universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Introdução à Computação Móvel

EchoSync

André Ribeiro
NMec: 112974

Rúben Garrido
NMec: 107927

Violeta Ramos
NMec: 113170

June 10, 2025

Contents

1 Introduction

EchoSync is a music mesh application that creates a synchronized multi-device audio system, allowing users to connect multiple smartphones, tablets, and computers to play music simultaneously with perfect synchronization.

The app transforms a collection of ordinary devices into a powerful distributed speaker system, eliminating the need for expensive audio equipment while providing an immersive listening experience. As a result, EchoSync is perfect for people who want to use their devices and still effortlessly set up a multi-room audio experience.

Any change on one device gets reflected on all the other nodes. Users can collaboratively manage playlists, and control other things such as playback and volume from any device within the network.

For a more immersive experience, EchoSync also allows gestures that turn into actions: for example, flipping the phone with the screen facing down will pause music, while facing up will resume again.

2 Architecture

2.1 Information layers

This app follows a BLoC pattern, with three layers of information:

- The data layer introduces the business logic of the app, and is distributed across services. Each service has its responsibility and coordinates a part of the app. For example, one of the services deals with MQTT, while another manages the queue and playback.
- The BLoC layer is the bridge between services and the UI. It is responsible for creating reactivity on the UI, and handles the callback events from it, back to the services.
- The UI layer refers to the views (such as pages, made with Widgets), which allow the app to display information from previous layers to the user. The views get their state through BLoCs.

2.2 Data storage & transmission

Data isn't stored in the cloud, ever. EchoSync is offline first, and doesn't require a connection to the Internet. All it requires is a local network — either from a Wi-Fi access point or a mobile hotspot.

To achieve such functionality, this app features a local MQTT server on each device, on the background. This means that, because every device has its own MQTT client and server, everyone can be both a leader and a follower, eliminating the need for an external MQTT server. This also creates some redundancy, since the failure of one device means that anyone else can take the lead and coordinate the group.

2.2.1 Song distribution

When it comes to audio distribution, sending files to other devices has a problem: songs are too big to be sent using MQTT. Splitting the file into chunks would be an alternative, but that would require a bit more effort in order to guarantee integrity.

As a workaround, EchoSync identifies songs by their hash and creates a local HTTP server on each device, which makes audios available for anyone to download.

As an example, as soon as device B gets an MQTT message about a song with hash `a1b2c3d4`, it downloads it by making a GET request to device A, on the path `/echosync/a1b2c3d4`. In the end, when the download is completed, device B also hosts the file in its HTTP server, to achieve redundancy and failure protection.

2.2.2 Data retention

When new devices join the network and there is already music playing (i.e., a non-empty queue), data should be sent to those devices. To help with such task, retention is used for some MQTT messages, which ensure new devices can read old ones, which contain information about already playing songs.

2.3 Device discovery

For devices to discover themselves, we're using Flutter's `nearby_connections` plugin, as it uses Bluetooth Low Energy (BLE) to advertise itself and to allow others to discover and get connected.

The advertisement contains the IP address of the corresponding node, so users can connect to its MQTT server. Besides that, it includes the name of the phone and the number of nodes already joined to it, so that users can better understand and find the device they're trying to connect to.

As a fallback option (e.g. inter connections between Android and iOS), QR code is available. This one indicates the host name and its IP as well, so that joining devices can connect to the host device.

2.4 Sensors & mobile functionalities

- WiFi: Create mesh network between devices for synchronized playback without requiring internet connection
- Camera: Scan QR codes to quickly pair and connect devices to the mesh network
- Gyroscope: Enable gesture-based music control through natural hand movements (e.g. flip to pause)
- Light/proximity sensor: Pause music by putting your hand over one phone
- Audio system: Access local music libraries and provide synchronized audio output across all devices
- File system: Access and share music files from device storage across the network

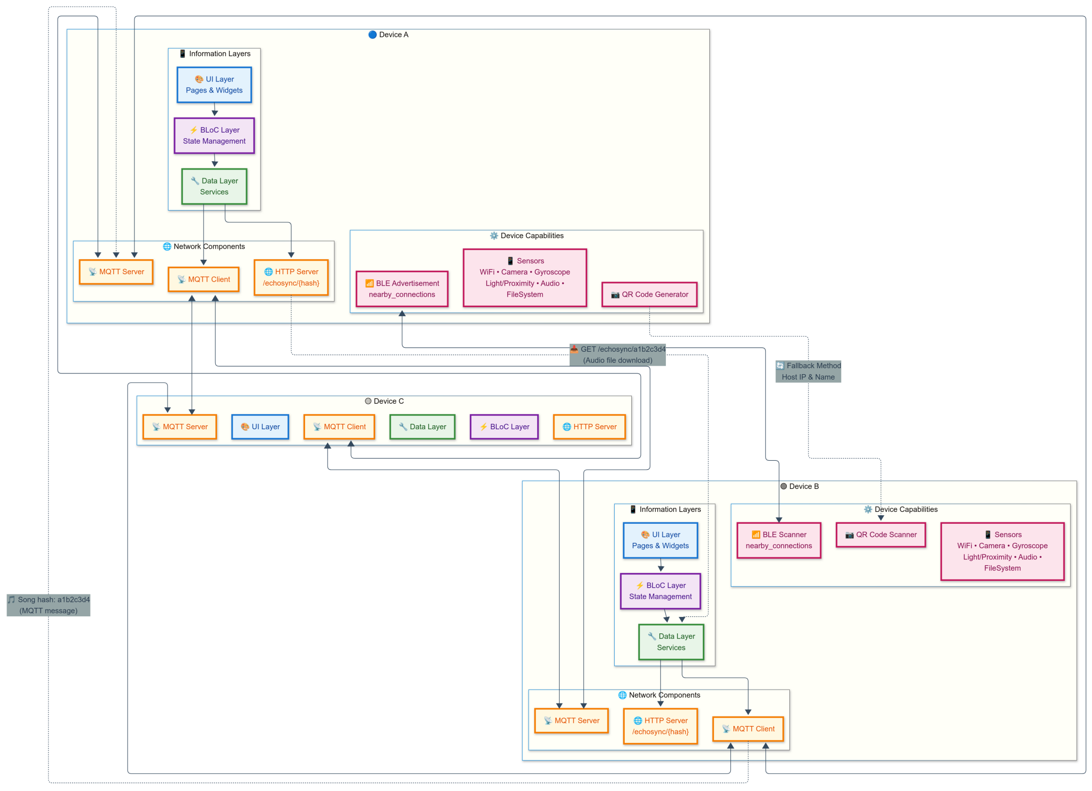

Figure 1: Architecture with 3 devices

3 Project limitations

Local network required

Although no Internet access is required for EchoSync to work, a local network connection is still required, either wirelessly or []. While this isn't a big problem (since nowadays pretty much every device can create a Wi-Fi hotspot), it still requires some configuration on all devices, such as connecting them all to the same network.

BLE inter-OS operability

The Bluetooth Low Energy feature to connect devices into the same MQTT server works perfectly when those devices use the same operating system (e.g. Android-Android or iOS-iOS). However, since each operating system uses its own protocol, they're incompatible between themselves. This means that connecting an Android and an iOS requires fall-backing to the QR Code method.

4 Post-project review

4.1 Lessons learned

As a team, we recognized Flutter and Dart as powerful tools for cross-platform development, enabling us to target multiple platforms (Android, iOS, web, desktop) from a single codebase. This significantly accelerated our development cycle and reduced platform-specific implementation costs. However, we also encountered challenges with ecosystem maturity: some libraries on pub.dev with high download counts and multi-platform claims (e.g., Android/iOS/Linux/macOS) exhibited unexpected platform-specific limitations, particularly on iOS. This was evident in GitHub issues remaining unresolved for months despite community reporting.

On a positive note, we found Flutter's hot reload feature to be a game-changer for our development workflow. It allowed us to quickly see the results of our code changes without restarting the app, significantly boosting our productivity. Furthermore, Flutter's rich set of customizable widgets enabled us to create a visually consistent and attractive user interface across all platforms with a single codebase.

4.2 Work distribution within the team

Everyone contributed equally for this project, which was very beneficial for the successful outcome of EchoSync.

5 Conclusion

EchoSync represents a successful implementation of a distributed music synchronization system that transforms ordinary mobile devices into a coordinated speaker network. The project demonstrates innovative use of mobile technologies, combining MQTT messaging, HTTP file distribution, and Bluetooth Low Energy for device discovery to create an offline-first, multi-device audio experience.

We successfully leveraged Flutter's cross-platform capabilities to build a solution that works across Android, iOS, web, and desktop platforms, while implementing creative gesture-based controls using device sensors like gyroscopes and proximity sensors. Despite encountering limitations with cross-OS BLE compatibility and some ecosystem maturity challenges, the project achieved its core objective of synchronized audio playback without requiring expensive equipment or internet connectivity.

5.1 Project resources

Resource	**Available at**
Code repository	https://github.com/RGarrido03/EchoSync
Production APK	https://github.com/RGarrido03/EchoSync/releases

Table 1: Project resources