In order to provide a better and sustainable user experience, we need to be able to have applications for different platformas that allow a user to connect with each Curiyo Album and play that album, or even connect with a numer of Curiyos and make a playlist from the albums.

The app must also have a cool way of connecting with the Bluetooth speakers available, so the user doesn't have to go to the settings in their devices. It should all be contained in the app.

The catalogue can be called the Curiyo Cabinet and will have not only the albums, but also the albums' extra content. The lyrics, pictures, video, education materials, special content, etc.

To create CuriyoApp

Set up a new iOS project in Xcode, and select the appropriate project template.
Organize the workspace, creating folders for models, views, and controllers, as well as utility classes and resources.
Begin by implementing the data model and networking code to fetch the album catalog from the Curiyo API.
Set up the URLSession and create Codable structs to map the JSON response to the Album object. Once the networking code is in place, test the API call to ensure that the data is being fetched and parsed correctly.
Next, focus on the Bluetooth connection page.
Import the CoreBluetooth framework and create the BluetoothManager class.
Write code to scan for nearby Bluetooth devices, connect to the selected device, and handle any Bluetooth-related errors.
Once the Bluetooth connection is stable, move on to the media playback controls.
Import the AVFoundation framework and create the PlayerManager class.
Write code to initialize the AVPlayer instance, load the selected track, and provide standard playback controls like play, pause, and next track.

User interface.
Design and build the interface using Interface Builder and Storyboards, ensuring a seamless and enjoyable user experience.

Incorporate custom UI elements and animations to bring the app to life.
Detail:

Set up a new iOS project in Xcode, organize the workspace, and import necessary frameworks (CoreBluetooth, AVFoundation, etc.).

1. Design the user interface:
    - Create a main screen that displays the catalog of Curiyo albums using UICollectionView or UITableView.
    - Implement an album detail screen with relevant information, lyrics, images, and video content.
    - Design a Bluetooth connection screen to list available Bluetooth devices and connect to the selected speaker.
    - Build a media playback control screen with play, pause, skip, and volume controls.
2. Implement the data model and networking code:
    - Create Album and Track structs to store album and track information.
    - Set up URLSession to fetch album data from the Curiyo API and parse JSON response using Codable.
3. Develop the Bluetooth connection functionality:
    - Create a BluetoothManager class that handles the discovery, connection, and disconnection of Bluetooth devices using the CoreBluetooth framework.
4. Implement the media playback controls:
    - Develop a PlayerManager class that uses AVPlayer from the AVFoundation framework to manage audio playback.
    - Provide play, pause, skip, and volume controls.
5. Perform thorough testing and debugging to ensure the app functions smoothly and provides a seamless user experience.
6. Submit the app to the App Store for review and release.

Draft Coding process
basic setup for a new iOS project in Swift using Xcode:

7. Launch Xcode and create a new project: Select "Create a new Xcode project" on the Xcode welcome screen or go to File > New > Project in the menu bar.
8. Choose the project template: Select "App" under the "iOS" tab and click "Next".
9. Configure your project:
    - Product Name: CuriyoApp
    - Team: (Choose your development team if you have one)
    - Organization Name: (Your organization name)

- Organization Identifier: (Your organization identifier, usually in reverse domain name format)
- Interface: SwiftUI or Storyboard (depending on your preference)
- Language: Swift

Select "Use Core Data" if you plan to use it for data persistence.

- Click "Next" and choose a location to save your project.

10. Organize your workspace:
    - Create folders to organize your files (e.g., Models, Views, ViewControllers, Networking, Utils).
    - Move existing files into their respective folders.

Import necessary frameworks:

11. In the files that require the CoreBluetooth or AVFoundation frameworks, add the import statements at the top of each file:

```
swift
import CoreBluetooth import AVFoundation
```

Step One: setting up the UI
For setting up the UI using SwiftUI for the Curiyo app's main views (catalog view, Bluetooth speaker connection view, and media controls view):
First, create a new SwiftUI file called `ContentView.swift`, which will be the main view of the application. In this file, you'll create a TabView to switch between the three main

views:

```
import SwiftUI


struct ContentView: View {
    var body: some View {
    TabView {
      CatalogView()
                .tabItem {
                    Image(systemName: "music.note.list")
                    Text("Catalog")
                }

            BluetoothConnectionView()
                .tabItem {
                    Image(systemName: "speaker.wave.2")
                    Text("Speakers")
                }

            MediaControlsView()
                .tabItem {
                    Image(systemName: "playpause.fill")
                    Text("Controls")
                }
        }
    }
}


struct ContentView_Previews: PreviewProvider {
  static var previews: some View {
    ContentView()
    }
}
```

Next, create three separate SwiftUI files for each of the main views: `CatalogView.swift`, `BluetoothConnectionView.swift`, and `MediaControlsView.swift`. Here's a simple starting point for each of these views:

`CatalogView.swift`:

```swift
import SwiftUI

struct CatalogView: View {
    var body: some View {
        NavigationView {
            List {
                // Add your list of albums here
                Text("Album 1")
                Text("Album 2")
                Text("Album 3")
            }
            .navigationBarTitle("Curiyo Catalog")
        }
    }
}

struct CatalogView_Previews: PreviewProvider {
    static var previews: some View {
        CatalogView()
    }
}
```

`BluetoothConnectionView.swift`:

```swift
import SwiftUI

struct BluetoothConnectionView: View {
    var body: some View {
        NavigationView {
            List {
                // Add your list of available Bluetooth speakers here
                Text("Speaker 1")
                Text("Speaker 2")
                Text("Speaker 3")
            }
            .navigationBarTitle("Bluetooth Speakers")
        }
    }
}

struct BluetoothConnectionView_Previews: PreviewProvider {
    static var previews: some View {
        BluetoothConnectionView()
    }
}
```

`MediaControlsView.swift`:

```swift
import SwiftUI

struct MediaControlsView: View {
    var body: some View {
        VStack {
            // Add your media controls such as play, pause, skip, and volume
adjustment here
            Button(action: {
                // Implement play/pause functionality
            }) {
                Image(systemName: "playpause.fill")
                    .font(.largeTitle)
                    .padding()
            }
        }
        .navigationBarTitle("Media Controls")
    }
}

struct MediaControlsView_Previews: PreviewProvider {
    static var previews: some View {
        MediaControlsView()
    }
}
```

These basic templates set up the main views for the app. You will need to expand upon these templates to incorporate the necessary functionalities such as fetching album data, connecting to Bluetooth speakers, and controlling media playback.