Bachelor's thesis

# The CountaBLE Attendance System using Bluetooth Low Energy (BLE)

**Yiannis Hadjiyiannis**

**UNIVERSITY OF CYPRUS**

**DEPARTMENT OF COMPUTER SCIENCE**

**May 2025**

**UNIVERSITY OF CYPRUS**
**DEPARTMENT OF COMPUTER SCIENCE**

# The CountaBLE Attendance System using Bluetooth Low Energy(BLE)

**Yiannis Hadjiyiannis**

Supervisor
Prof. Demetris Zeinalipour

The thesis was submitted in partial fulfillment of the requirements for obtaining the undergraduate degree in Computer Science from the Department of Computer Science of the University of Cyprus.

May 2025

## Acknowledgments

First, I would like to express my gratitude to my supervisor Prof. Demetris Zeinalipour for his continuous guidance throughout the year. The communication between us consistently provided me with valuable feedback, pushing me to achieve great results.

I also would like to express my gratitude to my friends and family for providing me with continuous unconditional love and support throughout this journey. Having them by my side, helped me maintain a clear and sharp mind by keeping me motivated. I could not have done this without them.

## Abstract

Attendance tracking in educational and organizational environments remains a big concern throughout the years, where accuracy and reliability matters. In modern days, the classic pen-and-paper solutions have proved to be inconvenient for both organizers and participants, leading to the development of various attendance systems utilizing different technologies with the goal of automating the process for greater reliability and convenience. Attendance systems using different technologies have been proposed and used, including manual sign-ins, QR Codes, RFID tags and Wi-Fi signal based where each system introduces different levels of automation and accuracy, but eventually suffer from common drawbacks such as, requiring additional hardware, relying on existing infrastructure, user inconvenience and fraudulent behavior.

In this thesis, the CountaBLE system is proposed looking to offer an automated solution to address these limitations by leveraging Bluetooth Low Energy (BLE) technology, supported by the majority of modern smartphone devices. The system, by utilizing BLE's power efficient technology along with the Generic Attribute Profile (GATT), it enables a peer-to-peer device discovery and data communication between an Organizer and multiple Participants. By following this approach, the system not only operates fully without internet connection, but also guarantees that attendance messages can only be exchanged through the user's personal smartphone device to significantly reduce the risk of remote attendance fraud. The use of a unique per-app device identifier during communication, enhances the reliability in recurring attendance tracking sessions allowing a consistent recognition of participants without the use of any device sensitive data, ensuring user privacy.

The system developed was built using a cross-platform approach in Ionic Framework, to ensure consistent user experience across different platforms while also minimizing the development costs. With full native Android functionality developed, the architecture of the system is in place for future iOS compatibility. For android platforms, the system makes use of Android's foreground service model allowing BLE operations to actively work in the background ensuring minimal user interaction.

# Contents

# Chapter 1

# Introduction

Attendance tracking has been a concern in institutional and organizational environments, especially in academic contexts where accurate records are important for various reasons, including compliance, statistical reasons, but also performance evaluation. The process of tracking attendance has relied on many forms such as manual sign-ins or even pen-and-paper evaluation by the organizer of the specific environment. While simple in theory, these attendance tracking techniques can often prove unreliable due to inaccuracies and inefficiencies since they are prone to human error, especially when an Organizer is controlling a large amount of participants. As a result, the need for automated attendance systems has gained increasing attention, mainly in scenarios where mobile computing and Internet of Things (IoT) technologies are involved.

Over the years, several technologies have been proposed and used in order to make attendance tracking less intrusive but also more reliable. This includes technologies that are based on RFID Tags, QR Code scanning or even Wi-Fi RSS (Received Signal Strength) where each technology presented partial solutions, making the attendance tracking process far more simple. However, these technologies inevitably address some limitations and introduce new challenges. For instance, RFID Attendance Systems - a solution seen in various Universities, requires specialized hardware and infrastructure, both for the University and each Student. QR Code based attendance systems, while affordable, suffer from inaccuracies since they rely on camera quality and can sometimes suffer from visibility issues. WiFi RSS techniques, even though they may be less intrusive for the user, can suffer from an unstable signal and require constant calibration. All these systems, carry trade-offs in terms of scalability, hardware requirements and user efforts.

Another big challenge in many automated attendance systems solutions is the vulnerability to fraudulent behavior. From the technologies referred such as the use of RFID and QR Codes, while they are widely adopted, they often face the problem of not enforcing the true presence of the participant. For instance, when considering RFID solutions, a participant can easily share their personal RFID Tag which serves as their personal identification with another Participant in order to register attendance on their behalf. Similarly, when considering QR code solutions, the organizer will often share a static QR Code on a screen to the participants. These QR codes can be captured and then easily shared amongst other participants that are not truly present. These limitations pose a big threat in reliably tracking a participant's presence.

In this thesis, an automated attendance system is proposed using Bluetooth Low Energy (BLE). BLE is a modern, standardized wireless communication protocol available on most of the smartphone devices on the market including both Android and iOS devices. The system, called CountaBLE Attendance System, leverages the benefits of using BLE Technology to tackle the various limitations exposed by other solutions in order to provide a robust, reliable system to enhance the attendance tracking process for both Participant and Organizer. Given the widespread adoption of smartphone devices in modern society, the system proposed can be considered infrastructure free since it requires no additional hardware. Moreover, it requires no internet access and supports power-efficient device discovery and communication over short distances.

The system leverages each participant's personal device as the BLE medium used for submitting attendance, requiring their true presence. This significantly reduces the feasibility of remote attendance. To strengthen this, the system employs the use of a unique per-app identifier that is generated per app signing key, making it a practical choice for identifying participants in a consistent manner while also avoiding any privacy and security issues that could arise during the communication of the organizer with a participant. In a scenario where recurring attendance sessions take place, this implies that when a participant submits their attendance to an organizer, the same identifier will be expected in every session allowing the system to uniquely identify valid participant devices.

Bluetooth Low Energy (BLE) along with the use the Generic Attribute Profile (GATT) plays a central role in enabling the design and functionality of the proposed attendance system. When using GATT, one central device called the GATT server, takes the role of the Organizer where it broadcasts advertisement packets while several other devices take the role of GATT Client which represent the Participants, and actively listen to those advertisement packets used to initiate an actual connection between them for data exchange, allowing a seamless peer to peer device discovery and communication.

The system developed, adopts a cross-platform development approach to reduce the complexity of the development process but also to increase user experience. By doing so, we avoid maintaining separate codebases for different platforms, enabling the development of just a single codebase that compiles to both Android and iOS platforms, leveraging the use of standard web technologies to create the user interface, and reducing the costs of the development needed for native functionalities. The system presented in this thesis supports full native functionality for Android platforms, but due to the nature of the cross-platform approach, future support for iOS platforms can be done by just developing the native functionalities using the Core Bluetooth iOS API.

Lastly, to achieve the goal of requiring minimal user effort to record attendance, the system developed makes use of Android's foreground service model, allowing BLE operations to persist in the background in a power-efficient manner, allowing continuous, uninterrupted communication between organizer and participant devices without user interaction.

## 1.1 Architecture Overview

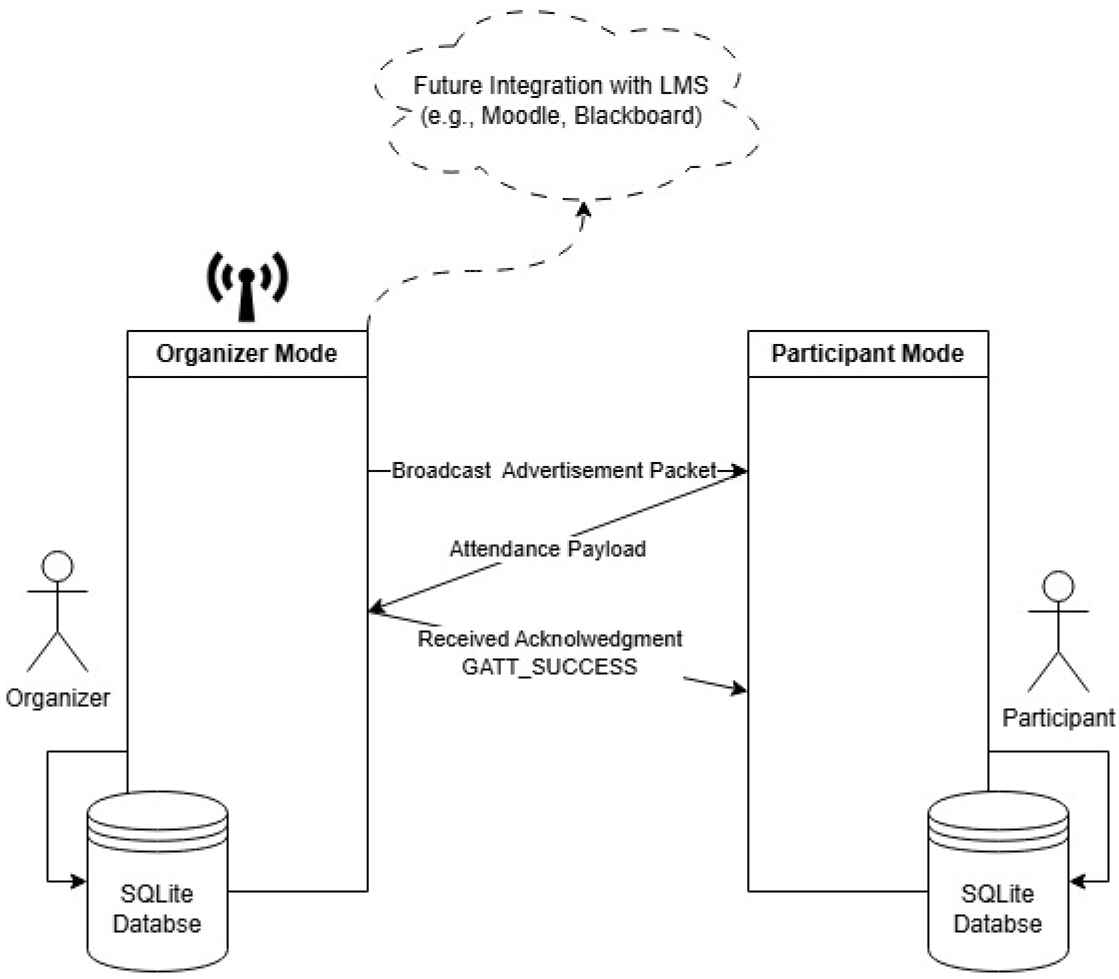


Figure 1.1: System Diagram

The proposed system is realized through a mobile application developed to operate in two modes: Organizer and Participant. These two modes represent the two entities interacting in an attendance marking scenario to facilitate an automated attendance system. In each session, an Organizer operates in Organizer Mode to initiate the process of broadcasting advertisement packets, used for participants to identify the correct session. Upon recognizing the appropriate packet, Participants may choose to establish a connection and transmit their attendance payload to the Organizer. Once the Organizer receives the payload from a Participant, it responds with an acknowledgment message to confirm the received attendance. The exchange of these messages, allows for both Organizer and Participant to store their events in a local database. This is done offline where both Organizer and Participant use an internal SQLite database embedded in the application. The communication between devices operating in the two distinct modes relies on a multi-layered abstraction that integrates front-end logic, native Android BLE APIs, and the low-level Bluetooth protocol stack.

The communication begins at the top User Interface layer which uses the Ionic Framework which uses Angular, allowing the use of standard web technologies for rapid UI developments while also maintaining a consistent, native-like user experience. In the Organizer Mode, users interact with the application to create different group sessions and, initiate and attendance sessions. In Participant Mode, users can scan for attendance group sessions and view the results in real time. Navigational structures such as drawers, stacks, and modals are used to create a user-friendly and intuitive experience while effectively managing the state of the application.

The Native Android layer handles the core BLE operations by interacting with the Android's Bluetooth stack. These commands are eventually expressed over low-level Bluetooth Protocols such as L2CAP, Attribute Protocol (ATT) and the Generic Attribute Profile (GATT) which is built on top of ATT. The User Interface Layer passes the appropriate parameters to the Native Layer through Capacitor, a cross-platform runtime that enables communication between web code and native code. Using these parameters, BLE operations are offloaded to native Android Java classes, which interact with the Android BLE APIs.

To ensure the persistence of attendance data on both the organizer and participant devices, the system implements a lightweight SQLite database embedded in the application. This database manages information for groups, participants, and attendance record, with data consistency enforced through foreign key constraints. For the Organizer, this ensures that each group is stored successfully on the device, with the relevant participants showing in each group session. For the Participant, when acknowledgments are received from an Organizer for an attendance session, it saves locally the attendance information for evaluation and organization. Additionally, user preferences such as username or selected mode, are saved and managed via a separate preference storage component.

## 1.2 Thesis Structure

In the Related Work and Background section, we first introduce different attendance tracking systems using different technologies, for the purpose of recognizing the limitations and challenges that can arise and understand the need for a different system. Then, we introduce basic concepts of Bluetooth Low Energy and Cross-Platform Development to understand the components presented in this thesis.
The Communication Layer follows, where the communication between organizer and participants is decomposed into structured, multi-layered abstraction to understand the flow of the proposed system. Then, the Data Layer discusses the data management strategies, explaining the use of SQLite database and preferences storage with details.
The next two chapters, User Interface and Native Layers, include technical details and implementation of the proposed system.
In Experimental Evaluation, we conduct a series of experiments to evaluate the efficiency and performance of the proposed system.
Finally, in Conclusions and Future Work, after concluding the proposed system, we discuss some further enhancements that can be applied to the proposed system.

# Chapter 2

# Related Work and Background

## 2.1 Related Attendance Systems

The attendance system that was developed utilizes Bluetooth Low-Energy technology with a Generic Attribute Server (GATT). In the following sub-chapters, references are made to attendance systems that use different technologies for comparison purposes and understanding the need for a different system.

### 2.1.1 Attendance Systems using WiFi (RSS)

A system developed by Khan, Haque, Tabassum, and Rahman in 2014 leverages existing Wi-Fi infrastructure to reduce implementation costs and provide an effective alternative to traditional attendance methods, such as handwritten lists or biometric systems, which are expensive and complex.

Rather than trying to locate the exact location of users, the system relies on zone detection, where the room is divided into a grid of zones, and for each point, Wi-Fi signal strength data from multiple access points (APs) is collected. This data is used to create a ”radio map” (fingerprint) during the preparation phase. Then, when a student or teacher uses the application to record their attendance, the current Wi-Fi data is compared to the map, and the system decides whether the device is inside or outside the boundaries of a room.

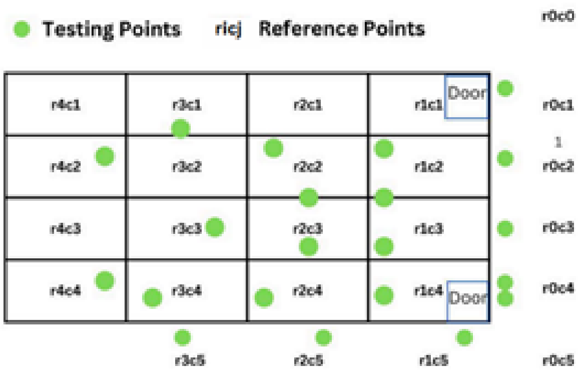


Figure 2.1: Radio Map (Fingerprint) [1]

### 2.1.2 Attendance Systems using QR Codes

WeConnect is an attendance recording system for classrooms, developed for the National University Laguna (NU Laguna) with the aim of automating the attendance tracking process. The system uses mobile-tethering technology between smart devices and QR codes, for practicality and economy compared to more traditional methods (e.g., RFID, Barcode Scanners) where these require additional hardware.

The system was implemented as an Android application that allows attendance recording either through a tethering connection with the teacher’s device or by scanning unique QR codes that are created uniquely for each course. The users of the application, namely teachers and students, have different levels of access: teachers can manage courses and export files, while students can view and be informed about their attendance.

### 2.1.3 Attendance Systems using RFID

Meili Liu and Yongge Yao propose an attendance recording system using radio frequency identification (RFID) technology to automate attendance recording in university classrooms. The system is based on RFID for rapid identification and recording of attendance, combining microprocessor, communications and data analysis technology, offering efficiency and time saving during teaching. The system includes three main functions: data collection, attendance data analysis and information extraction. The student has an RFID card with a unique identifier, which he scans when entering the classroom and a microprocessor controls the process, while the system integrates an LCD screen to display the recorded attendance data.

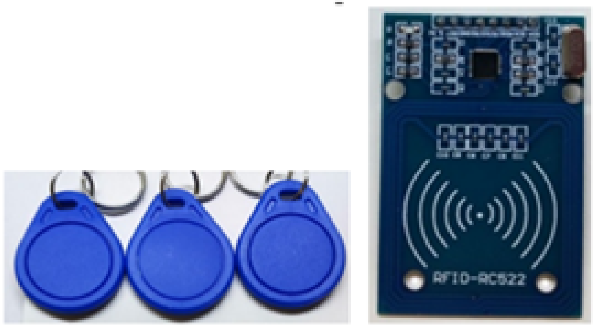


Figure 2.2: RFID Tag & Reader [4]

## 2.2 LTE Direct

LTE Direct is a device-to-device communication protocol that was designed for LTE operated devices to discover and communicate with each other by leveraging the existing LTE radio interface of mobile devices, allowing devices to broadcast and scan for discovery messages. LTE Direct was designed to operate in two different modes. The **Network-Assisted Mode** which involves a central LTE base station like eNodeB where it schedules and authorizes transmitted messages on active time slots . **Direct mode** on the other hand, allows devices to communicate without relying on existing networks infrastructure, meaning no device is connected to LTE base stations. Devices in this mode must be pre-configured with the appropriate parameters in order to be able to communicate without the assistance of the network. This mode was mainly designed for public safety communication in environments where connection to a LTE base station was not available.

## 2.3 Comparison of Different Technologies

Each technology discussed that is used in the various systems has unique advantages, but also presents specific challenges. Understanding the different approaches to attendance systems helps us understand which approach is appropriate depending on the needs of the system.

A system using RFID technology, provides a good degree of reliability and practicability. It doesn't rely on any network infrastructure like Internet access, but solely on the hardware equipment needed like an RFID reader and tag. It is also easy to use, as you do not handle any installation by the user (only by the administrator to configure the card). However, this makes the need for hardware equipment essential, making it a more expensive solution. On top of that, in scenarios where accuracy and true presence matter, it allows easily for fraudulent behavior since participants can hand around their unique identification tags.

Using QR Code technology also has its own advantages. It has very little setup and installation cost, especially on the participant side. The organizer is provided with a QR Code for an attendance session, which is then shared to participants who use their mobile devices to scan and submit the attendance. This means that no existing hardware infrastructure is required at all by the participant. Nevertheless, this means that it relies on Internet access to be available for both organizer and participant to access the online resource used to submit the attendance. In terms of attendance fraud when considering large crowds since participants can easily share the QR Codes provided.

When it comes to WiFi-based attendance systems, fraudulent behavior is much harder since it uses the WiFi RSS data of a participant to determine wether they are truly present in the room. However, it relies heavily on existing network infrastructures and requires constant calibration for each room to create the radio map.

The attendance recording process using BLE tackles most of the drawbacks found in other technologies. It requires no additional hardware infrastructure given the widespread adoption of smartphone devices in modern society therefore, Participants and Organizers can use their own personal mobile devices. Fraudulent behavior is also harder to achieve, since it requires the true presence of the participant in order to submit their attendance through BLE. On top of that, it can be implemented to require minimal user input from the participant by developing BLE operations as background services.

Table 2.1: Comparison of Attendance Tracking Technologies

| **Technology** | **Power Consumption** | **Reliability** | **Accuracy** | **Implementation Cost** |
|---|---|---|---|---|
| Wi-Fi RSS | Moderate | Moderate | Floor-level | Low but heavy calibration needed |
| QR Codes | Very Low | Low | Line-of-sight only | Very Low |
| RFID | Very Low | High | Moderate | Medium (tags and readers) |
| LTE Direct | High | Low | Low | High |
| **BLE** | **Low** | **High** | **Room-level** | **Low** (mobile devices) |

## 2.4 Security Concerns

The paper ”MiniBLE: Exploring Insecure BLE API Usages in Mini-Programs” examines the security issues associated with the use of BLE APIs in mini-programs. In this paper, the researchers deal with the WeChat platform. Mini-programs, which are lightweight applications that run within larger applications, offer convenience and functionality, but also bring with them serious security issues, especially when it comes to interacting with IoT devices via BLE.

To detect insecure uses of BLE APIs, the research develops the MiniBLE tool, which is a static contamination analysis tool that evaluates mini-programs to identify BLE-related security issues. MiniBLE was analyzed on more than 41,000 real mini-programs and proved effective in identifying potential vulnerabilities.

The researchers identified particular security issues in BLE connections, especially regarding the pairing process. A critical issue identified was the use of the wx.makeBluetoothPair

API, which allows developers to set a fixed PIN without user input, which reduces the security of the connectivity and introduces security risks later on. This reduces security at the pairing level, facilitating man-in-the-middle (MITM) attacks, as data can be intercepted or modified while traveling between devices. In addition, systems that use the BLE API primarily use simple registration and recognition of BLE characteristics, without proper encryption. This means that while a characteristic may be readable or writable, the transmission of the data may not be encrypted. The lack of encryption allows data sent to and from a BLE device to be vulnerable to being read or modified, especially in a man-in-the-middle (MITM) attack scenario.
The research emphasizes the importance of improving security methods in the use of BLE in mini-programs and suggests the development and implementation of more secure methods for managing communication and device pairing via BLE.

## 2.5 Background

### 2.5.1 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE), was introduced in 2010 as part of the Bluetooth 4.0 specification in 2010, to meet the need for a wireless technology, capable of exchanging small amounts of data in a way that is power efficient. BLE devices, most of the time, are idle and only wake up for short amount of time in order to either broadcast its presence which is called advertising or listen to others which is called scanning. If necessary, it is also capable of establishing a short-lived connection, where small data exchanges happen.
Therefore, two main modes form the basis of how BLE operates. Advertising and Scanning.

#### 2.5.1.1 Bluetooth Classic vs Bluetooth Low Energy (BLE)

When it comes to Bluetooth technology there are 2 main variants. One is Bluetooth Classic which is often referred to simply as ”Bluetooth” and then we have the newer Bluetooth Low Energy(BLE). Both of these technologies share most of their core principles. Both operate within dedicated 2.4 GHz ISM band channels but have quite different use cases.
Bluetooth Classic was designed for continuous, high-throughput data exchange. Therefore it is ideal for purposes that demand continuous data exchange such as streaming audio

to wireless headphones, file transfers between devices or for connecting peripharls such as bluetooth mice and keyboards. This model maintains persistent, high-bandwidth connections between devices, which inevitably leads in high power consumption. Hence, Bluetooth Classic works best for applications where the need for power efficiency is not a primary concern.

Bluetooth Low Energy on the other hand, was introduced as part of the Bluetooth 4.0 specification to meet the growing demand for ultra-low-power and burst communication-especially relevant in battery-operated and mobile devices. BLE emphasizes quick connections, short data packets, and minimal power drain. It is optimized for scenarios such as transmitting sensor data, sending small status updates, or triggering remote functions—all without requiring a continuous data stream.

BLE achieves its efficiency through a fundamentally different communication model that includes advertising, scanning, and a connection phase that centers around a lightweight Attribute Protocol (ATT) and a structured Generic Attribute Profile (GATT). This model allows for simple, structured, and minimalistic communication between devices.

#### 2.5.1.2 Advertising

When a device operates BLE in advertising mode, it broadcasts short packets on three dedicated 2.4 GHz channels. These packets are called ”Advertisement Packets” and contains just enough information to serve as the basic “hello” that allows scanning devices to discover and identify the advertiser. This includes a 128-bit service UUID-an identifier used to uniquely identify a service that the advertiser serves, optional data fields and flags. Because these broadcasts are brief and infrequent-often just a few hundred microseconds every 100 ms or more, they serve as very power consumption friendly. An advertiser can operate either as not-connected when its sole purpose is to broadcast information—or as connected that allows a listening peer to establish a direct link for message exchange.

#### 2.5.1.3 Scanning

When a device operates in BLE scanning mode, it passively listens on the three dedicated 2.4 GHz advertising channels for advertisement packets (broadcast by nearby devices running in advertising mode as explained in section 2.5.1.2. This allows a scanning device to discover advertisers in its area. During scanning, the device does not exchange any data but simply monitors and identifies the advertising packets. However, upon receiving an advertisement packet, it can process the contained data such as service UUIDs, flags, and optional payloads, and may choose to initiate a connection based on its relevance. This scanning mode in BLE is highly configurable, allowing developers to specify scan intervals (meaning how often the device listens) and scan windows (how long it listens during each interval). Because scanning is designed to take place in irregular intervals and the radio remains off between scan windows, the process remains highly power efficient. This efficiency enables continuous or periodic background scanning without significant battery drain, making it well-suited for mobile applications.

#### 2.5.1.4 Power Consumption

Bluetooth Low Energy was designed for minimal power draw, making it an ideal choice for smartphone-based IoT tasks. Rather than keeping the radio turned on continuously, a BLE peripheral spends most of its time in a low-power sleep state and wakes only for brief advertising events—often just a few milliseconds every 100–500 ms. Likewise, a central device only opens its receiver for short scan windows at configurable intervals, rather than maintaining an always-on listening state. This duty-cycling approach means that, in practice, BLE operations add only a few percent to a phone's overall battery drain, far less than Wi-Fi, GPS, or Bluetooth Classic.

#### 2.5.1.5 ATTribute Protocol & General ATTribute Profile (ATT & GATT)

The core of Bluetooth Low Energy's structured communication model is governed by two elements: the Generic Attribute Profile (GATT) and the Attribute Protocol (ATT). These are the elements that define how the data should be organized, accessed, and exchanged when a connection is established between two peripherals running BLE
The Attribute Protocol (ATT) provides a simple and efficient method to store and retrieve data in the form of attributes. Each attribute is uniquely identified by a handle and may represent things such as a sensor reading, a configuration parameter, or a user-defined piece of data. ATT is lightweight and designed to minimize overhead as much as possible making it highly suitable for devices where power is quite a concern.

Built on top of ATT is the Generic Attribute Profile (GATT), which defines how the data are grouped into meaningful services and characteristics. A service is a logical grouping of related data (e.g., an attendance service), and a characteristic represents an individual data item within that service (e.g., the timestamp of attendance). GATT defines how clients (typically scanning devices) can discover available services on a server (typically advertising devices), and how they can read from, write to, or subscribe to changes in those characteristics.

#### 2.5.1.6 Connection Phase

When a scanning device identifies an advertising device of interest, it can initiate a connection, marking the start of a more sustained and interactive communication phase. This connection is point-to-point and allows for reliable two-way data transfer between the two devices.

In this connected state, the GATT client (usually the scanner) and the GATT server (usually the advertiser) establish a communication session where the client can query the server's available services and interact with its characteristics. This interaction is made possible through the Attribute Protocol (ATT), which manages all data as discrete attributes with associated permissions and types.

The connection phase enables operations such service discovery where The client queries the server to retrieve a list of services it offers, reading/writing Characteristics so the client write data to the GATT server. Behind the scenes, this exchange is handled over BLE's connection-oriented channels, with attention to maintaining low latency and minimal power usage. Parameters such as connection interval, supervision timeout, and MTU size determine the frequency and size of data packets, striking a balance between performance and energy efficiency.

### 2.5.2 Ionic Framework

#### 2.5.2.1 Ionic Framework vs. Native Development

There are mainly two design approaches that can be followed when developing a mobile application-Native or Cross-Platform. Native Development is the approach that uses

Native Components and Frameworks provided by the device's manufacturer such as Swift for iOS Development or Java/Kotlin for Android Development. However, a Cross-Platform framework abstracts away platform-specific details. The Ionic Framework falls into this category of Cross-Platform where it allows deveopers to create optimized mobile applications using standard web technologies i.e HTML, CSS and JavaScript.
Native Development by nature, requires developers to maintain separate codebases for Android and iOS since the tools used for Native Development are completely different since they are platform-specific. Ionic, on the other hand, enables a single codebase that compiles into Native Apps for both platforms where it displays the content in a native web layer(e.g., WebView for Android). This grants a great advantage in terms of development efficiency. Nonetheless, developing natively offers direct integration with device hardware and more control. Still, a cross-platform framework such as Ionic uses bridges to access native device features, achieving a balance between functionality and portability.

#### 2.5.2.2 Capacitor and Plugins

Ionic's ability to interact with native device components is achieved using Capacitor which is Ionic's official native runtime. Capacitor Plugins act as a bridge between the Web Layer of the application-written in TypeScript with the Native Layer-written in platform-specific programming languages like Java or Swift.
Therefore, these Capacitor Plugins serve as the main mechanism for accessing native functionality such as Bluetooth, Geolocation, Sensors etc.. which is then exposed in the Web Layer of Ionic. There are quite a few plugins ranging from Official, Developed by Community or even Custom-built for more specific requirements. Along with Plugins, Capacitor offers event listeners which serve a critical role in communication between the Ionic and Native Layers, where an important event, for instance a GATT Server receiving a write on its characteristics, the Native Layer notifies the Ionic Layer to send the data in the Web Layer to be processed and then rendered in the User Interface.

# Chapter 3

# Communication Layer

## 3.1 Introduction to the Communication Layer

In the CountaBLE Attendance System, the two devices—what is called the "Organizer" and the "Participant" —play complementary roles in a classic client–server exchange. The Organizer acts as the Server whereas the Participant acts as the client. This end-to-end communication is realized over Bluetooth Low Energy and is decomposed into five distinct layers of responsibility:

1. Ionic/Angular with Capacitor
2. Native Android BLE API
3. GATT & ATT
4. L2CAP & HCI
5. Physical Layer

When a message is exchanged, each layer contributes its part in turn. The Ionic Typescript code calls the Capacitor plugin through a bridge which -> invokes Native Android's BLE API's methods which then-> structures the write as an ATT Write Request under the GATT profile, -> hands it off to L2CAP and the HCI interface for transport -> and finally the Physical layer deliver the packet over the 2.4 GHz radio. Figure 3.1 illustrates how data and control flow through these layers to achieve an acknowledged message transaction.

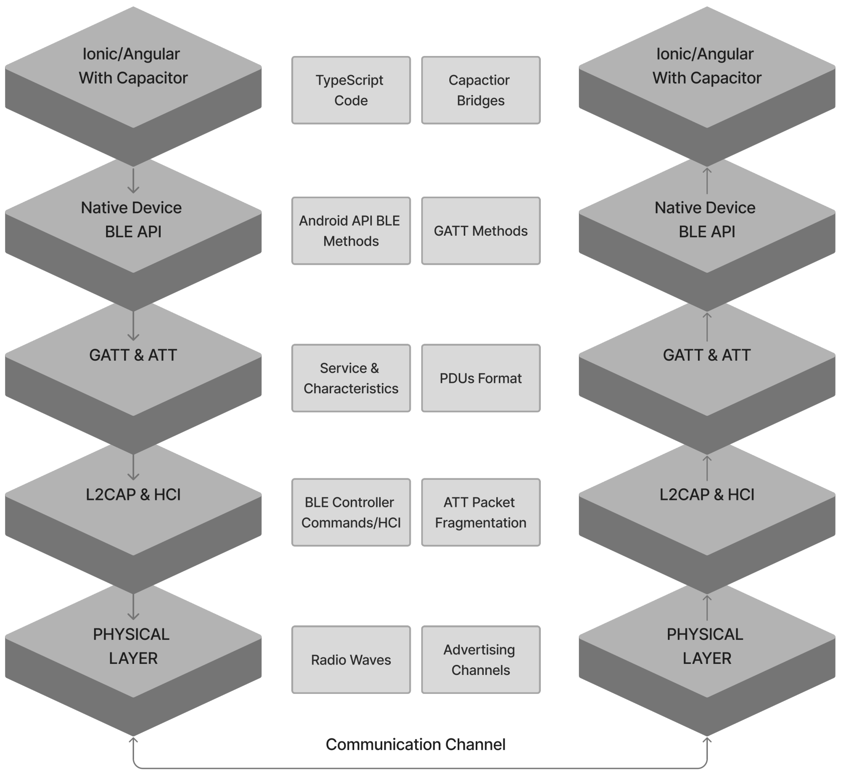


Figure 3.1: Communication Stack

## 3.2 Messages Exchanged Between Organizer and Participant

In the CountaBLE attendance system, three distinct kinds of BLE messages flow between the Organizer and the Participant:

### 3.2.1 Advertising Packet

As soon as the Organizer taps "Start Advertising," the Organizer's device begins broadcasting BLE advertising Protocol Data Units (PDU) packets. Each packet consists of a) The 128-bit Service UUID CountaBLE Attendance System Service and (b) include a small "service data" field. This field is used for identifying the advertising code of a group such as <epl400>—so that Participant Devices can display and identify human-readable information before connecting.

### 3.2.2 Client Payload (Write Request)

When a Participant selects an Organizer device and successfully connects, it writes a string to the Organizer's write-only characteristic. This string follows the format <username:deviceId> (e.g., johndoe:ABC123DEF456). Under the hood Android issues an **ATT Write Request** PDU, which encapsulates that byte sequence along with the handle of the characteristic.

### 3.2.3 Acknowledgment (Write Response)

The Participant (GATT Client) uses the default write type (WRITE_TYPE_DEFAULT). This write type is a **ATT Write-With-Response Message**, indicating that it requests and expects an acknowledgment from the Organizer (GATT Server). In response to the Write Request, the Organizer's GATT server emits an ATT Write Response PDU. This small packet carries no extra data beyond a status code :

1. *GATT_SUCCESS* –which indicates a successful communication
2. *ERROR <status_code>* –which indicates an error in communication with a status code

## 3.3 Communication Overview

When the Organizer enters "broadcast" mode, it becomes a BLE GATT Server. Behind the scenes, it configures the local radio to emit advertising packets on the three standard BLE advertising channels, carrying a 128-bit Service UUID that uniquely identifies the CountaBLE Attendance System Service. At the same time, it opens a GATT server interface, publishing a single write-only characteristic under that service UUID.
On the Participant side, the system invokes a BLE scanner and watches for any advertisement whose Service UUID matches the CountaBLE Attendance System Service UUID. If it matches, the system on the Participant side, initiates the GATT connection stack: negotiate link parameters, discovers services, and then hands control back so the app can locate the specific characteristic and then writes the payload i.e. < johndoe:ABC123DEF456>. Once the write completes, the Organizer's GATT Server receives that request, stores the attendance record, and sends back an ATT Write Response, which acts as an acknowledgment back to the Participant to confirm that the attendance is received. This entire exchange—from packet on the air to application callback—happens in a matter of tens of milliseconds.

## 3.4 Layers Breakdown

### 3.4.1 Ionic with Capacitor

The topmost layer of the stack is the Ionic/Angular front-end, augmented by Capacitor to bridge into native functionality. At this level the application is oblivious of Bluetooth Low Energy internals. It exists solely to render UI components, display information, respond to user input and communicate and transfer data to and from the native layer. For example, on the Organizer side, the "Start Advertising" button is implemented in an Angular component. when tapped the component's TypeScript method "startAdvertising()" is executed:

```
async startAdvertising() {
    try {
        await this.requestPermissions();
        await BleAdvertiser.startAdvertising({ className:
            this.classObj.advertcode });
        console.log("BLE Advertising started!");
    } catch (error) {
        console.error("Failed to start BLE advertising:", error);
    }
}
```

This method eventually calls BleAdvertiser.startAdvertising() which is essentially the bridge to call the Native Android startAdvertising() method in the BleAdvertiser Java Class, which starts BLE Advertising by sharing Advertising PDU's as explained in section 3.2.1 The Native Plugin's existence is revealed by exposing the Native's plugin public methods in an interface and then by Registering the Plugin with a specific name, e.g., BleAdvertiser.

```
// BLE Advertiser plugin (native)
export interface BleAdvertiserPlugin extends Plugin {
  startAdvertising(options: { className: string }): Promise<{ value:
    string }>;
  stopAdvertising(options: {}): Promise<{ value: string }>;
}

const BleAdvertiser = registerPlugin<BleAdvertiserPlugin>('
    BleAdvertiser');
```

The Ionic Layer with the Native Layer communicate using an event-listener pattern. On the Organizer side, as soon as the BLE GATT server processes an incoming attendance write, the plugin calls notifyListeners('onDataReceived', payload) to emit a JavaScript event containing the new data.

In the Ionic front-end a listeners is registered so that when the Native layer fires notifyListeners, the Ionic callback receives the attendance record and can format it and update the UI.

```
    this.dataListener = BleAdvertiser.addListener('onDataReceived', (
  data: any) => {
      this.ngZone.run(() => {
        // Pass the raw payload to our handler; it will split it.
        this.onDataReceived(data.data);
      });
    });
```

This approach cleanly decouples the low-level BLE logic from the display code, turning each native event into an asynchronous update in the Ionic/Angular component.

### 3.4.2 Native BLE API

In this layer reside all the Native Code required to handle BLE Operations in Android's terms. Here, the actual methods that were invoked by the Ionic capacitor bridge are executed using Android's Bluetooth LE framework.
The @CapacitorPlugin(name="") is used to register the Class as a Capacitor Plugin. This is what creates the "bridge" between the Ionic front-end Layer and the Native Code Layer

```
@CapacitorPlugin(name = "BleAdvertiser")
public class BleAdvertiserPlugin extends Plugin {
...
}
```

Then, functions that are going to be invoked by the Ionic Layer, are registered as a @PluginMethod. These are the same methods exposed in the Interface in the Ionic frontend, as explained in section 3.4.1

```
@PluginMethod
public void startAdvertising(PluginCall call) {
    ...
}
```

Methods such as startAdvertising(), startScan(), openGattServer(), connectGatt(), and writeCharacteristic() invoke the corresponding classes—BluetoothLeAdvertiser, BluetoothLeScanner, BluetoothGattServer, and BluetoothGatt—to carry out the actual radio operations, service publication, and attribute exchanges. All permission checks, callback registrations, MTU negotiations, and error-handling logic reside here, fully encapsulated. This is why the upper Ionic layer remains oblivious of the underlying Native BLE operations. As explained in section 3.4.1, when these native operations complete or generate events (for example, an incoming attendance write or a discovered peripheral), this layer uses Capacitor's notifyListeners(...) mechanism to emit events back into the Ionic front-end.

```
instance.notifyListeners("onDataReceived", data);
```

### 3.4.3 GATT & ATT

ATTribute Profile (ATT) defines the procedure for reading, writing, notifying, and handling every request or response, including the all-important Write Request / Write Response handshake that we use to guarantee that each attendance marker is reliably received. GATT builds on ATT by laying out how services, characteristics, and optional descriptors are organized: A service UUID tells the client 'this is the Attendance Service', and each characteristic UUID in it represents a single data channel for writing student credentials. The Organizer Native side, uses BluetoothGattService and BluetoothGattCharacteristic objects, register them with BluetoothGattServer.addService(...), and implement onCharacteristicWriteRequest(...) to capture each write. The Participant Native side invokes discoverServices(), fetches the exact service and characteristic by UUID, sets the charac-

teristic's write type to WRITE_TYPE_DEFAULT so that Android issues an ATT Write Request, meaning it expects an Acknowledgment.
The GATT server's subsequent sendResponse(..., GATT_SUCCESS, ...) call not only acknowledges the write at the ATT level but also triggers the client's onCharacteristicWrite(...) callback, giving a clear end-to-end confirmation of attendance mark success.

### 3.4.4 L2CAP & HCI

When using calls on Android's BLE API high-level methods—startScan(), startAdvertising(), connectGatt()—the system translates them into HCI commands and sends them to the Device's Bluetooth controller. For example, startAdvertising() maps to an HCI LE_Set_Advertising_Enable command; connectGatt() becomes an HCI LE_Create_Connection. Likewise, when the controller firmware detects an advertisement or establishes a link, it generates HCI events that Android's stack converts back into your callbacks: onScanResult() or onConnectionStateChange().
L2CAP lives above that link, multiplexing the single BLE connection into logical channels, fragmenting and reassembling ATT Protocol Data Units when messages exceed the negotiated MTU.
Although the application code never speaks L2CAP directly, the choice of MTU size (via requestMtu()) and reliance on write-with-response messages implicitly rely on L2CAP's ability to carry those larger packets end-to-end.

### 3.4.5 Physical Layer

In the Physical Layer the messages exchanged between the Organizer and the Participant become radio waves and vice versa, which of course, is completely invisible to the user. It leverages the device's 2.4 GHz transceiver to modulate and demodulate data across predefined advertising and data channels and by hopping frequencies to avoid interference. Although there is no interaction with it directly, this layer is the ultimate medium that carries every attendance-marking packet through the air.

# Chapter 4

# Data Layer

## 4.1 Overview

The Data Layer in the CountaBLE Attendance System ensures that the application can retrieve, and process data efficiently and consistently. It is responsible for storing both persistent data as well as small configuration parameters like as attendance records, class records, user preferences or selected application mode.
Data in the CountaBLE Attendance System are stored locally in a SQLite database - a very lightweight relational database that allows for structured and reliable data management offline. Therefore, the system has the ability to operate without Internet connection. The structure of the database is formed by relationships between core entities such as Classes, Attendances, and Participants, and is captured formally in the application's entity-relationship diagram and relational schema.

For managing lightweight user preferences and configurations, the system also uses a simple key-value system called Preference Storage. These are handled separately from the SQLite relational database. Such preferences are selected User-mode and Auto-Join preferences on specific classes when opearting as a Participant.
The system was designed to maintain a clean architecture, by abstracting the data layer entirely from the user interface logic. With this approach, the code that is responsible for managing the User Interface can interact with the data layer by utilizing a controlled set of public methods, without being exposed to the underlying storage mechanics.

## 4.2 SQLite Database

The SQLite local database acts as the core storage mechanism for the CountaBLE Attendance System, since data must be persistent - meaning they must ”saved” even when the application closes. It stores data for when operating in both Participant and Organizer modes.

### 4.2.1 Entity-Relationship Diagram

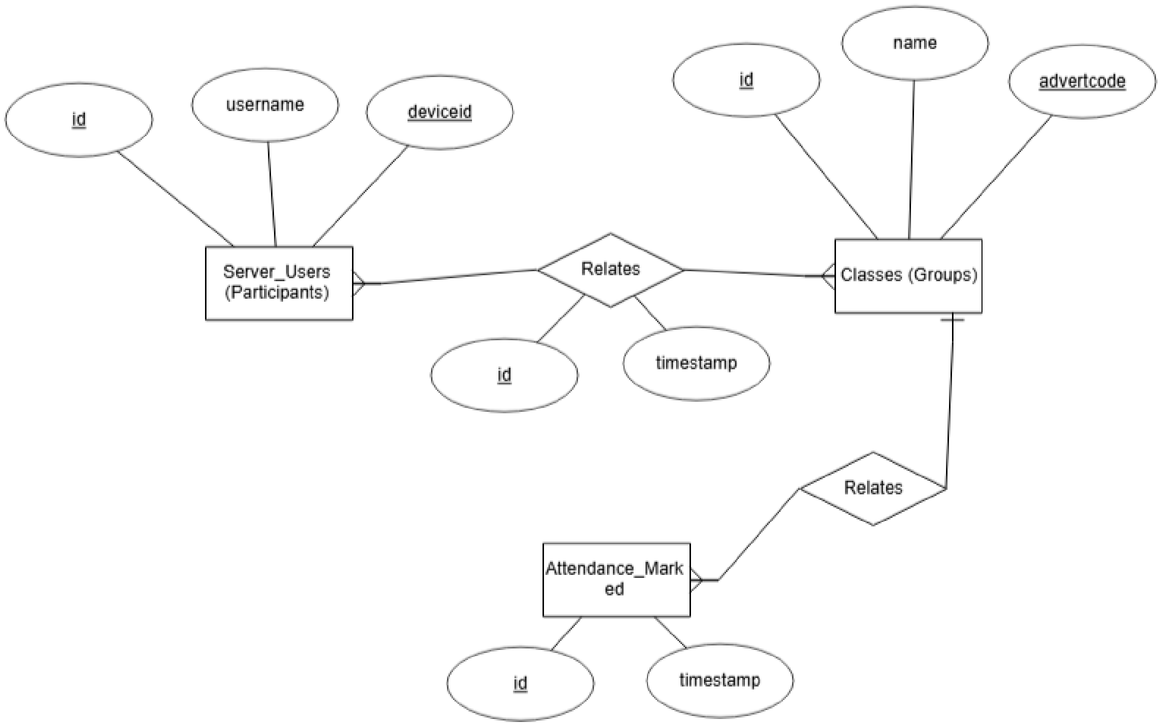


Figure 4.1: Entity-Relationship Diagram

### 4.2.2 Relational Schema

The relational schema of the CountaBLE Attendance System is designed to reflect the logical structure needed for an attendance tracking system. Four core tables are used which are represented visually in Figure 4.2, showing the relationships that exists between them-the foreign key constraints that enforce data integrity and consistency between attendance records with their associated classes and participants.

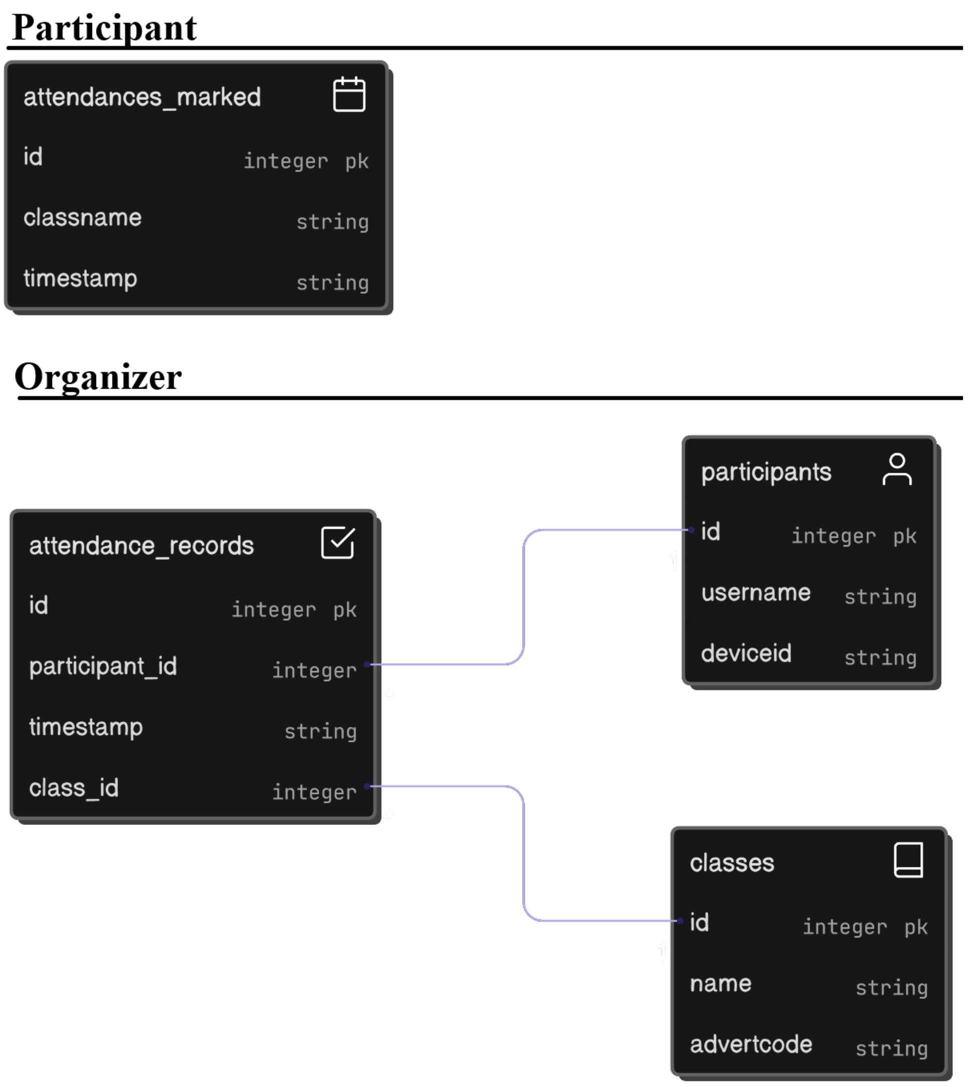


Figure 4.2: Relational Schema Diagram

The **classes** table defines each individual group that a user running in Organizer mode can create. It is uniquely identified by an auto-incremented ID and referenced by a user-defined name that the Organizer inputs upon creation. Each class is also associated with an advertisement code used during the Bluetooth communication between Organizers and Participants so that Participants can identify the desired group to connect to.

The **participants** table records hold information about each participant device, including a username and a device identifier. These entries represent unique users who can connect to the Organizer's device during a session.

The **attendance_records** table is a linking table that holds information about which participant has attended which class, along with a timestamp, recorded at the moment of attendance. This table holds foreign key constraints to both classes and participants tables.

Lastly, the **attendance_marked** table is distinct fully from the rest of the tables as it serves solely the Participant. It stores a local history of attendance timestamps by class name, used for displaying information in Participant's calendar view.

### 4.2.3 Queries

#### 4.2.3.1 Tables Creation

```
CREATE TABLE IF NOT EXISTS classes (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name UNIQUE TEXT NOT NULL UNIQUE,
            advertcode TEXT NOT NULL CHECK (LENGTH(advertcode) <= 6)
);


CREATE TABLE IF NOT EXISTS participants (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT NOT NULL CHECK (LENGTH(username) <= 14),
  deviceid TEXT NOT NULL CHECK (LENGTH(deviceid) <= 16)
);

CREATE TABLE IF NOT EXISTS attendance_records (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  participant_id INTEGER NOT NULL,
  class_id INTEGER NOT NULL,
  timestamp TEXT NOT NULL,
  FOREIGN KEY(participant_id) REFERENCES participants(id),
  FOREIGN KEY(class_id) REFERENCES classes(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS attendances_marked (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  classname TEXT NOT NULL,
  timestamp TEXT NOT NULL
);
```

#### 4.2.3.2 Creating a Group as an Organizer

When Creating a new Group as an Organizer, a TypeScript Method in the Ionic Front-end is called with the parameters

```
(className: string, advertcode: string)
```

```sql
INSERT OR IGNORE INTO classes (name, advertcode) VALUES (?, ?)
```

#### 4.2.3.3 Retrieving all Groups as Organizer

```sql
SELECT id, name, advertcode FROM classes
```

#### 4.2.3.4 Renaming a Group as Organizer

When renaming an existing Group as an Organizer, a TypeScript Method in the Ionic Front-end is called with the parameters

```
(oldName: string, newName: string, newAdvertcode: string)
```

```sql
UPDATE classes SET name = ?, advertcode = ? WHERE name = ?
```

#### 4.2.3.5 Marking Attendance as Organizer

When Marking an attendance, a TypeScript Method in the Ionic Front-end is called with the parameters

```
(className: string, username: string, deviceid: string, timestamp:
    string)
```

These are used to query the appropriate tables in order to mark the attendance of a Participant using the CapacitorSQLite Plugin.

```sql
--First finds the class id using the <className> paramater
SELECT id FROM classes WHERE name = ?
--Checks and finds if participant exists in the participants table
    using the <username,deviceid> parameters
SELECT id FROM participants WHERE username = ? AND deviceid = ?
--If participant doesn't exist (meaning its a new participant that hasn
    't connected before), then stores the record in the participant
    table using <username,deviceid> parameters. Then it selects that
    participant to retrieve the id
INSERT INTO participants (username, deviceid) VALUES (?, ?)
SELECT id FROM participants WHERE username = ? AND deviceid = ?
--Finally, inserts the attendance record in the attendance_records
    table using the retrived class and participants IDs and the <
    timestamp> parameter
INSERT INTO attendance_records (participant_id, class_id, timestamp)
    VALUES (?, ?, ?)
```

#### 4.2.3.6 Retrieving all Participants in a Group as Organizer

When Retrieving all Particiapnts in a Group, a TypeScript Method in the Ionic Front-end is called with the parameter

```
(className: string)
```

This is used to query and retrieve all the Participants that have previously attended a specific Group to display them in the Organizer's UI.

```sql
SELECT DISTINCT p.username, p.deviceid
FROM attendance_records ar
INNER JOIN participants p ON ar.participant_id = p.id
WHERE ar.class_id = ?
ORDER BY ar.timestamp DESC
```

#### 4.2.3.7 Inserting attendance in the Calendar as Participant

When an attendance is acknowledged by the Organizer, the Participant's side code, inserts a record in the *attendances_marked* table.

This is done by again, calling a TypeScript Method with the parameters

```
(classname: string, timestamp: string)
```

```
INSERT INTO attendances_marked (classname, timestamp) VALUES (?, ?)
```

#### 4.2.3.8 Retrieving unique attendance dates for the Calendar as Participant

When a Participant wishes to view their Calendar, all dates within the *attendance_marked* table are marked in the Participant's UI. This is done by retrieving all their attendance dates, and acknowledging the day of the retrieved records.

```
SELECT DISTINCT substr(timestamp, 1, 10) AS attendanceDate
FROM attendances_marked
ORDER BY attendanceDate DESC;
```

#### 4.2.3.9 Retrieving attendance records on a specific date for the Calendar as Participant

When a Participant selects a specific date (specifically, a day) on the Calendar, all the records that match the desired day are retrieved and shown in the Participants UI.

A TypeScript Method in the Ionic Front-end is called with the parameter

```
(date: string)
```

```
SELECT * FROM attendances_marked
WHERE timestamp LIKE ?
ORDER BY timestamp DESC;
```

### 4.2.4 Data Integrity

The CountaBLE Attendance System, maintains Data Integrity with the use of foreign key constrains as defined in the relational schema in Section 4.2.2. To maintain relational consistency, we use **ON DELETE CASCADE** clauses applied to the **class_id** foreign key in the **attendance_records** table to ensure that when a Group is deleted from the **classes** Table (i.e, a Group), all records for that specific Group in the **attendance_records** Table, are automatically deleted as well. This is done to ensure that no orphaned rows remain in the database.

## 4.3 Preferences

The Preferences Storage is used mainly for storing user preferences. In the case of the CountaBLE system, it holds the **Group ID** set by the user when logging in at the begining, the **Usermode** selected and the **Auto-Join Preferences for Groups** used in the Participant side code to Auto Join desired groups.

# Chapter 5

# User Interface & Ionic Layer

## 5.1 Navigation and Menu Drawer

In the application, Navigation and Routing is mainly managed through Ionic's routing system, which allows navigating between the pages based on the application's state and the user's actions. The app features a contextual side menu drawer as shown in Figures 5.1 and 5.2 where it offers different choices based on the selected user mode - Participant or Organizer.
Since the development of this project was done in Ionic powered by Angular, the use of the application component allows us to set global content such as the Header, User Mode Ribbon, and drawer side menu, which stay persistent across all pages. The drawer serves as a navigational aid for session-level actions like changing modes or logging out to re-enter

a different Group ID. It becomes available only when one of the two modes is selected in the Select Page (Section 5.1) and remains fixed when navigating between pages within that mode. The ”Change Mode” option in the drawer is the only way that the user can navigate back to the Select Mode Page - a restriction done intentionally using Navigation guards and by resetting the Navigation’s Root to set the desired page as the head of the Navigation Stack. This is done to prevent users from accidentally navigating backwards to ensure a predictable behavior when navigating backwards

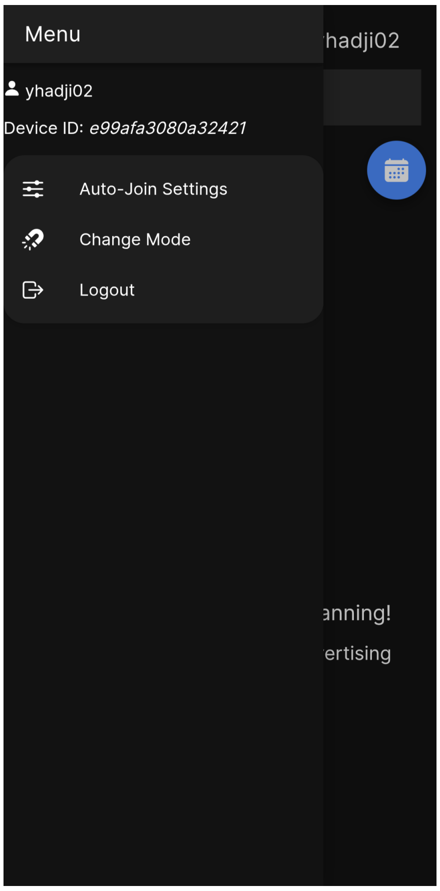


Figure 5.1: Drawer when in Participant Mode

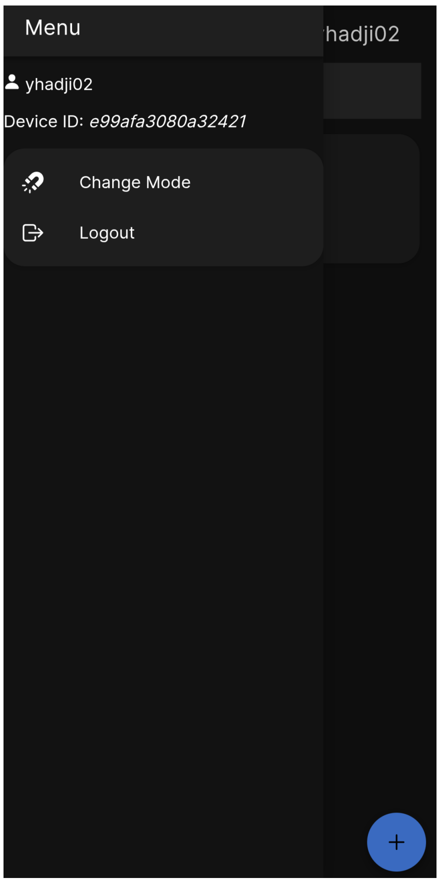


Figure 5.2: Drawer when in Organizer Mode

For these functionalities, Ionic provides different modules like *MenuController* for the Menu Drawer, *CanDeactivateGuards* that are fired on specific pages when trying to exit the page.

**Code Snippet of Menu Controller**

First the IonMenu component is imported from Ionic’s Angular Modules

```
import { IonMenu, ... } from '@ionic/angular/standalone';
```

which then allows the use of the IonMenu element in the application's component HTML File. By also importing Angular's CommonModule, we can use Ng Standard Directives in the HTML. This allows us to create contextual side menus by using NgIf directive which checks wether to display the parent HTML tag based on the condition provided.

```
      <ion-list>
        <ion-menu-toggle>
          <ion-item (click)="settingsAutoJoin()" *ngIf="this.usermode
    ==='participant'"}>
            <ion-icon name="options" slot="start" color="white"></ion-
    icon>
            Auto-Join Settings
          </ion-item>
        </ion-menu-toggle>
        ...
        ...
      </ion-list>
```

**Code Snippet of CanDeactivateGuard**

CanDeactivate Guards code fires when trying to exit a specific page. This is done by creating separate TypeScript files in the project and importing the CanDeactive Module from Angular's libraries

```
import { CanDeactivate } from '@angular/router';
```

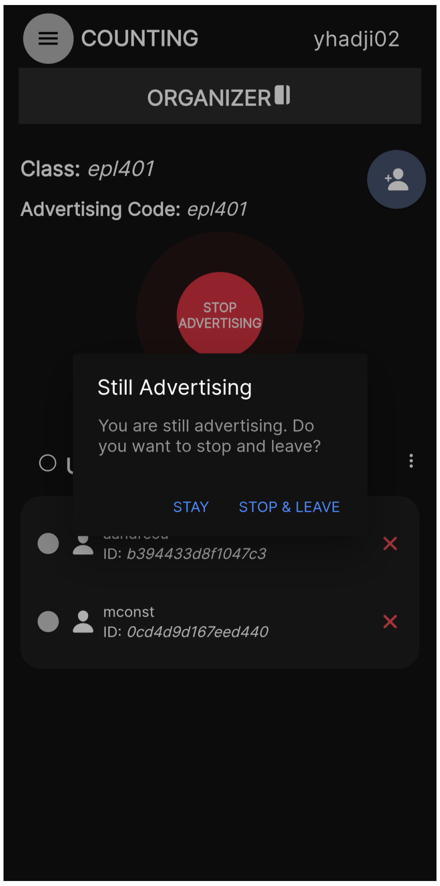
Then, the class must implement the CanDeactive interface which overrides the canDeactive function, where the logic of exiting a page takes place.

```
export class ExitAdvertismentGuard implements CanDeactivate<
    CanAdvertiseDeactivate> {
     ....
    async canDeactivate(component: CanAdvertiseDeactivate): Promise<
  boolean> {
        if (!component.isAdvertising) {
        return true;
    }
        // otherwise, prompt the user with an alert
        const alert = await this.alertCtrl.create({
            header: 'Still Advertising',
            message: 'You are still advertising. Do you want to stop
  and leave?',
            ....
        )}

    }
}
```

In Angular's application routes typescript file, we declare with the canDeactivate directive which Guard should take place when exiting the page

```
export const routes: Routes = [
    {
        path: 'advertise',
        loadComponent: () => import('./advertise/advertise.page').then(
    m => m.AdvertisePage),
        canDeactivate: [ExitAdvertismentGuard]
    },
  ...
]
```



Class: *epl401*

Advertising Code: *epl401*

Still Advertising

You are still advertising. Do you want to stop and leave?

ID: *b394433d8f1047c3*

mconst
ID: *0cd4d9d167eed440*

Figure 5.3: Organizer Prompt when trying to Navigate Back while Scanning

## 5.2 Login & Select Mode Pages

The **Login Page** provides a simple Alert Dialog prompting the user to enter their Group Identification name (such as johndoe@ucy.ac.cy) used when communicating with BLE. Even though a *CONTINUE WITH GOOGLE* button is shown in the Login page, this system focuses more on local device-based attendance, therefore no authentication mechanism such as email/password or OAuth were implemented. The user simply provides a Group ID and then is redirected to the Select Mode Page.

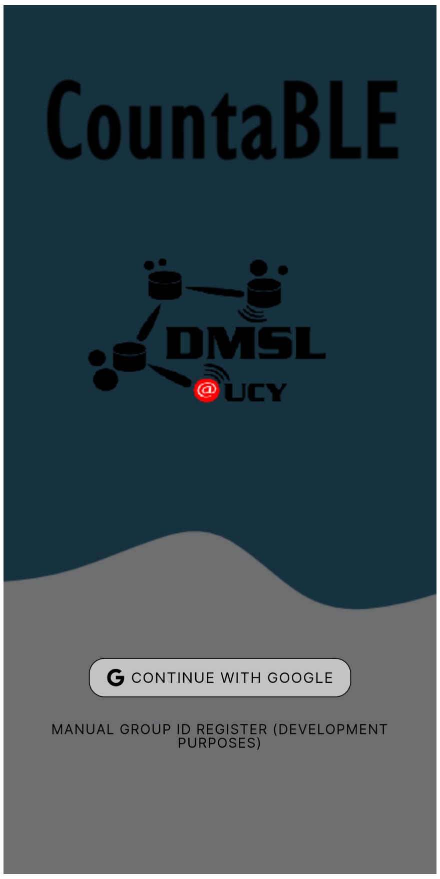


Figure 5.4: Login Page

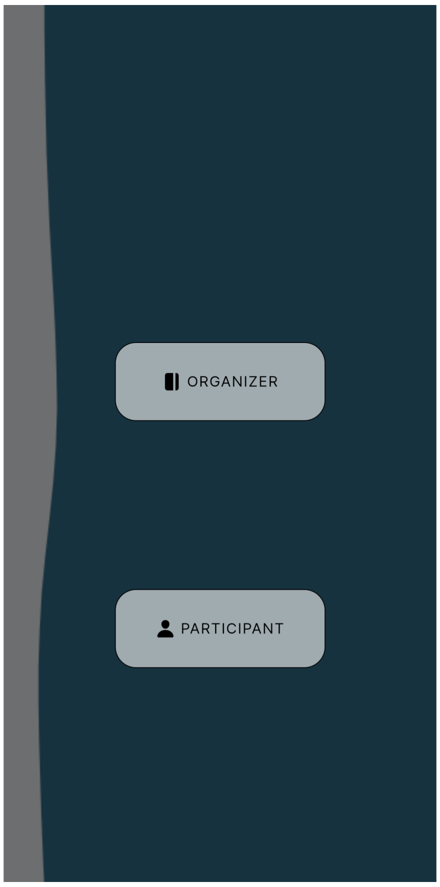


Figure 5.5: Select Mode Page

The **Select Mode Page** is where the user decides where they should enter in Organizer or Participant mode. This determines the behavior of the application for the rest of the session or until the user decides to Change Mode using the Menu Drawer. When Selecting one of the two modes, it saves the decision of the user in the Preferences Storage, and then a CanActivateGuard fires which then decides wether the app should navigate to the **Classes Page** when selecting the Organizer Mode, or to the **Scanning Page** when selecting the Participant Mode.

**Code Snippet of HTML & TypeScript Method when Clicking the *Login with Group ID* Button**

By including an attribute ”(click)= functionName()” in an HTML tag in Angular, we can declare which function to execute from the Component’s TypeScript file.

```
<ion-button class='manual' fill="clear" size="small" (click)="
    loginManualDev()">
  Manual Group ID Register
```

```
</ion-button>
```

```
async loginManualDev(){
    ...
}
```

**Code Snippet of HTML & TypeScript Method when Clicking the Mode Buttons**

The function selectMode() is used when either the Participant or the Organizer buttons are clicked

```
  <ion-button fill="clear" (click)="selectMode('organizer')">
    ...
  </ion-button>
  <ion-button fill="clear" (click)="selectMode('participant')">
    ...
  </ion-button>
```

And by then importing and using the Ionic's Navigation Controller we can set the root of the navigation stack according to the user's selection

```
import { NavController } from '@ionic/angular';

async selectMode(mode: 'organizer' | 'participant'){
   this.userService.setUsermode(mode);
   if (mode === 'organizer')
      this.navCtrl.navigate(["/classes"]);
   else
      this.navCtrl.navigate(["/connect"]);
}
```

## 5.3 Organizer Mode

As explained in previous sections, when **Organizer Mode** is selected from the **Select Mode Page** the application's context is tailored for organizing and managing attendance sessions. This means that the device takes the role of Bluetooth Low Energy Advertsier where the Organizer is responsible for creating attendance sessions for advertising to nearby Participant devices running in Participant Mode, and then handling the data received.
As the User is redirected to the **Classes Page**, all the Groups created by an Organizer previously are shown in a list. This is done by retrieving a list of all classes as shown previously in the Data Layer Chapter in Section 4.2.3.2. These list items represent distinct attendance groups or sessions such as courses for University Classes or Events. A user operating in Organizer Mode has the ability to create, rename, or delete a group. Each group holds a unique advertisement code identifier (i.e., <advertcode>) used in the advertisement packet as explained in the Communication Layer in Section 3.2.1.
When the user finally selects a specific group from the list, the application navigates to the **Advertisement Page**. This page starts the BLE advertising using custom Capacitor Plugins in the Native Layer (explained in the Native Layer Chapter 6) and listens for incoming data from participant using event listeners. Once a participant devices (running a GATT Client) successfully connects to the Organizer's device, this event listener is notified and receives the Participants payload (as explained in the Communication Layer Chapter in Section 3.2.2) and stores it in the SQLite database, as explained in Data Layer Chapter in Section 4.2.3.4.

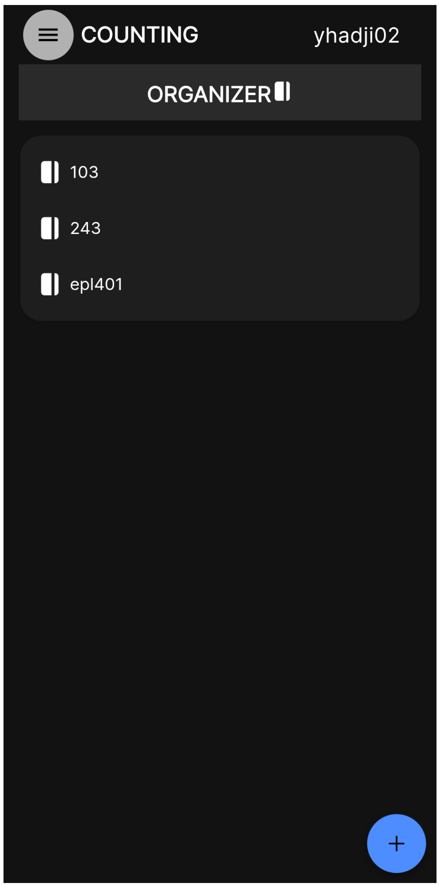


Figure 5.6: Groups List

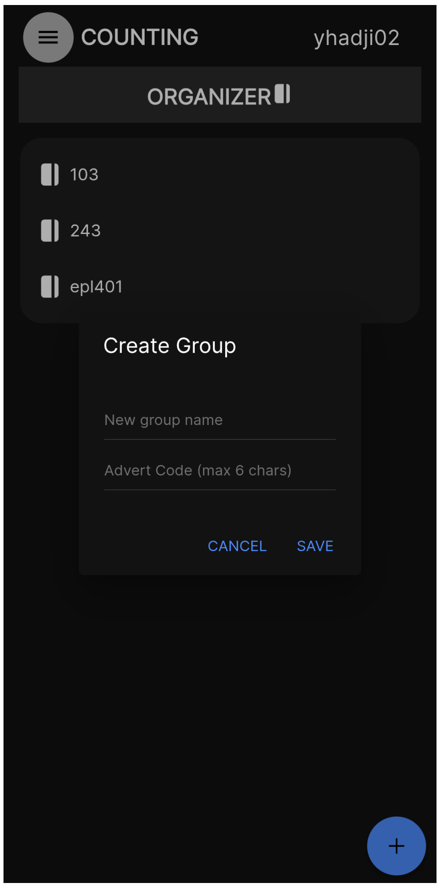
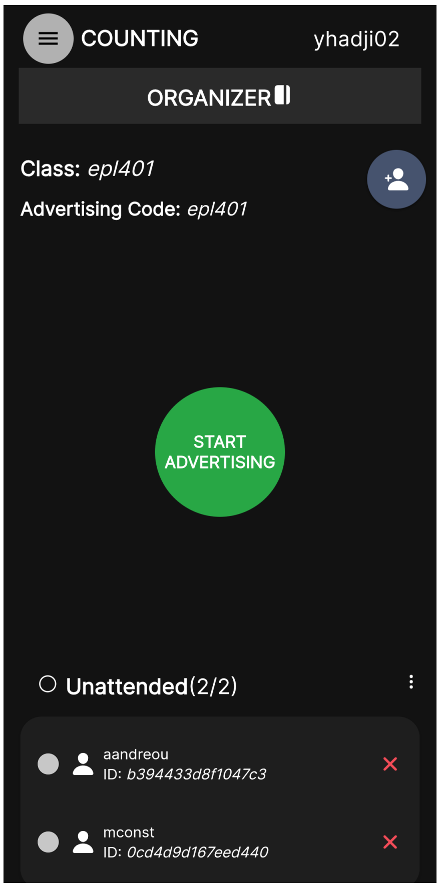


Figure 5.7: Creating a new Group



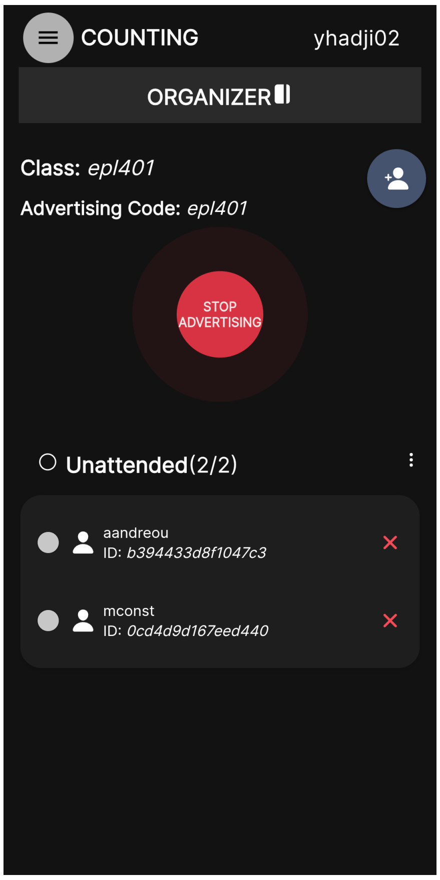
Class: *epl401*

Advertising Code: *epl401*

aandreou
ID: *b394433d8f1047c3*

mconst
ID: *0cd4d9d167eed440*

Figure 5.8: Advertising Page



Class: *epl401*

Advertising Code: *epl401*

aandreou
ID: *b394433d8f1047c3*

mconst
ID: *0cd4d9d167eed440*

Figure 5.9: Change of Advertise button to indicate that Organizer is Advertising

### 5.3.1 Class Page Code Snippets

All functionalities provided on the Class Page such as Viewing a list of all Groups, Creating a new Group, Renaming a Group etc., are operations that communicate with the application's SQLite Database. A custom SQLite Service (a TypeScript file) was created, and then each function in the Organizer's code calls the appropriate function from the SQLite Service.

**Retrieving and Displaying all Groups from the table**

```
//classes.pages.ts
async ionViewWillEnter() {
  try {.
    this.classes = await this.sqliteService.getAllClasses();
  } catch (error) {
    console.error("Error loading classes:", error);
  }
}
```

Each function in the the SQLite Service, runs the appropriate query (as shown in the Data Layer Chapter in Section 4.2.3), and then returns an array of objects based on the results form the SQLite Database.

```
//sqlite.service.ts
// Retrieve all classes - returning an array of objects { id, name,
   advertcode }
async getAllClasses(): Promise<{ id: number; name: string; advertcode:
   string }[]> {
  await this.ensureDBReady();
  try {
    const res = await this.db.query("SELECT id, name, advertcode FROM
   classes");
    return res.values || [];
  } catch (error) {
    console.error("Error fetching classes:", error);
    return [];
  }
}
```

### 5.3.2 Advertisement Page Code Snippets

Similarly to the **Class Page**, the functionalities of the Advertisement Page such as Loading Attendees List for a selected Group or Marking Attendances for Incoming Attendances from Participants, relies on communicating with the application's SQLite database. It follows exactly the same logic as explained in Section 5.3.1.

However, the core functionality of the Advertisement Page lies in communicating with the Native Layer to start the Advertising BLE Operation (where the Oeganizer acts as a GATT Server), and set up an event listener for incoming attendance packets from Participants, as explained in the Communication Layer Chapter in Section 3.4.1)

**Code Snippet of the function executed upon receiving an attendance packet from a Participant**

We set up an event listener upon receiving participant packets from the Native Layer (explained later in Chapter 6)

```
//Listen for BLE data received. The native side sends payloads in the
    form "username:deviceid"
this.dataListener = BleAdvertiser.addListener('onDataReceived', (data:
    any) => {
     this.ngZone.run(() => {
         this.onDataReceived(data.data);
      });
});
```

The data received from the Native Layer are automatically converted into JavaScript Objects which are then used to mark the Participant's attendance

```
async onDataReceived(payload: string) {
    ...
    await this.sqliteService.markAttendance(
        this.classObj.name,  // current class name
        receivedUsername,   // extracted username from the payload
        receivedDeviceId,  // extracted deviceid from the payload
        timestamp         // current timestamp
    );
    let attendanceRecord = { username: receivedUsername, deviceid:
   receivedDeviceId, timestamp:timestamp };
    this.removeAttendeesAddAttended(attendanceRecord); //Adds the
   Attended Participant in the this.attended list for displaying in
   the UI
}
```

**Code Snippet of the HTML displaying the contents of the received data list**

Using the CommonModule's NgFor directive, we can display all the Participant information received

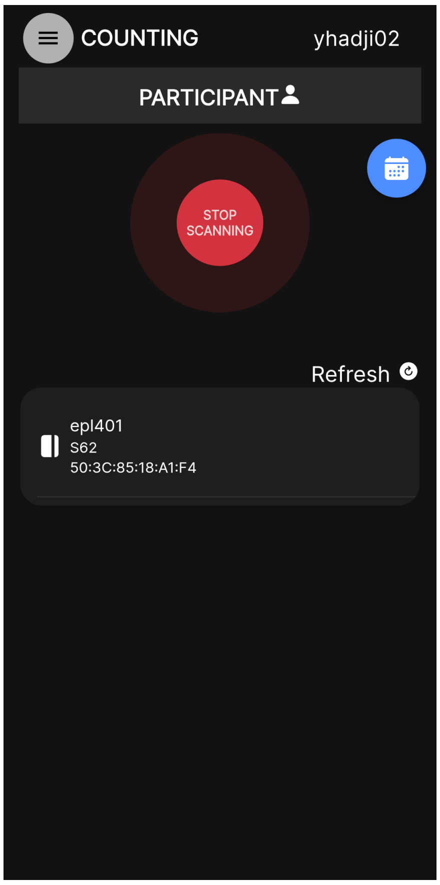
```
<ion-list lines="none">
    <ion-item *ngFor="let attendee of this.attended">
        <ion-label>
            <h3>{{ attendee.username }}</h3>
            <h4>ID: <i>{{ attendee.deviceid }}</i></h4>
        </ion-label>
    </ion-item>
</ion-list>
```

## 5.4 Participant Mode

Like Organizer Mode, when **Participant Mode** is selected from the **Select Mode Page** the application's context is tailored for joining attendance sessions hosted by a device running in Organizer Mode. This means that the device takes the role of Bluetooth Low Energy Scanner to identify valid attendance group/sessions and submitting attendance data

This page starts the BLE advertising using custom Capacitor Plugins in the Native Layer (explained in the Native Layer Chapter 6) where it filters and detects only advertisements packets holding a specific BLE Service UUID - the one that matches the CountaBLE Attendance Service UUID. This is pre-configured in the Native Layer. Once the scanning operation successfully detects an Organizer's device, it presents the data in a list in the Ionic Layer using event listeners. The data presented hold the Advertisement Code as

explained in Section 5.2 so that the Participant can spot the correct Group to join.
When finally selecting a Group session hosted by an Organizer, the Connection Phase begins where the Native Layer uses the BLE API to initiate a GATT connection, and upon success, transmits the payload in the form of <username:deviceId> to the Organizer's Device, as explained in the Communication Layer Chapter in Section 3.2.2. Upon successful communication, the Organizer's devices issues a Write Response which is basically an Acknowledgment that the attendance has been received, as explained in the Communication Layer in Section 3.2.3. Then and only then, the Participant's device issues an Alert Controller to the user, to prompt them wether they want this class to Auto-Join next time it is encountered while scanning, meaning the Participant doesn't need to interact with the UI to mark the attendance. Along with the prompt, the application records a local copy of the attendance in the SQLite Database of the Participant's logic as shown in the Data Layer Chapter in Section 4.2.3.7. This is then presented visually on a separate Attendance Calendar Page.
Overall, the Participant Mode is designed to require minimal manual input from the user by automating the scanning process, matching only with valid Participant's device running the app and by giving the option to Auto-Join Group sessions hosted by Organizers to achieve the goal of low-effort attendance registration.


COUNTING yhadji02

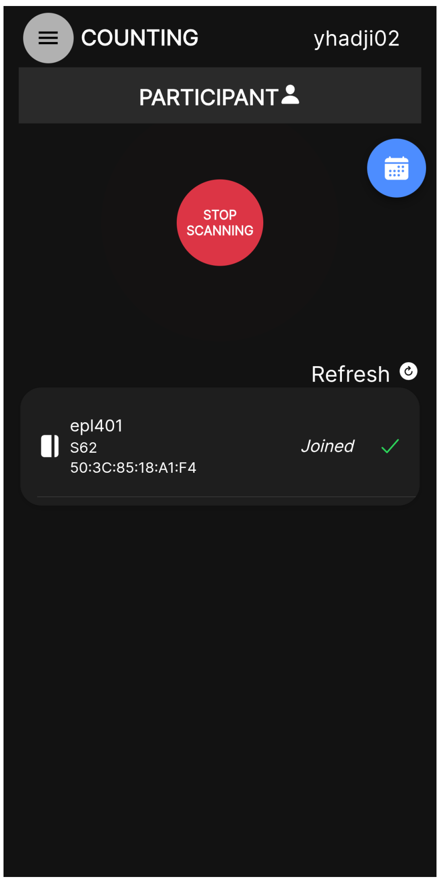
PARTICIPANT

STOP SCANNING

Refresh

epl401
S62
50:3C:85:18:A1:F4

Figure 5.10: Searching for Groups


COUNTING yhadji02

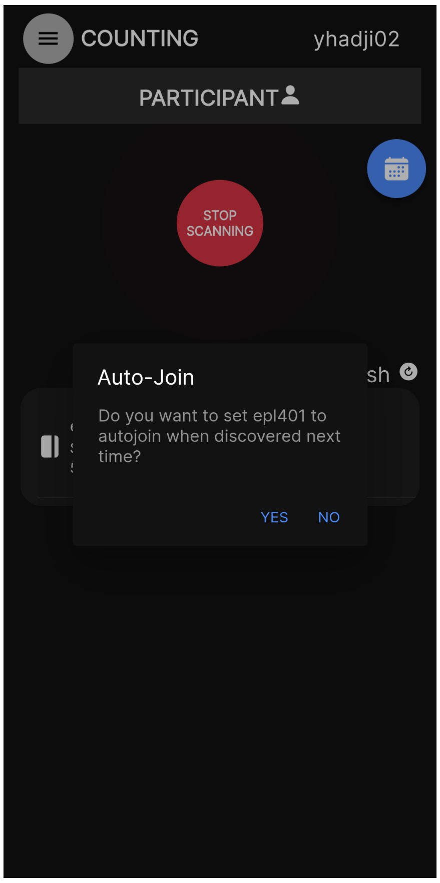
PARTICIPANT

STOP SCANNING

Refresh

epl401
S62
50:3C:85:18:A1:F4
*Joined* ✓

Figure 5.11: Group Found


COUNTING yhadji02

PARTICIPANT

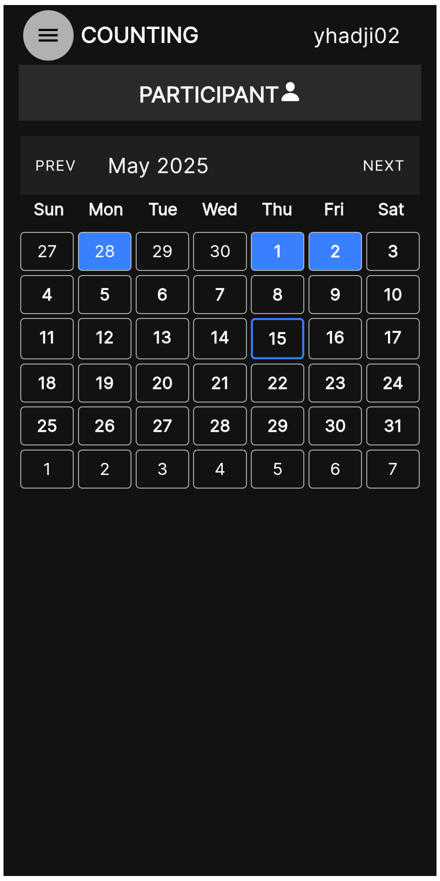
STOP SCANNING

Auto-Join

Do you want to set epl401 to autojoin when discovered next time?

YES NO

Figure 5.12: Prompt for Auto-Joining


COUNTING yhadji02

PARTICIPANT

PREV May 2025 NEXT

| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| 27 | 28 | 29 | 30 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Figure 5.13: Calendar Page

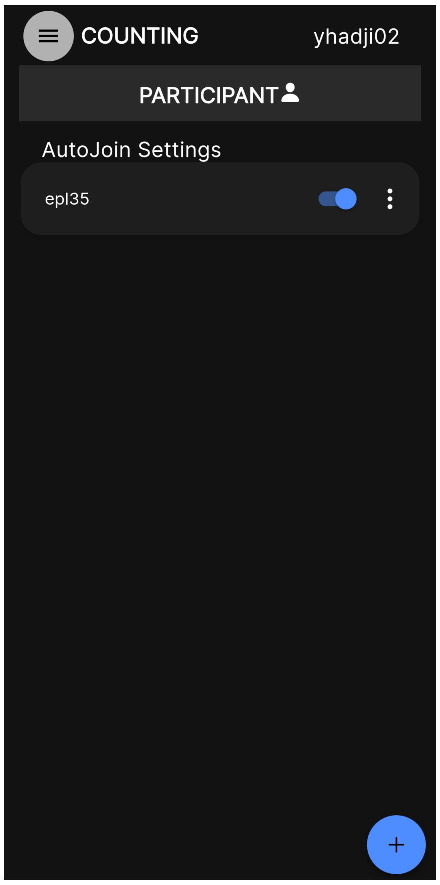


Figure 5.14: Auto Join Page

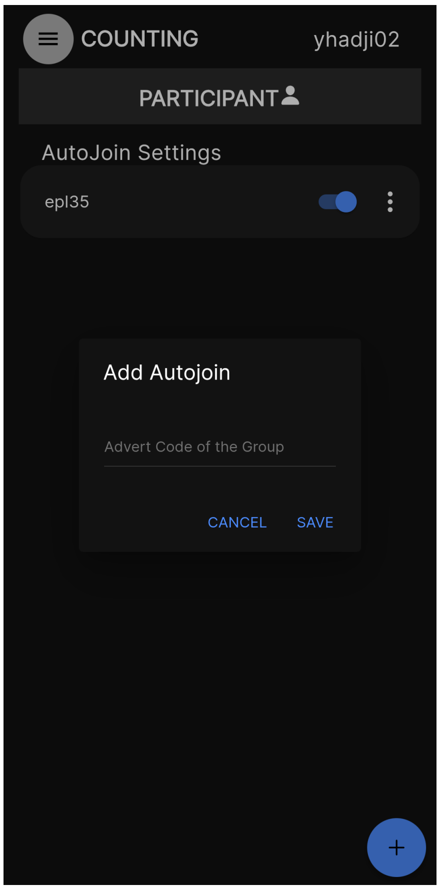


Figure 5.15: Add Manual Auto-Join Setting

### 5.4.1 Scanning Page Code Snippets

**Code Snippets for Event Listeners when Acknowledgment has been received from Organizer**

```
BleConnect.addListener('onAckReceived', (classname: {"advertcode":
    string}) => {
  this.ngZone.run(() => {
    const timestamp = new Date().toISOString();
    for (let device of this.discoveredDevices){
      if (device.classname == classname.advertcode )
        device.joined=true;
    }
    this.sqliteService.markAttendanceForParticipant(classname.
   advertcode, timestamp);
    this.presentToast("Connected to "+classname.advertcode);
  });
});
```

### 5.4.2 Auto-Join Page Code Snippets

#### Code Snippets for Retrieving, Displaying and Changing Auto-Join Settings

```
async loadSettings() {
  const allSettings = await this.autoJoinService.getAllAutoJoinSettings
    ();.
  this.ngZone.run(() => {
    this.settings = allSettings;
  });
}

async getAllAutoJoinSettings(): Promise<{ className: string; autojoin:
   boolean }[]> {
  const keysResult = await Preferences.keys();
  const keys: string[] = keysResult.keys;
  const settings: { className: string; autojoin: boolean }[] = [];
  for (const key of keys) {
    if (key.startsWith('autojoin_')) {
      const { value } = await Preferences.get({ key });
      // remove prefix to get raw class name
      const className = key.substring('autojoin_'.length);
      settings.push({
        className,
        autojoin: value === 'true'
      });
    }
  }
  return settings;
}

async onToggleChange(className: string, event: any) {
  const newValue: boolean = event.detail.checked;
  await Preferences.set({
    key: `autojoin_${className}`,
    value: newValue.toString()
  });
  //update local state
  this.settings = this.settings.map(s => {
    if (s.className === className) {
      return { ...s, autojoin: newValue };
    }
    return s;
  });
}
```

# Chapter 6

# Native Layer (implemented for Android)

## 6.1 Overview

The Native Layer implemented in the system is responsible for all the critical Bluetooth Low Energy (BLE) operations that are used in the system - Scanning And Advertising using the Generic Attribute Profile (GATT). These operations are written in Native Programming Language Code and are then exposed to the higher-level Ionic Layer through Capacitor Plugins, that otherwise wouldn't be possible from Ionic alone.
In this Layer, Capacitor, Ionic's native runtime, communicates with the Native Code using bridges. For Android, the implementation is done in Java Classes that extend the Capacitor Plugin base class and exposes methods that are annotated with the @PluginMethod declaretive, as explained in the Communication Layer Chapter in Section 3.4.2. This is what creates the bridge between the two layers.
The BLE operations for Advertising and Scanning - including the correct initialization and startup of the GATT Server and Client, is handled by the Android's BLE API, by using classes such as BluetoothLeAdvertiser, AdvertiseCallback, BluetoothLeScanner, Scan-

Callback etc. These APIs offer reliable control over advertisement settings (e.g., transmit power, advertising intervals), advertising payloads (e.g., service UUIDs, custom data), and scan filters.

## 6.2 Foreground Services

In order to minimize the user interaction with the application, the Native Layer written in Java for Android leverages the foreground services model. This allows Scanning and Advertising to run even when its not actively in use or when the device enters a low-power state like locking the screen. Implementing the BLE operations as a foreground service, allows us to execute them in the background since otherwise the Operating System might block background execution.
Foreground services are instantiated through a standard Android procedure and always needs to be accompanied by a visible notification.

## 6.3 Code Snippets

To start a foreground service, the Ionic front-end calls the relevant function from the Native Layer, annotated by the @PluginMethod declarative. For instance, for starting the Advertising Foreground service: **Starting and Stopping the Advertising Foreground Service**

```java
@CapacitorPlugin(name = "BleAdvertiser")
public class BleAdvertiserPlugin extends Plugin {
    ...
    // start the foreground advertising service
    @PluginMethod
    public void startAdvertising(PluginCall call) {
        String className = call.getString("className", "default");
        Intent serviceIntent = new Intent(getContext(),
    BleAdvertisingService.class);
        serviceIntent.putExtra("className", className);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            getContext().startForegroundService(serviceIntent);
        }
    call.resolve();
    }
}
```

This starts the service where the actual BLE API provided by Android is used. First, we configure the advertisement packet. **Using the Android's BLE's API for Advertising**

```
...
//Configure Advertisement Packet
AdvertiseSettings settings = new AdvertiseSettings.Builder()
        .setAdvertiseMode(AdvertiseSettings.ADVERTISE_MODE_LOW_LATENCY)
        .setTxPowerLevel(AdvertiseSettings.ADVERTISE_TX_POWER_HIGH)
        .setConnectable(true)
        .build();
AdvertiseData data = new AdvertiseData.Builder()
        .setIncludeDeviceName(true)
        .addServiceUuid(new ParcelUuid(SERVICE_UUID))
        // Include the className in the service data.
        .addServiceData(new ParcelUuid(SERVICE_UUID), className.
    getBytes(StandardCharsets.UTF_8))
        .build();
```

If Advertising started successfully, the GATT Server is configured and then started. An Abstract Class called BluetoothGattServerCallback is implemented. The onCharacteristicWriteRequest(onCharacteristicWriteRequest(BluetoothDevice device, byte[] value, booleanResponseNeeded, ...)) fires when a GATT Client send their data, meaning a Participant has submit their attendance. The responseNeeded boolean parameter specifies if this is a Write-With-Response request, meaning it expects and acknowledgment back from the GATT Server.

```
BluetoothGattService service = new BluetoothGattService(SERVICE_UUID,
    BluetoothGattService.SERVICE_TYPE_PRIMARY);
BluetoothGattCharacteristic characteristic = new
    BluetoothGattCharacteristic(
  CHARACTERISTIC_UUID,
  BluetoothGattCharacteristic.PROPERTY_WRITE,
  BluetoothGattCharacteristic.PERMISSION_WRITE
```

```
);
service.addCharacteristic(characteristic);
gattServer.addService(service);

gattServer = bluetoothManager.openGattServer(this, new
    BluetoothGattServerCallback() {
     onCharacteristicWriteRequest(BluetoothDevice device, byte[] value,
    booleanResponseNeeded, ...){
        if (responseNeeded) {
            //Send GATT_SUCCESS code which serves as Acknowledgment
            boolean ok = gattServer.sendResponse(device, requestId,
    BluetoothGatt.GATT_SUCCESS, 0, null);
        }
```

Finally, we construct the payload to be then sent back to the Ionic Layer by notifying the event listener.

```
        //Send the Received data back to the Ionic Frontend
        String receivedData = new String(value, StandardCharsets.UTF_8)
    ;
        JSObject response = new JSObject();
        response.put("device", device.getAddress());
        response.put("data", receivedData);
        BleAdvertiserPlugin.broadcastDeviceReceivedData(response);

     }
});
```

## 6.4 Threat Model

In a real-world scenario, different threat cases can occur, which could potentially cause confusion or disruption in the attendance tracking process. When designing the CountaBLE Attendance System, some of these threats were considered, therefore different features were implemented to ensure the system remains functional and reliable while also offering a smooth experience when using the system.

**A Participant doesn't own a Mobile Device**

In the case where a participant does not own a mobile device to run the mobile application, the system offers a fallback mechanism in the Organizer Mode. In the Group Page where the Organizer can view all the participants and their status, an "*Add Person*" button allows manual registration, prompting the Organizer for the participant's name and generates a random UUID to be used instead of an actual device identifier. This ensures that a participant who doesn't know a mobile device, is still included in the attendance records.

**Device Changed**

If a participant has changed their device, the unique identifier that is tied with them will also change. To address this, the participant can simply reconnect using their new device. This will result in creating a new entry on the organizer's device. The organizer then, can simply select the old entry from the participants list of that particular group and remove it using the 3 dotted menu context menu.

**Incorrect Group Connection**

In a real-world scenario, for instance in an institutional organization, many rooms are located close to each other. This means that if multiple organizers advertise at the same time, participants could possibly connect to the wrong attendance session. In this case, the organizer can identify the wrongfully submitted attendance and remove them directly from the user interface using the 3 dotted menu context menu.

**Forgotten Device Scenario**

If a participant forgets their smartphone on the day of attendance, the system provides a feature to manually set the particular participant as attended. The organizer can locate the participant in the group list and, via the 3 dotted context menu, manually mark them as present.

# Chapter 7

# Experimental Evaluation

In this chapter, a series of evaluations conducted are presented to assess the practical performance of the developed attendance system. To validate the system's performance, experiments were designed around three key aspects. First, the performance of the local data layer (meaning the SQLite Database), in executing the appropriate queries and returning the responses back to the user interface. Then, the energy consumption profile of continuous BLE operations that run uninterrupted in the background and lastly, the communication layer is evaluated in a real pilot with 4 smartphones, designed to reflect a real world attendance scenario.

## 7.1 Evaluating the Communication Layer

To evaluate the practical viability and responsiveness of the communication layer, a real-world pilot experiment was conducted with four smartphone devices. One device was operating in Organizer Mode while the other three in Participants Mode, allowing the experiment to focus on measuring the peer discovery, connection establishment, and data exchange between Organizer and Participants using BLE.

In this experiment, the Organizer device was configured to start advertising a group session called ”EPL400”, to simulate the beginning of an attendance tracking session.

Here, the time needed from the moment that the advertising begun, up until all three participant devices successfully joined the attendance session was measured.

```
if (this.isAdvertising) {
        this.startTime=performance.now();
        this.startBackgroundAdvertising();
}
async onDataReceived(payload: string) { //executed when receiving an
    attendance
     ...
     if (this.k==3) //k indicates number of attendances received
        this.endTime=performance.now();
        console.log("Total time to discover 3 devices: "+(this.endTime-
    this.startTime))
}
```

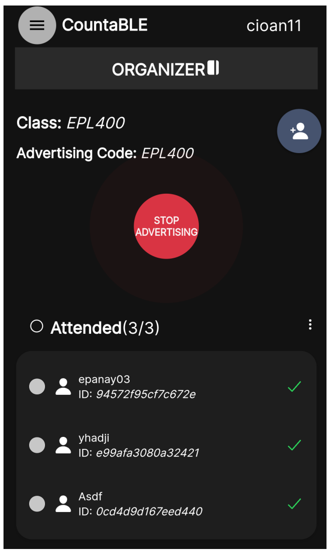
The Participant devices were running with the Auto-Join feature enabled, meaning it required no user interaction after initiating the scanning. Then, for each participant device, the time needed starting from when the Start Scan button was pressed up until an acknowledgment has been received from the Organizer, was also measured.

```
if (this.isScanning) {
        this.startTime=performance.now();
        this.startBackgroundScanning();
}
BleConnect.addListener('onAckReceived', (classname: {"advertcode":
    string}) => {
          .....
          this.endTime=performance.now();
          console.log("Total time for sending attendance message: "+(this
    .endTime-this.startTime));
}
```

The pilot experiment was successfully completed, validating the end-to-end communication between the organizer and all participant devices. Each device operating in Participant

Mode, was able to successfully detect the Organizer's device, establish a connection, transmit the attendance payload and finally receive an acknowledgment without requiring any manual intervention, other than initiating the scanning process. At the same time, the device operating in Organizer Mode, received all of the attendance records from Participant devices and updated its local user interface in real time.

Screenshots captured from all four devices, illustrate the correctness of this experiment, confirming that all devices running in the appropriate mode, function as intended. To further verify the system's robustness, the experiment was repeated using different smartphones as the organizer device, always achieving the same results.



Class: *EPL400*
Advertising Code: *EPL400*

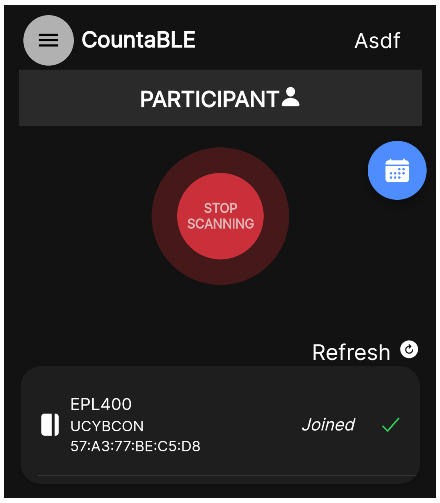
Attended(3/3)

epanay03
ID: *94572f95cf7c672e* ✓

yhadji
ID: *e99afa3080a32421* ✓

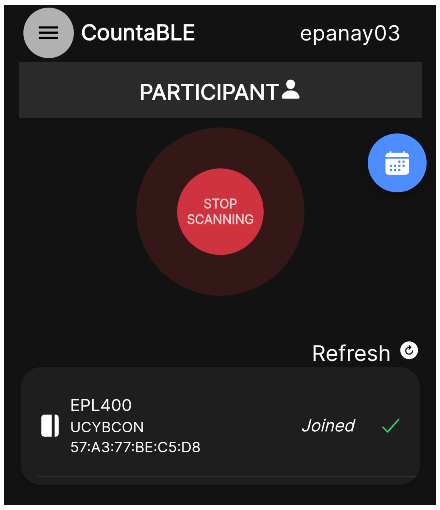
Asdf
ID: *0cd4d9d167eed440* ✓

Figure 7.1: Searching for Groups



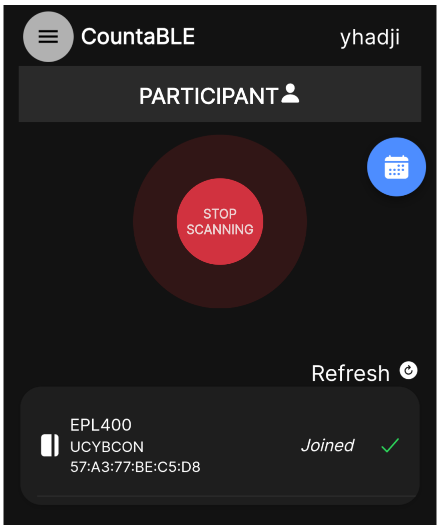
EPL400
UCYBCON
57:A3:77:BE:C5:D8
*Joined* ✓

Figure 7.2: Group Found



EPL400
UCYBCON
57:A3:77:BE:C5:D8
*Joined* ✓

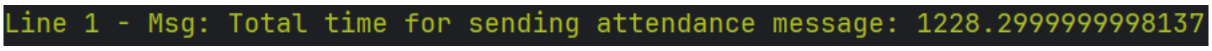
Figure 7.3: Prompt for Auto-Joining



EPL400
UCYBCON
57:A3:77:BE:C5:D8
*Joined* ✓

Figure 7.4: Calendar Page

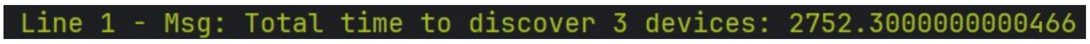
In terms of performance for the Participant, the whole process, starting from when the Start Scan button was clicked up to when the Acknowledgment was received back from the Organizer, was notably short. Across all tests, the average time per device was on average 1.60 seconds.



```
Line 1 - Msg: Total time for sending attendance message: 1228.2999999998137
```

Figure 7.5: Execution time for Participant

For the Organizer, the whole process, starting from when the Start Advertising button was clicked up to when attendance has been received from all three participants, was on average 3.36 seconds.



```
Line 1 - Msg: Total time to discover 3 devices: 2752.3000000000466
```

Figure 7.6: Execution time for Organizer

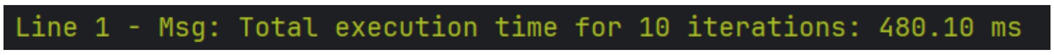
These results show that the system can effectively support simultaneous discovery and data exchange in realistic conditions, ensuring the viability of the communication layer.

## 7.2 Evaluating the Data Layer

To evaluate the system's local data layer performance, simulations were run on a physical Google Pixel 6 device operating in Organizer Mode. Specifically, a script was used, which inserts participants in a specific group, starting from 10 participants up to 150, incrementing the amount of participants by 10 each time, to assess the time needed for the whole operation to be completed.

```
async onTestSelect(value: string) {
    const iterations = parseInt(value, 10);
    let totalExecutionTime = 0;
    for (let i = 0; i < iterations; i++) {
      const start = performance.now();
      await this.onDataReceived(testuser${i}:testdevice${i});
      const end = performance.now();
      const iterationTime = end - start;
      totalExecutionTime += iterationTime;
    }
    console.log(Total execution time for ${iterations} iterations: ${
   totalExecutionTime.toFixed(2)} ms);
}
```

When running this simulation test, the physical Google Pixel device was connected with a Windows machine running the native android project in Android Studio, allowing the view of the ”Logcat” that displays any console logs from the ionic front end.



```
Line 1 - Msg: Total execution time for 10 iterations: 480.10 ms
```

Figure 7.7: Screenshot of the Logcat displaying the console.log command

The resulting execution times that were measured in milliseconds, were plotted against the number of participants inserted into the database in each test.

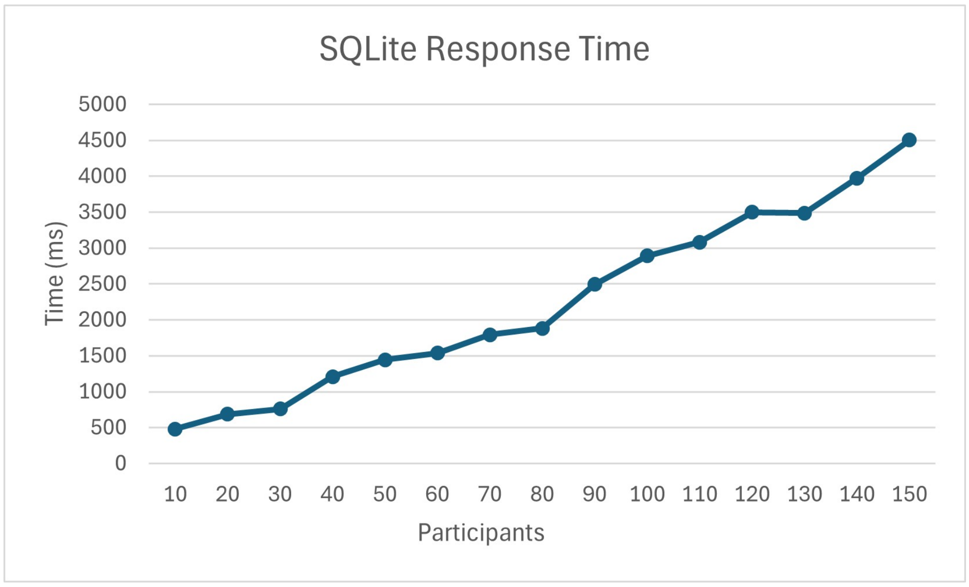
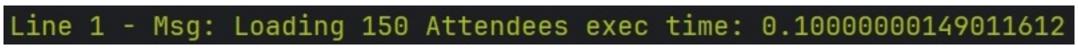


Figure 7.8: Plot for SQLite Database response time based on the amount of participants

As shown in the figure above, the insertion time increases gradually as the participant count increases from 10 to 150. Even though SQLite is optimized for embedded applications, it works as a single-threaded architecture, where it stores data on the device's internal storage making its performance sensitive to the quantity of the write operations. Therefore this is an expected behavior.

The highest recorded time sits at around 4,5 seconds for 150 participant insertions which still falls within an acceptable threshold for our system since these operations are infrequent and usually occur during setup rather than in real-time interaction.

After inserting the 150 participants, the execution time of loading all the participants was also measured in a similar manner, by measuring the time needed to execute the load participants query. The execution time sits at around tenths of a milliseconds, meaning 0.0001 seconds, proving SQLite's efficiency.

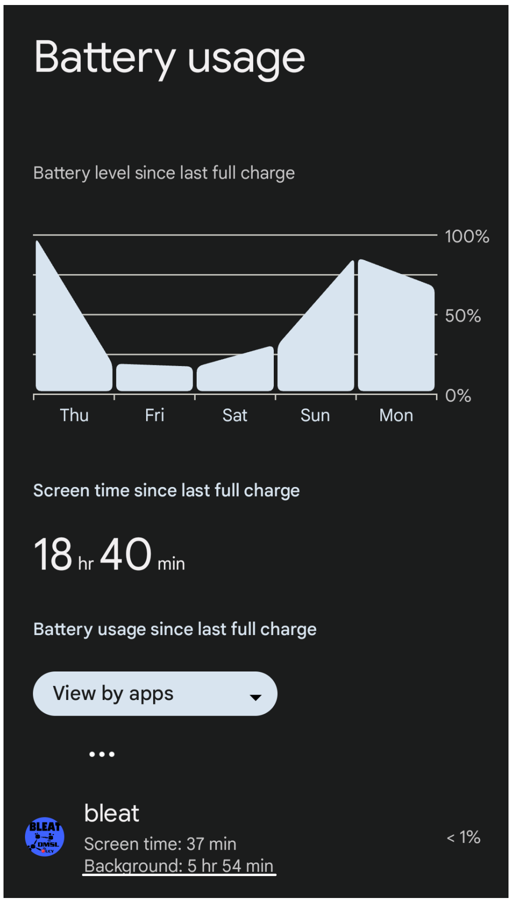


```
Line 1 - Msg: Loading 150 Attendees exec time: 0.10000000149011612
```

Figure 7.9: Caption

## 7.3 Evaluating Battery Consumption

To confirm the low-energy expectations of the BLE operations, scanning and advertising in the background continuously, a long-duration power consumption test was performed on a Google Pixel Android device. The device was configured to maintain the scanning process for 6 hours uninterrupted.


Battery usage

Battery level since last full charge

Screen time since last full charge

18 hr 40 min

Battery usage since last full charge

bleat

Screen time: 37 min

Background: 5 hr 54 min

< 1%

Figure 7.10: Battery Usage Profile

The screenshot captured above, illustrates the battery usage profile of the Google Pixel device, where it was actively scanning, running in the background as a foreground service. During this test, the device's screen activity was minimized and no additional apps were running, to isolate the energy impact of the BLE scanning process.

The battery usage profile shows that the CountaBLE application, consumed less than %1 battery over 6 hours. This result, validates and highlights the efficiency of the BLE technology, confirming that it is suitable for prolonged background operations without affecting the overall battery life of the device.

# Chapter 8

# Conclusions and Future Work

## 8.1 Conclusions

In this thesis, the CountaBLE Attendance System was proposed which utilizes BLE technology to tackle the various limitations exposed by other attendance methods - both manual or technology-assisted. It offer a power-efficient attendance tracking solution that requires no infrastructure or hardware, other than a modern smartphone, also without relying on internet connection. The system leverages each participant's personal device as the BLE medium used for exchanging attendance messages, and, along with the use of a unique per-app identifier that is generated for each user, it ensures that recurring attendance records can be reliably associated with the correct device without storing sensitive information. With the use of the Generic Attribute Profile (GATT), the Organizer of an attendance session acts as a GATT Server, broadcasting attendance information under a specific service UUID and characteristic, where Participants who act as GATT Clients, discover, establish a communication, and exchange data with the Server by writing on its characteristics. With the use of Write-With-Response writes in the GATT communication, a Participant receives an acknowledgment from the Organizer, indicating that the attendance was successfully received, ensuring a reliable exchange of attendance messages.

The system was implemented as a cross-platform mobile application to prioritize development efficiency but also to enhance user experience by maintaining consistency across Android and iOS platforms. This was possible through the use of Ionic Framework which uses standard web technologies, combined with custom native plugins for BLE operations. In this way, the system achieves a balance between portability and native performance. Furthermore, to achieve the goal of requiring minimal user effort to record attendance, the developed system uses the foreground service model of Android, allowing BLE operations to persist in the background in a power-efficient way.

The experiments conducted to evaluate the system, proved that the proposed architecture is reliable across all core layers. The evaluation of the data layer showed that even when the attendance session consists of a high number of participants, the embedded SQLite database handles data operations at an acceptable time frame, confirming its suitability for real-world scenarios. For energy consumption, the experiment where a Google Pixel device was configured to operate a prolonged scanning BLE operation uninterrupted for six hours, resulted in draining less than 1% of the device's battery, proving its suitability for prolonged background usage.

Lastly, the communication layer was evaluated through a real-word scenario involving one organizer and three participant devices. Each device operating in Participant mode was able to successfully detect and exchange attendance messages with the Organizer without any manual intervention. The experiment was repeated using a different device as the organizer device, where the communication still works as expected. The average time needed for an Organizer device to receive attendance from all three participant devices proved to be on average 3.66 seconds, while for a Participant, it took on average 1.6 seconds to discover, connect, exchange data, and finally receive an acknowledgment from the Organizer.

Altogether, the system proposed in this thesis, proves to be a practical attendance system that is infrastructure-free and energy-efficient, designed with cross-platform compatibility in mind.

## 8.2 Future Work

To further improve the proposed system, various possible enhancements can be done to improve the system's capabilities, performance, robustness and reliability. First, the Organizer Mode can be improved by providing a reliable way to export attendance session data, allowing integration with a Learning Management System (LMS) like Moodle. This would provide an easy way for Organizers working in a institutional environment to keep track of each attendance session in a the main centralized LMS that the institution uses, further increasing the automation of the CountaBLE system.
In addition, since the developed system was implemented in a cross-platform manner, currently it supports full functionality for Android devices. However, iOS compatibility can be supported by simply developing native functionality using the Apple Core Bluetooth BLE API.
Another direction for future development is the integration of BLE 5.0, which introduces significant improvements in the advertising and broadcasting capabilities. BLE 5.0 supports larger advertisement packets where the payload is increased from 31 bytes up to 255 bytes. With this improvement, the initial advertising message exchanged by Organizers could be enriched to include more data used for uniquely identifying attendance sessions.

# Bibliography

[1] M. Liu and Y. Yao, "Design of Intelligent Attendance System Based on RF Technology," in *Proc. 2nd Int. Conf. on Artificial Intelligence and Information Systems (ICAIIS 2021)*, New York, NY, USA: ACM, 2021, Art. no. 254, pp. 1–7. [Online]. Available: https://doi.org/10.1145/3469213.3470687

[2] A. D. D. Bayani, M. G. Fabra, V. M. C. Joseph, C. J. A. Pelayo, M. A. Diloy, and E. M. Esberto, "WeConnect: Class Attendance Monitoring System with the use of Mobile Tethering," in *Proc. 2023 8th Int. Conf. on Information and Education Innovations (ICIEI '23)*, New York, NY, USA: ACM, 2023, pp. 126–129. [Online]. Available: https://doi.org/10.1145/3594441.3594462

[3] Z. Zhang, J. Du, W. Diao, and J. Wu, "MiniBLE: Exploring Insecure BLE API Usages in Mini-Programs," in *Proc. ACM Workshop on Secure and Trustworthy Superapps (SaTS '24)*, New York, NY, USA: ACM, 2024, pp. 18–22. [Online]. Available: https://doi.org/10.1145/3689941.3695774

[4] S. M. Khan, M. T. Maliha, M. S. Haque, and A. Rahman, "WiFi Received Signal Strength (RSS) Based Automated Attendance System for Educational Institutions," in *Proc. 11th Int. Conf. on Networking, Systems, and Security (NSysS '24)*, New York, NY, USA: ACM, 2025, pp. 172–180. [Online]. Available: https://doi.org/10.1145/3704522.3704523

[5] A. Barua, M. A. A. Alamin, M. Hossain, and E. Hossain, "Security and Privacy Threats for Bluetooth Low Energy in IoT and Wearable Devices: A Comprehensive Survey," *IEEE Open J. Commun. Soc.*, vol. 3, pp. 1–1, 2022. doi: 10.1109/OJCOMS.2022.3149732.

[6] J. Stevenson, “Android Unique Identifiers,” in *Pro Android Privacy*, 1st ed., Berkeley, CA, USA: Apress, 2021, pp. 85-89. doi: 10.1007/978-1-4842-6914-5_7.

[7] S. Farooq, S. Riaz, A. Alvi, A. Ali, and I. Rehman, “Cross-Platform Mobile Development Approaches and Frameworks,” *VFAST Trans. Softw. Eng.*, vol. 10, no. 2, pp. 79–93, 2022. doi: 10.21015/vtse.v10i2.978.

[8] D. H. Morais, “Bluetooth LE Overview,” in *5G NR, Wi-Fi 6, and Bluetooth LE 5*, Cham, Switzerland: Springer, 2023. doi: 10.1007/978-3-031-33812-0_10.

[9] Bluetooth SIG, “Logical Link Control and Adaptation Protocol Specification,” Bluetooth, [Online]. Available: https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/host/logical-link-control-and-adaptation-protocol-specification.html

[10] Sphero Team, “Bluetooth Low Energy vs. Bluetooth: What’s the Difference?,” Sphero, Nov. 7, 2022. [Online]. Available: https://sphero.com/blogs/news/bluetooth-low-energy-vs-bluetooth

[11] Texas Instruments, “Generic Attribute Profile (GATT),” SimpleLink CC2640R2 SDK Documentation, [Online]. Available: https://software-dl.ti.com/lprf/simplelink_cc2640r2_sdk/1.35.00.33/exports/docs/ble5stack/ble_user_guide/html/ble-stack/gatt.html

[12] Adafruit, “Introduction to Bluetooth Low Energy: GATT,” [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

[13] Argenox, “BLE Advertising Primer,” [Online]. Available: https://argenox.com/library/bluetooth-low-energy/ble-advertising-primer#:~:text=BLE%20Advertising%20Packets&text=The%20Packet%20data%20unit%20for,of%20the%20Advertising%20Channel%20PDU

[14] M. Cunche, A. Boutet, C. Castelluccia, C. Lauradoux, and V. Roca, “On using Bluetooth-Low-Energy for contact tracing,” Inria Grenoble Rhône-Alpes and INSA de Lyon, Research Report, 2020. [Online]. Available: https://hal.science/hal-02878346v5

[15] U. Çabuk, G. Kanakis, and F. Dalkılıç, "LTE Direct as a Device-to-Device Network Technology: Use Cases and Security," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 5, p. 401, 2016. doi: 10.17148/IJARCCE.2016.5779.

[16] Qualcomm Technologies Inc., "Expanding Your Horizons with LTE Direct," Qualcomm Incorporated, White Paper, 2014. [Online]. Available: https://www.qualcomm.com/media/documents/files/srg-whitepaper-expanding-your-horizons-with-lte-direct.pdf

[17] MOKOSmart, "LTE Beacons vs. Bluetooth Beacons: What's the Difference?," MOKOSmart, [Online]. Available: https://www.mokosmart.com/lte-beacons-vs-bluetooth-beacons/

[18] C. Laoudias, M. Raspopoulos, S. Christoforou, and A. Kamilaris, "Privacy-Preserving Presence Tracing for Pandemics Via Machine-to-Machine Exposure Notifications," in *Proc. 23rd IEEE Int. Conf. Mobile Data Management (MDM)*, Paphos, Cyprus, 2022, pp. 355–360. doi: 10.1109/MDM55031.2022.00080.

[19] 'Android Open Source - accessory-samples Peripheral Activity' Java2s, [Online]. Available: http://www.java2s.com/Open-Source/Android_Free_Code/Example/code/com_example_android_bluetoothgattperipheralPeripheralActivity_java.htm

[20] Android Developers, "Bluetooth," Android Developer Documentation, [Online]. Available: https://developer.android.com/develop/connectivity/bluetooth/

[21] Ionic Framework, "Ionic Documentation," [Online]. Available: https://ionicframework.com/docs

[22] Capacitor, "Capacitor Documentation," [Online]. Available: https://capacitorjs.com/docs

[23] Angular Team, "Angular Overview," [Online]. Available: https://angular.dev/overview

# Appendix