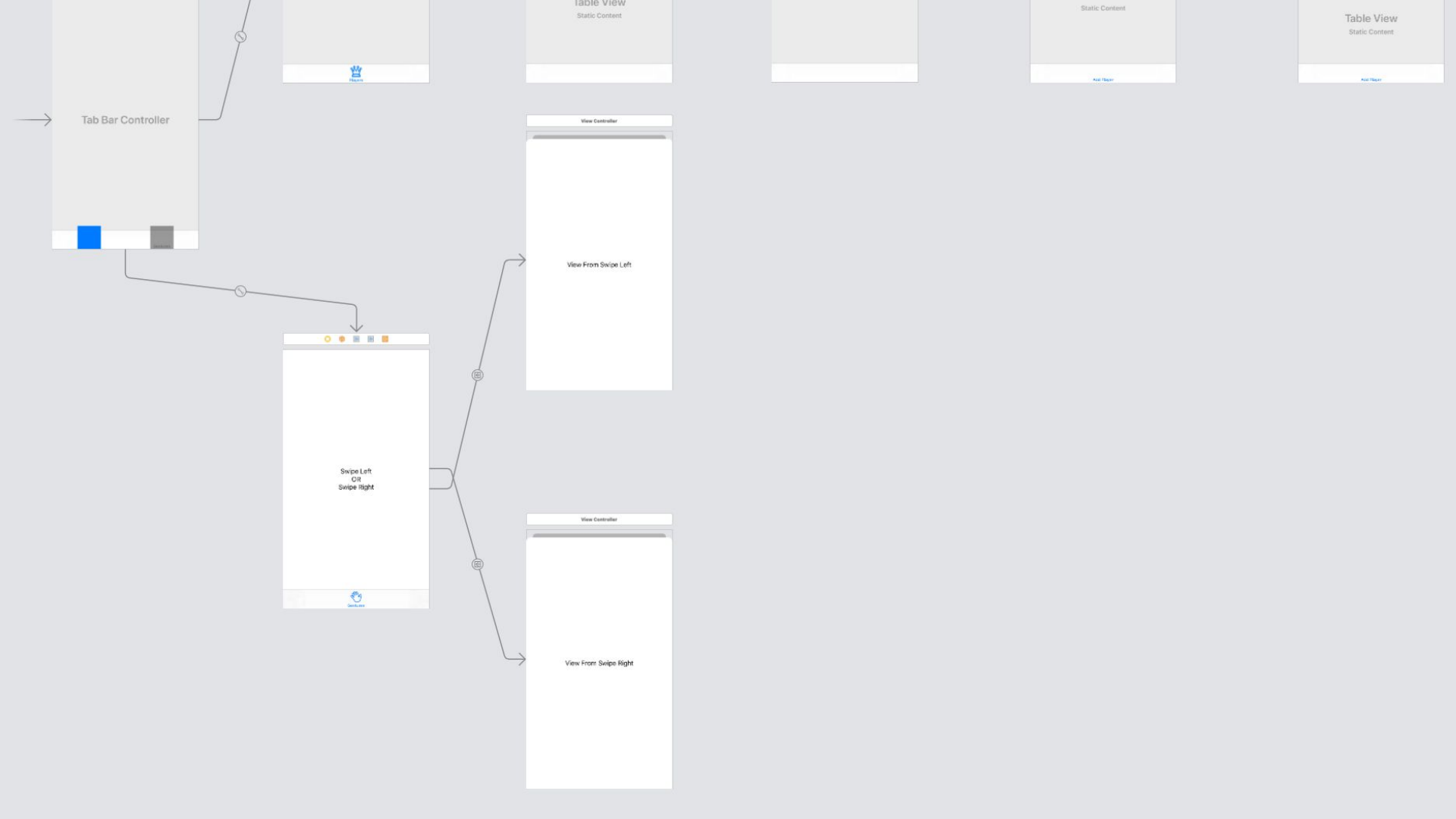
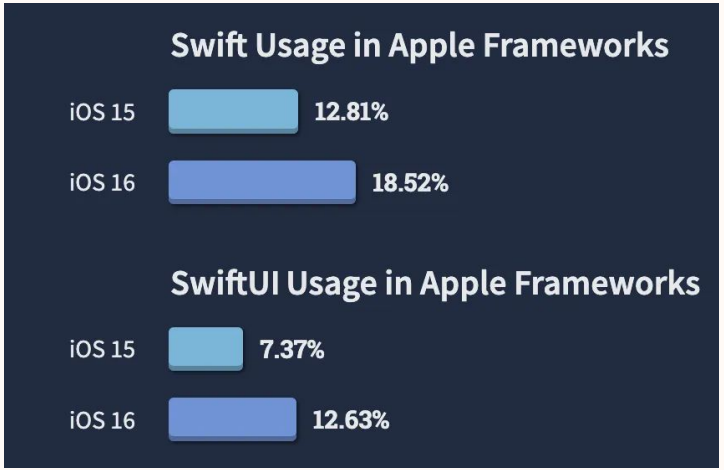


Table View
Static Content
Table View
Static Content
Tab Bar Controller
View Controller
View From Swipe Left
Swipe Left
OR
Swipe Right
View Controller
View From Swipe Right

Other Reasons to Choose SwiftUI

- Even though Apple still makes updates to UIKit, it is clear that it will be deprioritized in the future as functionality is added to SwiftUI

- SwiftUI app and framework usage is growing rapidly and is likely even higher than these numbers with today's iOS 18



03

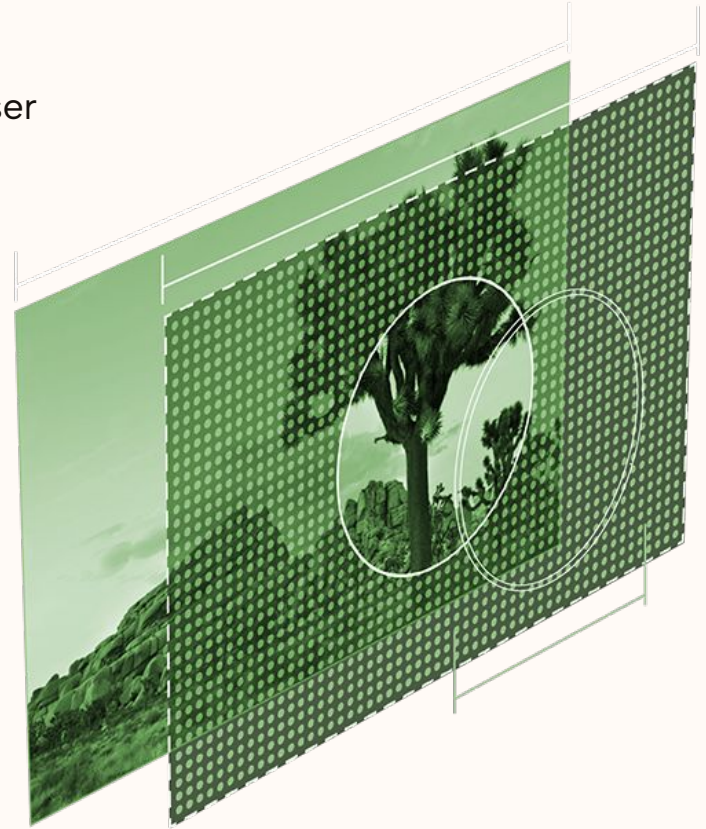
Learning SwiftUI

My Approach

- Learn:
  - Learn and follow guides breaking down components of Swift & SwiftUI
- Build:
  - Create examples and test features of SwiftUI on my own machine
- Understand:
  - Take notes on these topics to use them in my control app for the mirror array.

Views

- Views are the fundamental components used to construct the user interface.



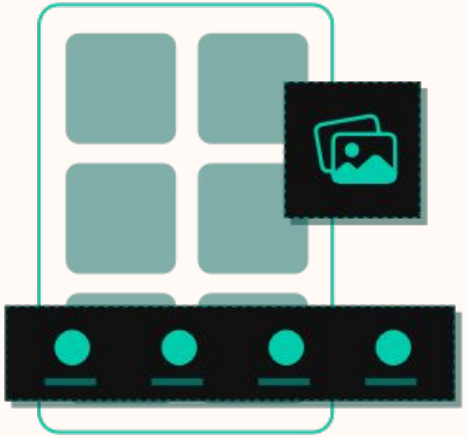
- Each view describes a portion of the UI
    - Basic View Types: Text, Image, etc

- Can be customized with modifiers which work like CSS in typical website development.

- Views can be nested within other views to create a hierarchy

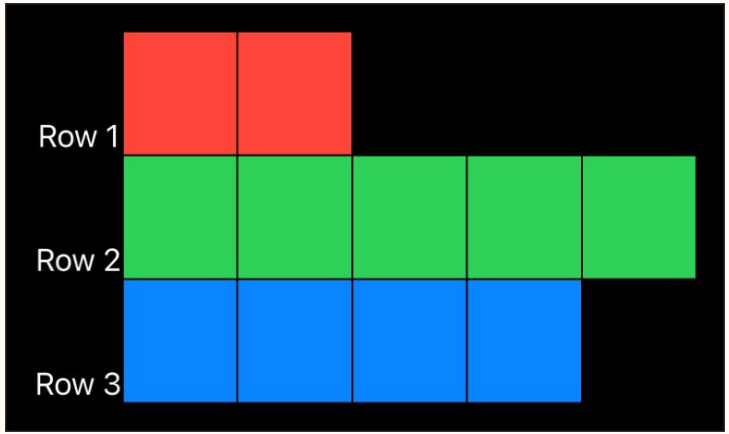
Presentation of Content

- When your app runs, it displays whatever content you place in the body property.



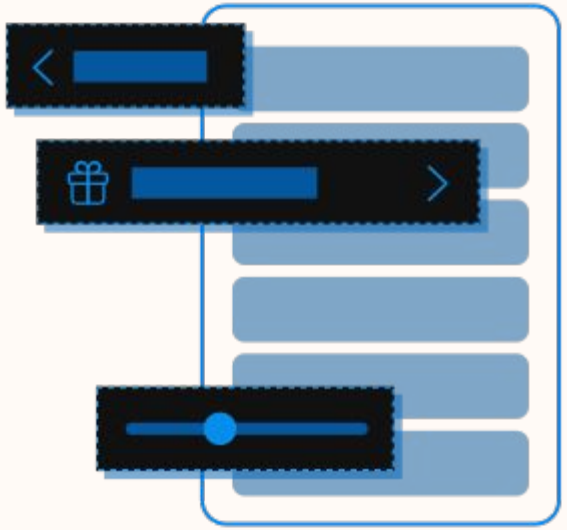
- However, you may want to format this content differently instead of just a static object in a place

- ContentViews are an extension of views in which you can build collections of data, like “Taskbars” at the bottom and grid view content



Navigation

- NavigationStack creates a stack-based navigation flow for users to push and pop views.

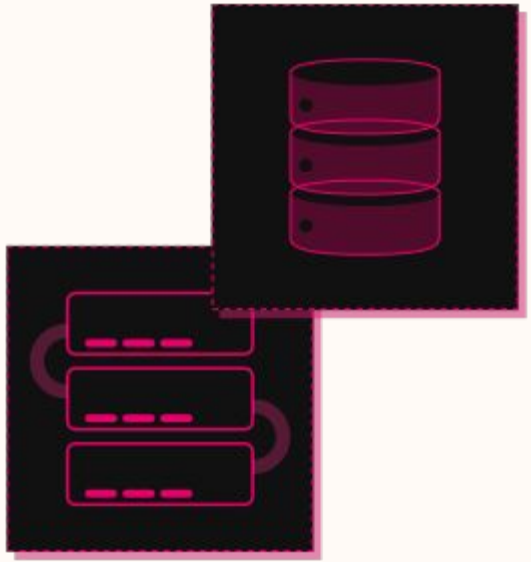


- NavigationLink for user-initiated navigation and navigationDestination(for:) to define destinations based on data types

- NavigationPath used to programmatically manipulate the navigation stack

Retrieving Server Content

- Create a Swift struct that conforms to the Codable protocol to match the structure of the JSON data you expect.
- Utilize URLSession to initiate asynchronous network calls
- Parse JSON data using built-in JSONDecoder
- Update @State properties to update UI

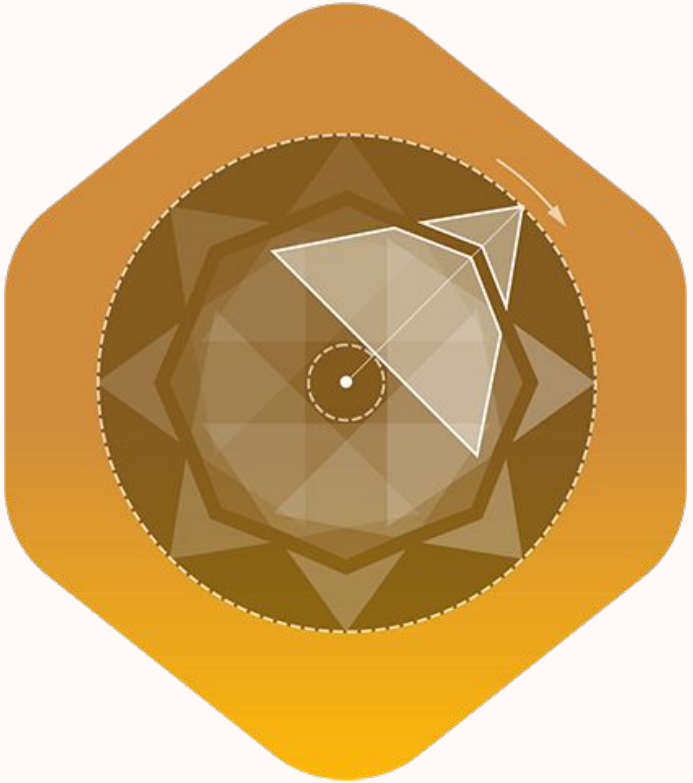


Drawing & Animation

- GeometryReader in conjunction with ZStack allows you to create vector graphics that automatically resize.

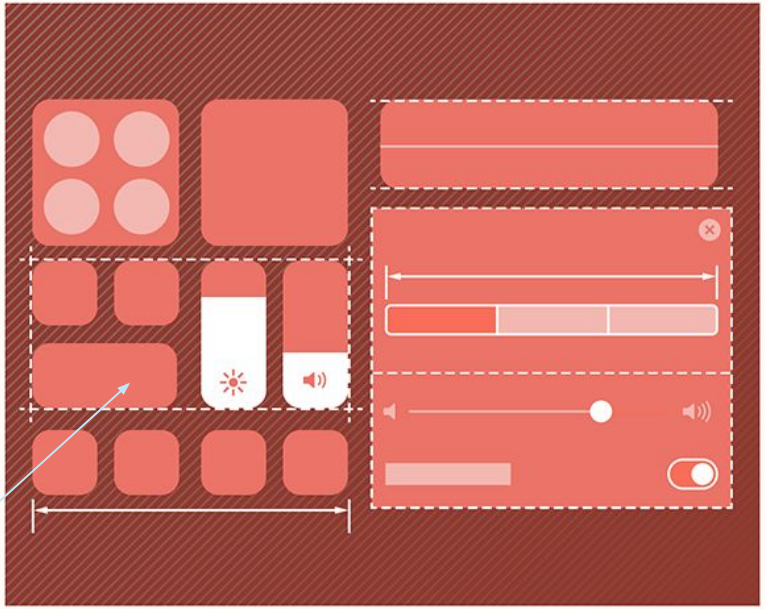
- Using the Zstack also allows you layer multiple shapes and views to build complex graphics.

- SwiftUI also has built in animation properties which allow you to make your app responsive.



App Design & Layout

- Can horizontally scrollable rows, using List, ScrollView, HStack, and VStack to display landmarks grouped by category



- You can create your own custom views by building new structs to conform to the View protocol

- By coming multiple views, you can create complex layouts.

Frameworks & Integrations

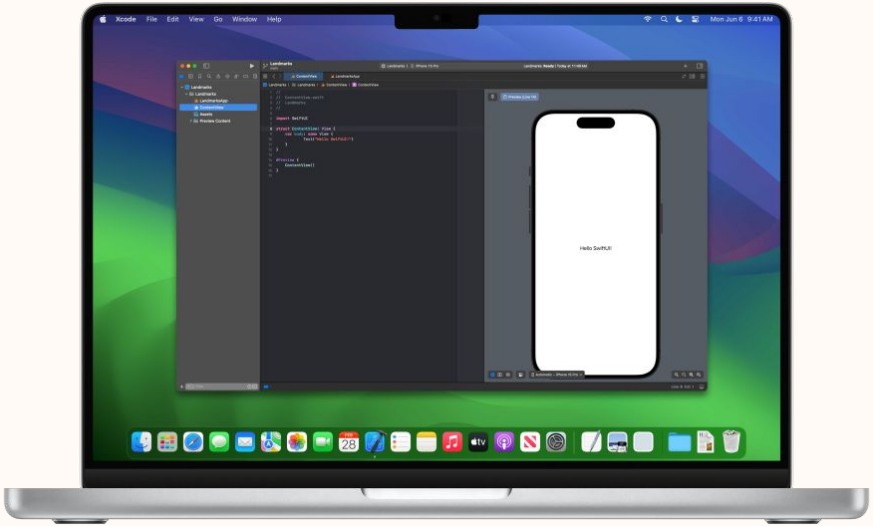
- `UIViewControllerRepresentable` and `UIViewRepresentable` to wrap UIKit components inside SwiftUI

- Sync SwiftUI state with UIKit components using `@Binding`

- Uikit has built in controls like page swiping and tappable indicators which can be brought into SwiftUI

Apple Ecosystem Compatibility

- Share views, models, and resources across platforms by adding a macOS target
- Can specify which views are platform specific to optimize for macOS
- To make use of macOS settings, you can add a Settings scene with @AppStorage to persist user preferences

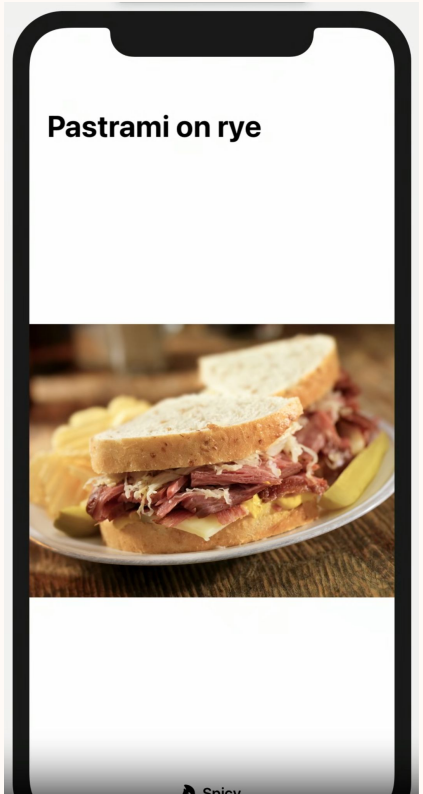


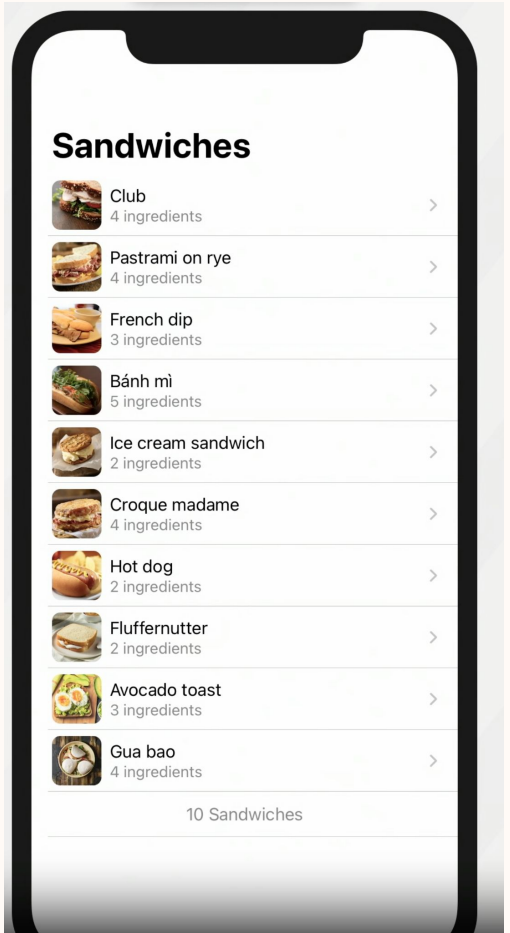
04

Applying SwiftUI

My First App Build: Sandwiches

This was an app I worked on as I followed some of the original SwiftUI announcement videos from the 2020 WWDC developer conference. This gave me experience with using images in SwiftUI and building interactive lists with navigation.





What I Used & What I Learned

- Got used to working with the new views introduced into SwiftUI compared to UI kit.

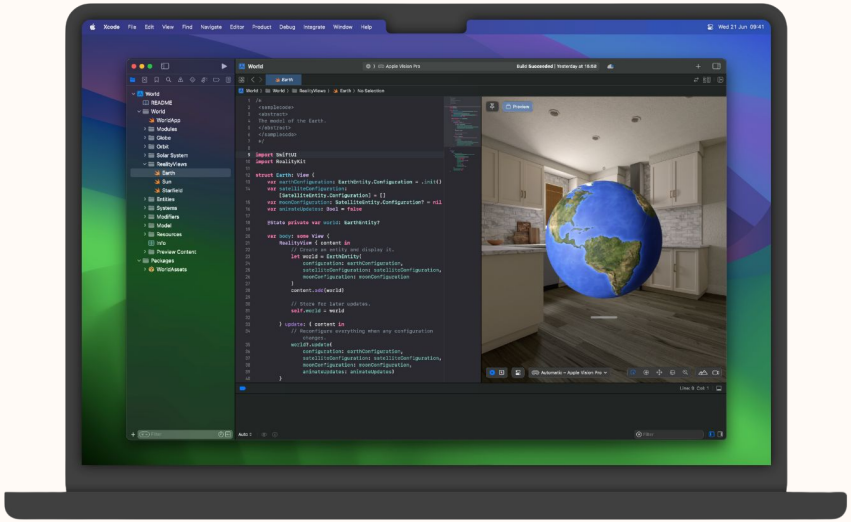
- Practiced the new form of navigation introduced in the SwiftUI framework.

- Learned how to add images and access them within the app, learning how to properly use Xcode with SwiftUI

```
// Views are composed

struct SandwichDetail: View {
    let sandwich: Sandwich

    var body: some View {
        Image(sandwich.imageName)
            .resizable()
            .aspectRatio(contentMode: .fit)
    }
}
```

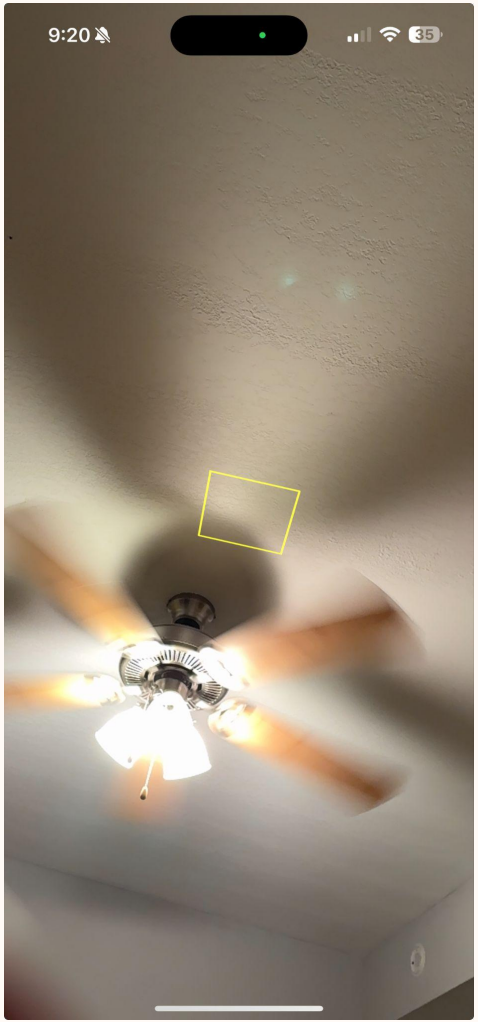
My Second App Build: 3D Visualizer

This was some experimentation I did with some of the new Apple frameworks related to 3D modelling and placing objects in 3D space. (RealityKit & ARkit)



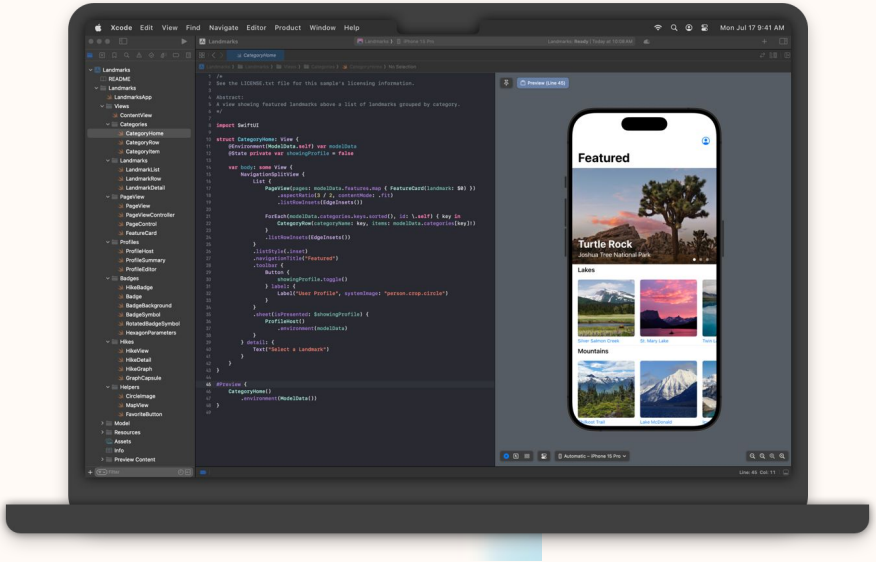
What I Used & What I Learned

- Bridging SwiftUI and UIKit
- AR Session Configuration and Plane Detection
- Implementing FocusEntity for Visual Placement Cues



My Third App Build: National Parks

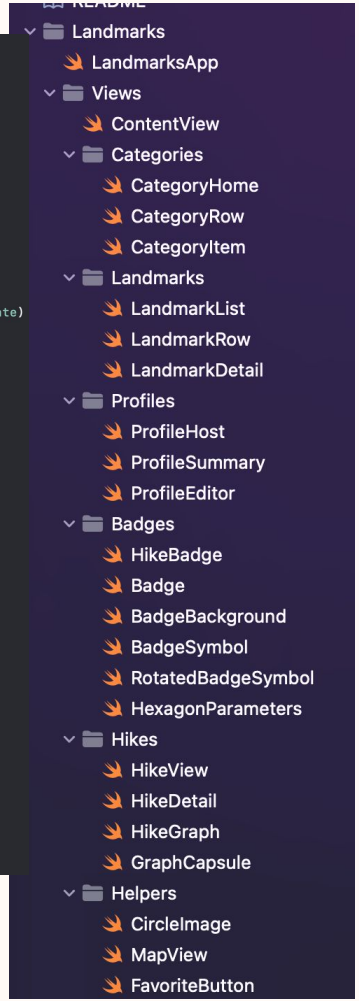
This app introduced me to many aspects of SwiftUI that I covered in the previous slide, including grid layouts, scroll views, image use, cross compatibility, and most features necessary for being well rounded in SwiftUI.



What I Used & What I Learned

- Navigation
  - Using the new navigation links and replacements for the Storyboard from UIkit
- Views
  - Using all types of SwiftUI views and proper file management to combine them.
- Data From Server
  - Creating structs to house data and applying to SwiftUI views.

```
import SwiftUI

struct LandmarkDetail: View {
    @Environment(ModelData.self) var modelData
    var landmark: Landmark

    var landmarkIndex: Int {
        modelData.landmarks.firstIndex(where: { $0.id ==
            landmark.id })!
    }

    var body: some View {
        @Bindable var modelData = modelData

        ScrollView {
            MapView(coordinate: landmark.locationCoordinate)
                .frame(height: 300)

            CircleImage(image: landmark.image)
                .offset(y: -130)
                .padding(.bottom, -130)

            VStack(alignment: .leading) {
                HStack {
                    Text(landmark.name)
                        .font(.title)
                    FavoriteButton(isSet:
                        $modelData.landmarks
                        [landmarkIndex].isFavorite)
                }

                HStack {
                    Text(landmark.park)
                    Spacer()
                    Text(landmark.state)
                }
                .font(.subheadline)
                .foregroundStyle(.secondary)

                Divider()

                Text("About \(landmark.name)")
                    .font(.title2)
                Text(landmark.description)
            }
            .padding()
        }
        .navigationTitle(landmark.name)
        .navigationBarTitleDisplayMode(.inline)
```



Timeline - Catoptric Surface Control Interface

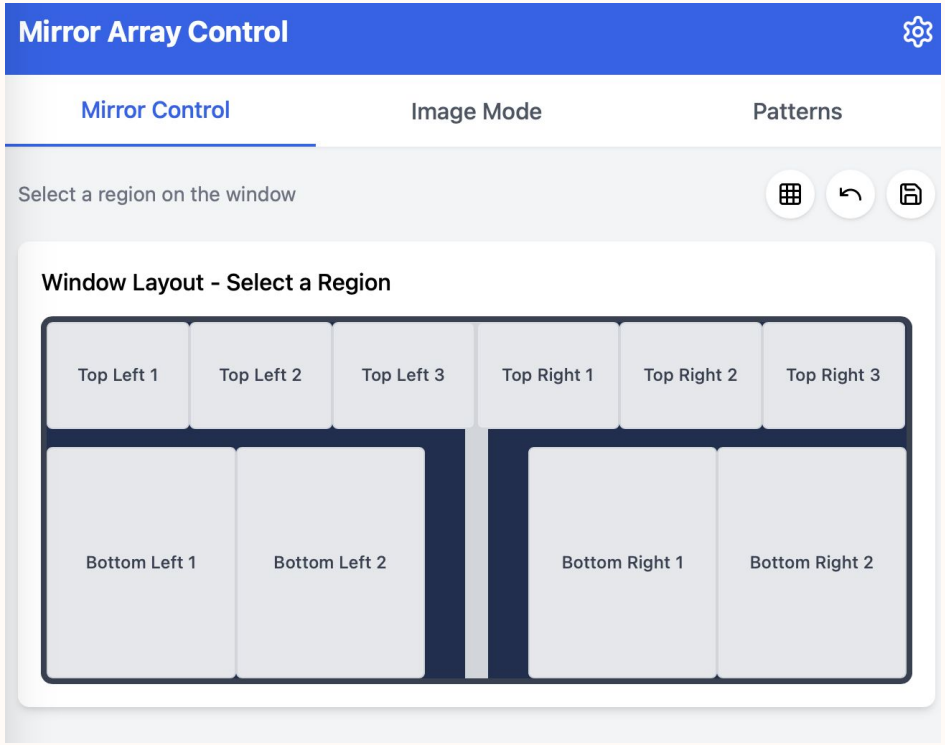
- Reviewed Chandler's Grasshopper Work
- Laid Out Planned Features
- Web App "Wireframe" using React
- Collect Wireframe Feedback
- Convert Wireframe to SwiftUI Application
- Debug & Refine
- Deploy Application to Real iPhone

05

Web App

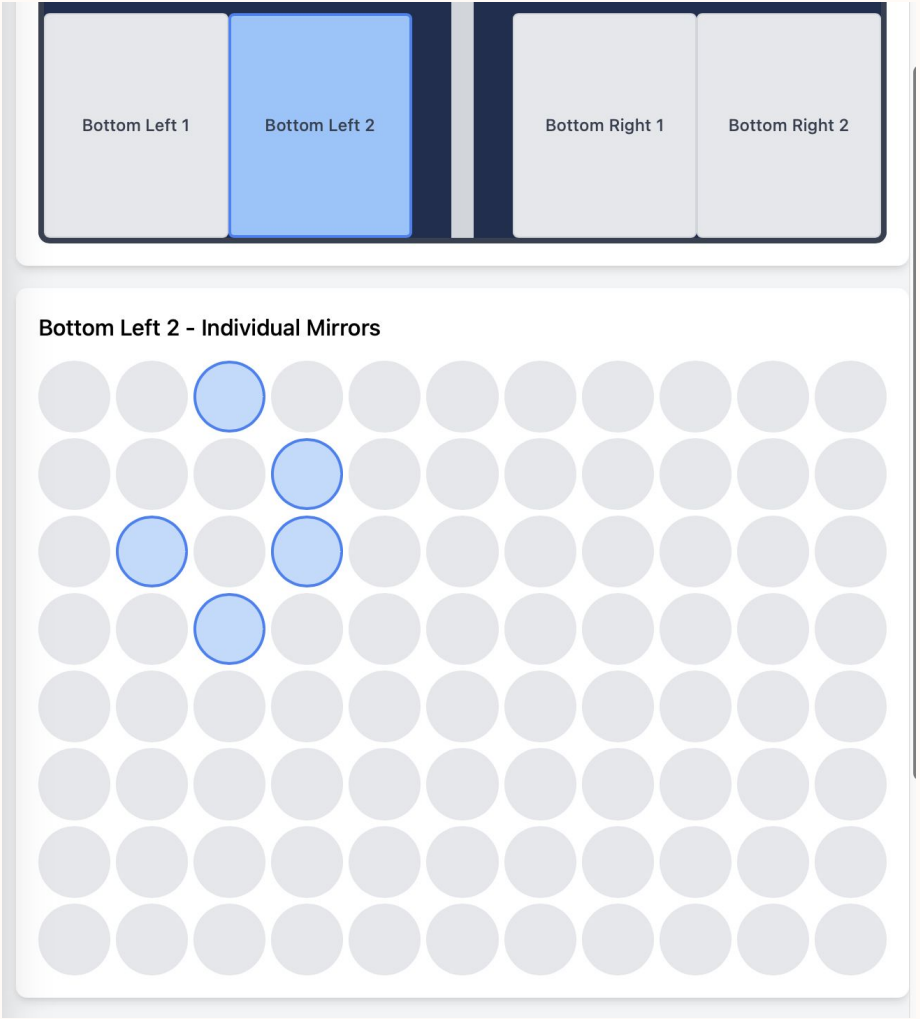
Control Overview

I used my proficiency in React web development to create a simple overview for how I wanted to mirror array to be presented for fine tuned control.



Mirror Subsection

When a user selects a subsection of the mirror array, they are presented with a fine tuned control.



Pan/Tilt Control

With Mirrors selected, users can control the pan and tilt.

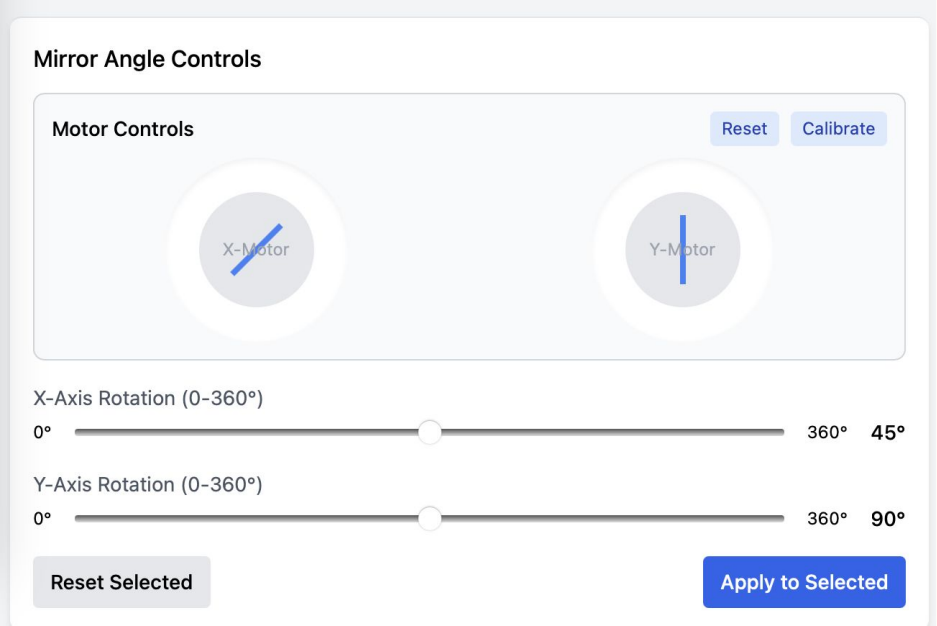
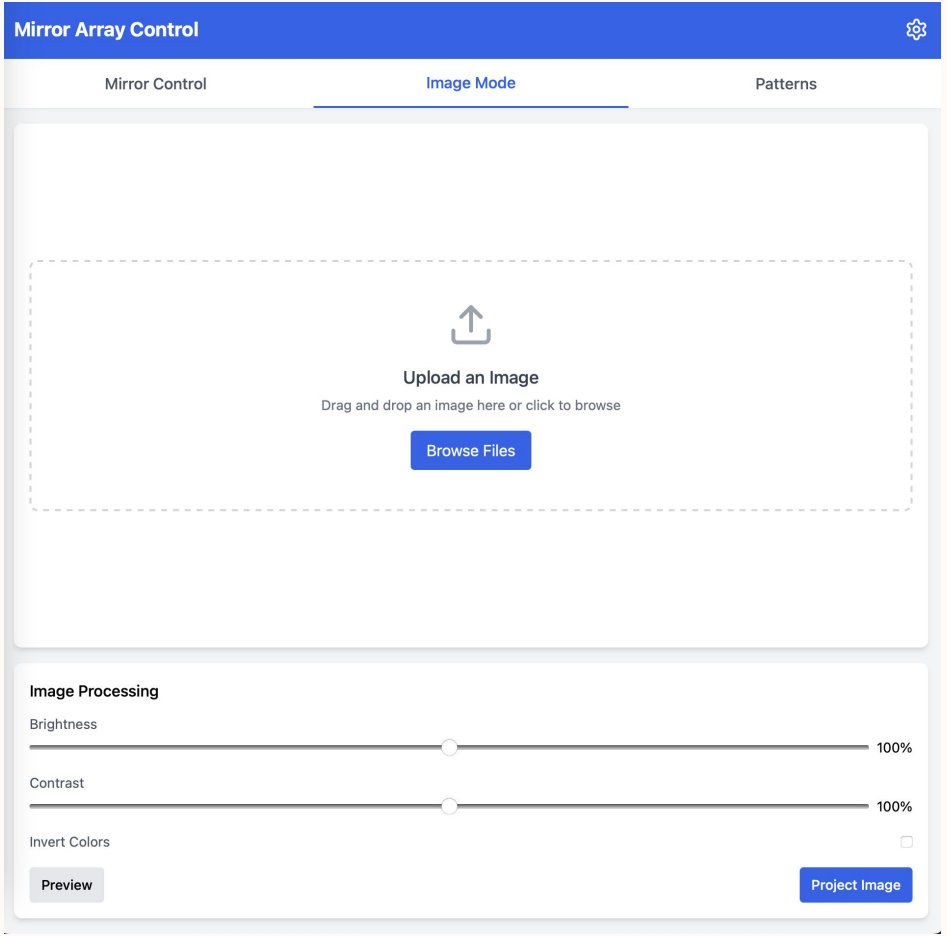


Image Uploader

Users can upload and an image to be processed by the Grasshopper / Python projection script.

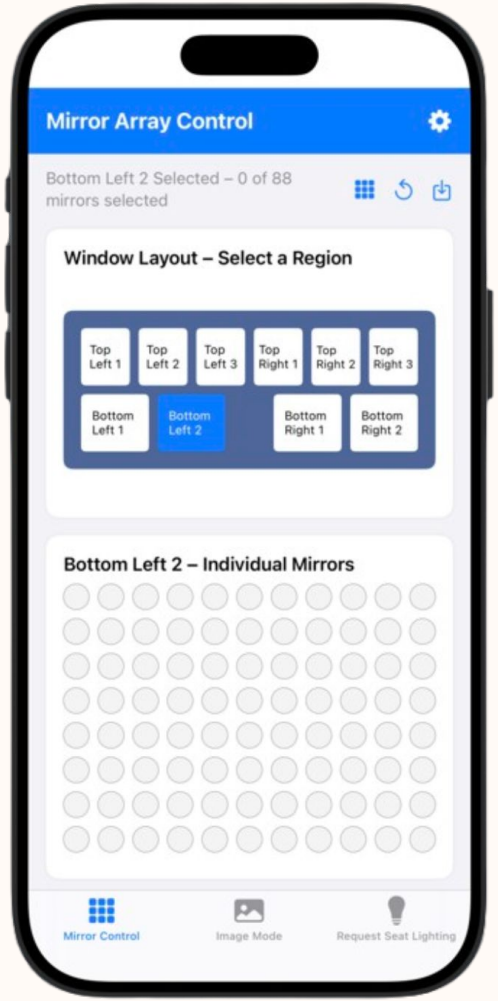


06

SwiftUI App

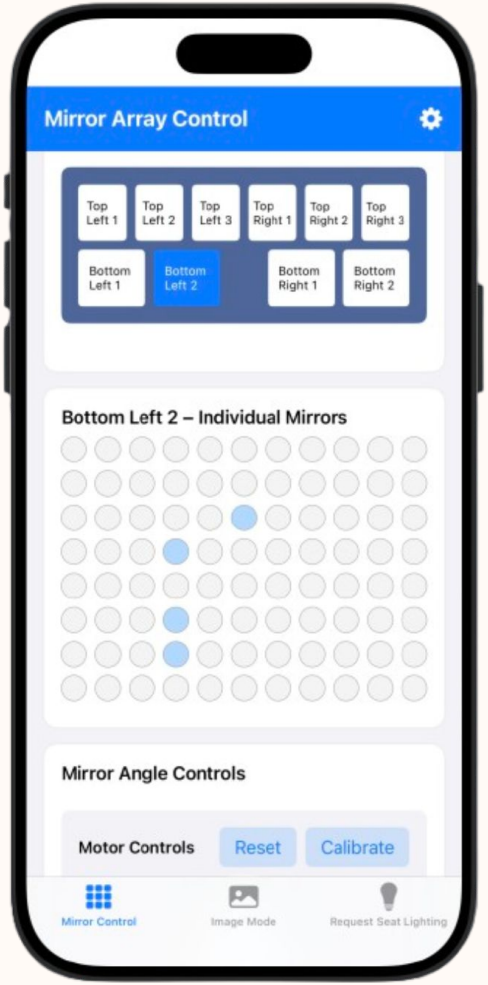
Control Overview

Combining everything I learned in SwiftUI – Here is the wireframe design working in SwiftUI



Mirror Subsection

When a user selects a subsection of the mirror array, they are presented with a fine tuned mirror selector.



Pan/Tilt Control

With Mirrors selected, users can control the pan and tilt.

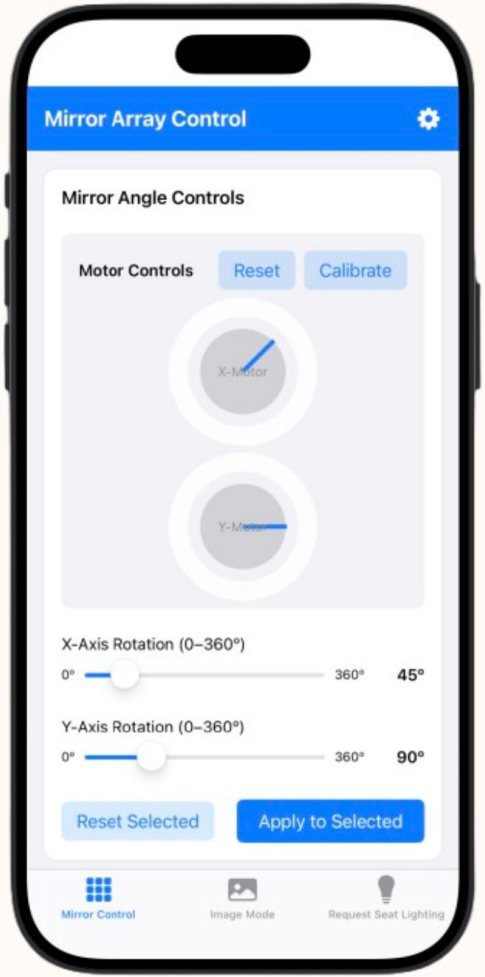
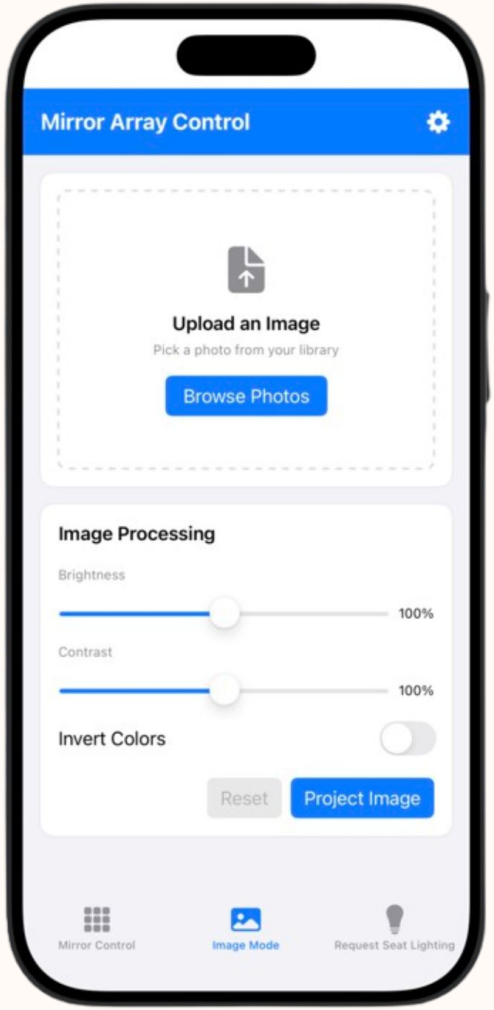


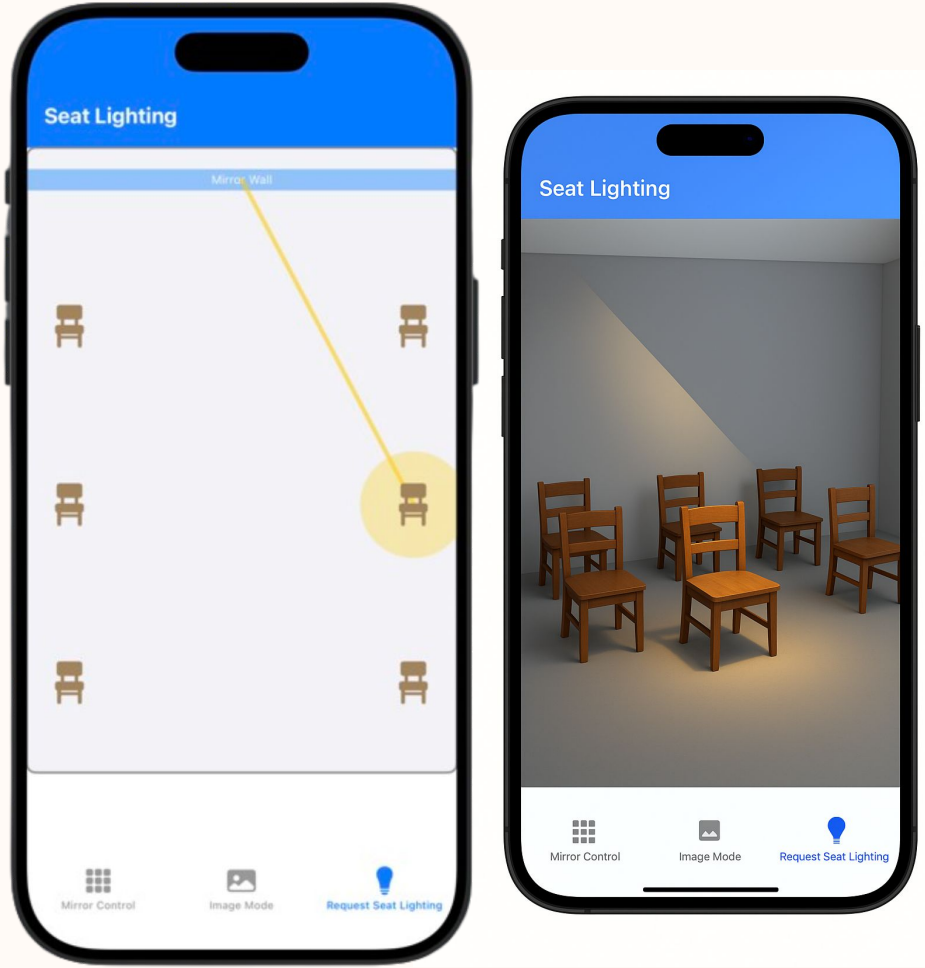
Image Uploader

Users can upload images from their camera roll, do basic edits, then “project” said image to the mirror array.



Chair View

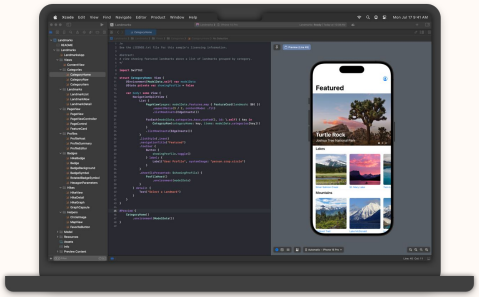
Prior conversation with Roger, where we talked about the ability to request lighting to your seat to help you read. Interface allows you to do just that.



07

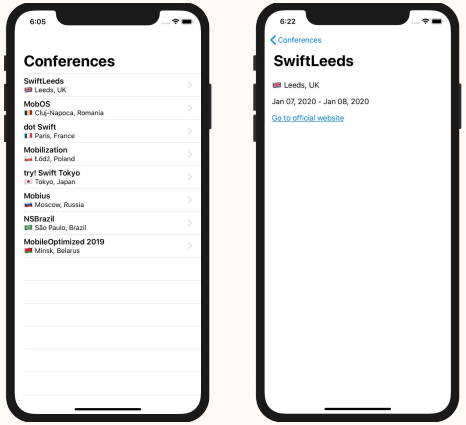
Conclusions & Future Work

Recap



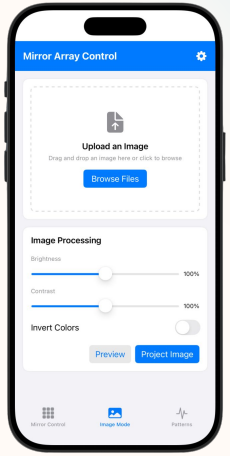
Learned SwiftUI

Watched videos weekly, learning new features as Apple creates new updates.



Built with SwiftUI

Building sample apps, new ideas, testing different approaches, and researching common wireframes.



Applying SwiftUI

Used new skills combined with previous experience to build a “remote control” interface for the mirror array.

Automatic Image Processor

- Adapt Chandler's work in Grasshopper into a Python Script or wrap the Grasshopper script in Python

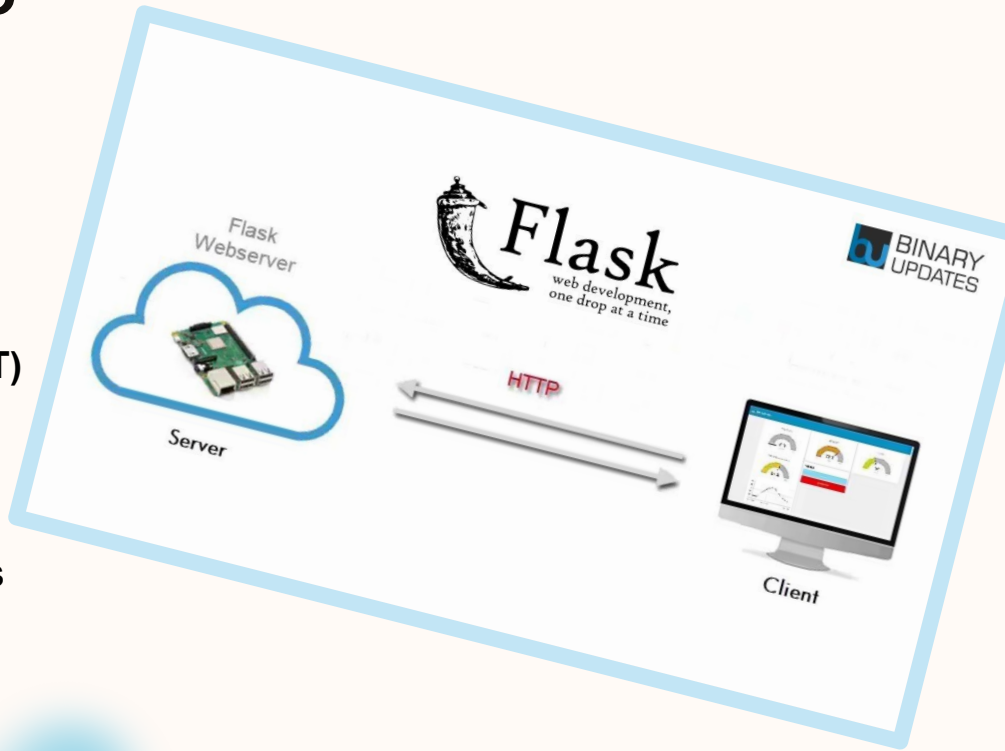
- Make that Python Script available via flask server

- Process and Change Mirrors via the app image uploads

```
1. read image with cv2.imread
2. convert to grayscale → flatten → argsort() to pick N brightest pixels
3. for each pixel
      • map to 3-D target point (ceiling plane transform)
      • calc t = (p - mirror_pos) / ||…||
      • get current solar vector s (pvlib, NREL SPA, or simple formula)
      • n = unit(s + t)
      • pan = atan2(n·x, n·z) ;  tilt = asin(-n·y)  # example axis choice
      • steps  = round(pan/step_size), round(tilt/step_size)
4. write CSV or send directly over the network
```

Connecting the App to the Python Script

- Run a Python script as a **Flask web server**.
- The phone app makes **HTTP requests (POST)** to the server with commands.
- The script processes the command and sends a response to the app letting the user know of success.
- Python script proceeds with sending required arguments to adjust mirrors

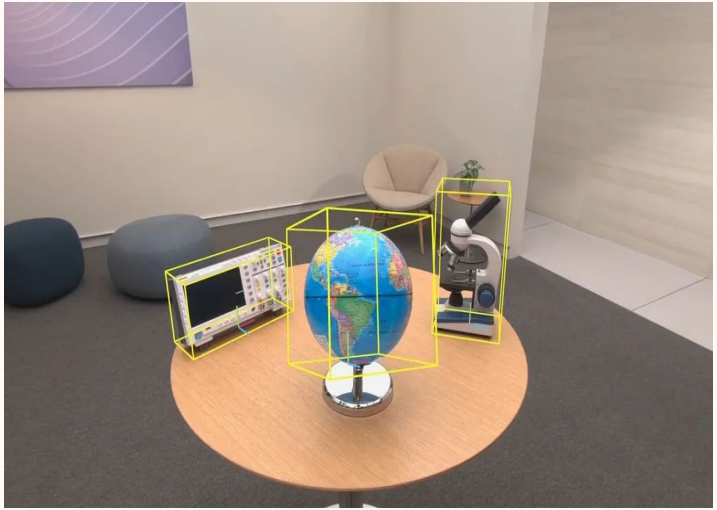


3D Visualization of the Space

- WWDC 2025 will likely see the release of VisionOS 3 which will bring many new 3D features to the Swift ecosystem.

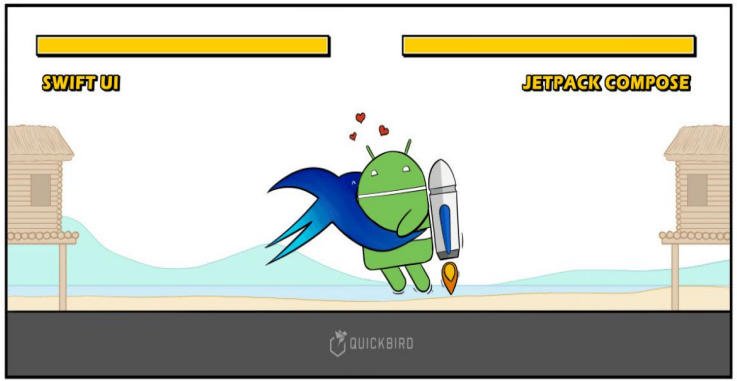
- Realitykit and ARkit could be used to interact with the mirror array in a locally running 3d representation

- This could be used to select objects in a space to focus the light on, like a person sitting in a chair.



Android Support

- Not everyone has an iPhone, and there is no cross capability with android and SwiftUI apps.



- The optimal approach would be to rewrite the app in Kotlin and jet pack compose for the best support.

- Two separate projects for iOS and android would need to be maintained.

08

Thank you! Questions?