International Journal of Computer Science and Mathematical Theory (IJCSMT) E-ISSN 2545-5699
P-ISSN 2695-1924 Vol 11. No. 5 2025 www.iiardjournals.org online version

Software Development Process for an IOT-Based Fingerprint Device Based on ESP32 MCU.

[*]Gabriel Ebiowei Moses and [**]Youdiowei Oyineteimode
[*]Department of Electronics/Electrical Engineering, Niger Delta University, Wilberforce Island Bayelsa State, Nigeria.
[**]Department of Mechanical Engineering, Niger Delta University, Nigeria
gabrielebiowei@ndu.edu.ng +(234)7987303430
DOI: 10.56201/ijcsmt.vol.11.no5.2025.pg46.62

Abstract
The rapid advancement of the Internet of Things (IoT) has led to the development of innovative applications in various domains, with biometric authentication being a significant area of focus. Among these, IoT fingerprint devices offer a promising solution for secure access control, identity verification, and time-tracking systems. The software development process for these devices is a multidisciplinary endeavor that integrates embedded system programming, biometric signal processing, and IoT communication protocols. This paper explores the key stages involved in the software development lifecycle of IoT fingerprint devices, from requirements gathering and system design to implementation and testing. Emphasis is placed on the challenges associated with ensuring device security, data privacy, and real-time performance while integrating fingerprint recognition algorithms, communication protocols, and cloud connectivity. Additionally, the paper discusses best practices for developing robust, scalable, and energy-efficient solutions that can operate effectively within the diverse and evolving IoT ecosystem. The outcome is a comprehensive understanding of the complexities of developing IoT fingerprint devices, offering valuable insights for engineers and developers in this cutting-edge field.

Keywords: *IoT, Fingerprint Device, Attendance, Time, Student, Lecturer*

INTRODUCTION

The Internet of Things (IoT) has revolutionized how devices interact with each other and the world, offering unprecedented automation, data collection, and connectivity (Zeng, D. et al., 2018). One such device that leverages IoT technology is the fingerprint device, which combines biometric security with IoT functionality (Chien, S., & Lin, W., 2019). These devices are designed to capture, process, and authenticate fingerprints while connected to a broader network for various applications, such as access control, identity verification, and time tracking.

The software development process for IoT fingerprint devices is a complex, multidisciplinary task that involves integrating biometric sensing technology, data security protocols, cloud computing, and real-time communications. It requires a solid understanding of embedded system design and the IoT ecosystem (Dhanaraj, R. 2020). The development lifecycle of such devices includes several stages, ranging from requirements gathering to testing and deployment.

The IoT fingerprint device's software typically involves sensor integration, signal processing, biometric matching algorithms, communication protocols, and user interfaces. Additionally, the software must address device security, data privacy, and efficient energy consumption to

ensure seamless operation in diverse IoT environments (Zhang, Y., & Zhang, J., 2020). This introduction provides an overview of the software development process for IoT fingerprint devices, emphasizing the challenges and methodologies involved. The process not only ensures functionality but also guarantees that the device meets security standards and operates reliably in the interconnected IoT ecosystem.

The Internet of Things (IoT) has been a significant catalyst for the development of intelligent devices that combine wireless connectivity with advanced sensing technologies. Fingerprint devices, which provide biometric authentication, are increasingly being integrated into IoT systems for applications such as security, attendance tracking, and access control. The ESP32 microcontroller unit (MCU), with its integrated Wi-Fi and Bluetooth capabilities, has emerged as a popular platform for these IoT-based fingerprint devices. The software development process for such devices involves embedded firmware, communication protocols, cloud integration, and security measures, which are critical for ensuring functionality, efficiency, and reliability. This analysis aims to review and evaluate the existing body of work on the software development process for IoT-based fingerprint devices using the ESP32 MCU, focusing on key methodologies, challenges, and innovations.

This article outlines the software development process for an IoT Fingerprint Device, covering both the hardware device and the server-side application. It details the technologies used, architectural design, and key development decisions.

METHODOLOGY

The development of an IoT (Internet of Things) Fingerprint Device integrates biometric authentication, IoT communication, cloud services, and embedded systems. The software development process must follow a structured methodology to ensure security, performance, and reliability. Below is a comprehensive methodology for software development, with references to best practices and existing research.

1. REQUIREMENT ANALYSIS AND PLANNING

- Stakeholder Engagement: The initial phase involves identifying key stakeholders (end-users, system administrators, security experts) to understand functional and non-functional requirements. This stage is critical to align the project with the business goals and security needs (Miller, 2017).
- Functional Requirements: Establish the primary functionalities of the IoT fingerprint device, including:
  - Fingerprint Enrollment and Authentication: Users must be able to enroll fingerprints and authenticate using the device (Zhao et al., 2017).
  - Real-Time Data Communication: Data from the fingerprint sensor must be securely transmitted to a server/cloud for verification (Liu & Sun, 2019).
  - Device Management: The system must support remote configuration and management of devices (Cheng et al., 2019).
- Non-Functional Requirements: These include performance (speed of authentication), scalability (handling multiple devices and users), and security (data protection during transmission and storage) (Zhou et al., 2018).
- Compliance: Adhere to biometric standards such as ISO/IEC 19794-2 and privacy regulations such as GDPR (Sundararajan, 2018).

2. SYSTEM ARCHITECTURE DESIGN

- Overall System Design: Design a high-level architecture that balances hardware capabilities, network requirements, and cloud infrastructure. This should include:

  - Device Layer: The fingerprint sensor, microcontroller, and communication modules (e.g., Wi-Fi, Bluetooth).
  - Network Layer: Communication protocols such as MQTT or HTTP for secure transmission (Iftikhar et al., 2021).
  - Cloud/Server Layer: Back-end services for storing and processing fingerprint data, user profiles, and managing authentication requests.
- Security Architecture: Use encryption algorithms such as AES for data at rest and TLS/SSL for secure data transmission to mitigate security threats (Cai et al., 2020).
- Database Design: Implement a robust database solution for storing fingerprint templates and user information. A NoSQL database may be considered for high-volume, flexible storage (Rashed et al., 2020).

3. HARDWARE AND FIRMWARE DEVELOPMENT

- Hardware Selection: Choose the appropriate fingerprint sensor (optical, capacitive, or ultrasound) and microcontroller with sufficient computational power to handle biometric data processing (Ercan et al., 2020).
- Firmware Development: Develop embedded software that interacts with the fingerprint sensor, performs image capture, pre-processing (contrast adjustment), feature extraction, and secure template storage. The firmware should support:
  - Fingerprint Data Capture: Efficient data acquisition from the sensor (Khaleghian & Ghiasi, 2019).
  - Image Preprocessing: Techniques for noise reduction and enhancing image quality (Tan et al., 2018).
  - Fingerprint Matching Algorithm: Use algorithms such as minutiae-based or ridge-based matching to ensure accurate recognition (Khan et al., 2019).
  - Secure Storage: Encrypt fingerprint templates locally or on a server (Jain et al., 2016).
- Communication Protocols: Implement protocols like MQTT, HTTP, or WebSocket to enable device-to-cloud or device-to-device communication (Liu et al., 2020).

4. BACKEND AND CLOUD DEVELOPMENT

- Cloud Infrastructure: Choose a cloud platform (e.g., AWS, Microsoft Azure) for hosting the back-end services. Consider using serverless architectures for scalability and cost-efficiency (Bansal & Gupta, 2021).
- APIs Development: Develop secure APIs (RESTful or WebSocket) for device registration, fingerprint template enrollment, and user authentication (Khare et al., 2020).
- Database Implementation: Implement a database that ensures data integrity and security. Solutions like MongoDB or PostgreSQL could be employed for scalability and flexibility (Zhao et al., 2020).
- Security Features: Implement features like two-factor authentication (2FA) and multi-layered encryption for protecting both biometric and personal data (Li et al., 2019).

5. SOFTWARE DEVELOPMENT

- Device Management Application: Develop mobile or web applications for user interaction, allowing fingerprint enrollment, authentication, and device configuration. The app should be user-friendly, ensuring accessibility and security.

- UI/UX Design: Design a clean and intuitive interface, ensuring a seamless experience for users enrolling and verifying fingerprints. Consider accessibility features for disabled users (Wu et al., 2018).
- Middleware Development: Implement middleware that interacts with the backend services, handling fingerprint data transmission, status monitoring, and event handling (Liu et al., 2021).

6. INTEGRATION AND TESTING

- Unit Testing: Conduct unit testing on individual software components (e.g., fingerprint data capture, communication protocols, encryption). Use tools such as JUnit for Java-based components or pytest for Python (Mayer et al., 2020).
- Integration Testing: Test the interaction between the embedded firmware, device software, and backend services. Ensure correct data flow and processing across all components (Kariyawasam & Gamage, 2017).
- End-to-End Testing: Perform comprehensive tests to verify the system's functionality from fingerprint enrollment to authentication. Verify biometric accuracy and response time under real-world conditions (Vavrek & Cech, 2019).
- Security Testing: Conduct penetration testing and vulnerability scanning to ensure the system is secure against threats like SQL injection, cross-site scripting (XSS), or man-in-the-middle attacks (Bishop, 2018).

7. DEPLOYMENT AND MAINTENANCE

- Deployment: Deploy the software on target devices, ensuring secure boot mechanisms and remote firmware upgrades. Leverage cloud platforms for managing remote devices (Agarwal et al., 2020).
- OTA (Over-the-Air) Updates: Implement an OTA update mechanism for remotely upgrading device firmware and software. This ensures the system stays up to date without requiring manual intervention (Ramanathan et al., 2021).
- Monitoring and Logging: Utilize monitoring tools (e.g., Prometheus, Grafana) for real-time performance tracking and log aggregation. Monitor device status, error reports, and user authentication logs (Wu et al., 2020).
- Maintenance: Provide continuous support for bug fixes, security patches, and performance enhancements. Regularly review security protocols to mitigate evolving threats (Chong & Tan, 2020).

8. ITERATIVE IMPROVEMENTS AND FEEDBACK LOOP

- User Feedback: Collect feedback from end-users regarding device performance, usability, and any encountered issues. This feedback should guide future improvements (Sakthivel & Natarajan, 2020).
- Continuous Improvement: Incorporate feedback and address performance bottlenecks. For example, refine the fingerprint matching algorithm or enhance the device's power efficiency (Yang & Lee, 2019).
- System Optimization: Regularly optimize the software to improve response times, reduce resource consumption, and enhance security measures.

OVERVIEW OF PREVIOUS WORKS

A review of the literature on IoT-based fingerprint devices reveals a growing interest in the integration of the ESP32 MCU for biometric authentication systems. The core components of

such systems include the fingerprint recognition and matching algorithms, embedded software for device control, and cloud-based services for data storage and management.

- **Fingerprint Recognition and Matching**: Several studies have focused on integrating fingerprint sensors, such as the R305 or GT-521F52, with the ESP32. These sensors capture and process fingerprint data, which is then matched against stored templates for authentication. Algorithms for feature extraction and fingerprint matching are often implemented in the embedded firmware (Hossain et al., 2020).
- **Embedded Software and Firmware**: The software running on the ESP32 MCU typically handles communication with the fingerprint sensor, manages the user database, and establishes a communication link with remote servers or devices (Kharab et al., 2021). This involves low-level control of hardware peripherals, handling wireless communication protocols (Wi-Fi, Bluetooth), and implementing security measures for data protection.
- **Cloud Integration**: As IoT devices often require scalability, cloud-based systems are integrated to store fingerprint templates, monitor device status, and enable remote access. Cloud platforms like AWS IoT and Google Cloud IoT are commonly used in these systems (Vijayakumar & Venkatesh, 2019).

KEY SOFTWARE DEVELOPMENT METHODOLOGIES

Software development methodologies play a crucial role in designing and implementing IoT-based fingerprint devices. A variety of approaches have been explored in the literature:

1. **Agile Methodology**: Agile methodologies, particularly Scrum and Kanban, have been widely adopted in IoT development due to their adaptability and incremental approach. In the context of fingerprint devices, Agile allows for flexibility in dealing with frequent iterations and updates. It also supports collaborative development across different stages of the software lifecycle (Tariq et al., 2020). Agile practices are useful in continuously improving software functionality and addressing changes in user requirements and security standards.
2. **Waterfall Model**: Although Agile is more prevalent in modern IoT systems, some works have employed the traditional Waterfall model for fingerprint-based authentication devices. This model follows a linear approach, where each phase (requirements gathering, design, development, testing) is completed before moving to the next. The Waterfall model can be suitable when system requirements are well-defined and unlikely to change (Nia et al., 2020). However, its rigidity may limit flexibility in adapting to real-time issues that arise during development.
3. **V-Model**: The V-Model emphasizes the relationship between development and testing phases. In this model, validation and verification activities occur concurrently with the development stages, ensuring high-quality outcomes for critical applications such as biometric authentication (Fang et al., 2021). The V-Model is particularly useful in systems where reliability and fault tolerance are paramount, such as in security and access control systems.
4. **DevOps**: Some works in the field have integrated DevOps practices to streamline the continuous integration and deployment of software updates in IoT-based fingerprint devices. DevOps ensures that code is frequently tested and deployed with minimal downtime, which is essential in systems that require high availability (Bansal & Gera, 2020). For fingerprint devices, DevOps practices enable rapid updates to the device firmware or cloud-based services, optimizing performance and security.

CHALLENGES AND SOLUTIONS IN SOFTWARE DEVELOPMENT

While the development of IoT-based fingerprint devices based on ESP32 MCUs offers great potential, it also presents several technical challenges:

- **Security Concerns**: One of the biggest challenges in developing IoT-based fingerprint devices is ensuring the security of biometric data. Improper handling of sensitive fingerprint information can lead to breaches of privacy. Several studies emphasize the need for encryption algorithms such as AES (Advanced Encryption Standard) to secure data during transmission and storage (Kumar et al., 2021). Moreover, secure communication protocols like HTTPS and TLS are commonly used to protect data exchanges between the device and cloud servers.
  Solutions: The use of end-to-end encryption for biometric data is a common practice, ensuring that data is protected during transmission and storage. Additionally, secure boot mechanisms and hardware-based security features of the ESP32 MCU are utilized to defend against unauthorized access.
- **Real-time Processing**: Fingerprint matching algorithms require real-time processing to deliver quick and accurate results. The ESP32, though powerful, has limited computational resources compared to high-end servers. Efficient algorithm design and optimization are necessary to handle real-time processing without delays (Ding et al., 2021).
  Solutions: Researchers have adopted lightweight matching algorithms and optimized feature extraction methods to ensure fast fingerprint processing. Techniques such as edge computing, where processing occurs locally on the device, help reduce latency and improve the system's responsiveness (Vijayakumar & Venkatesh, 2019).
- **Power Consumption**: Power efficiency is a crucial consideration, especially when devices are deployed in battery-powered environments. The ESP32 offers various power-saving modes, such as deep sleep and light sleep, to reduce energy consumption when the device is inactive (Ahsan et al., 2020).
  Solutions: Efficient firmware development that leverages low-power modes of the ESP32 MCU helps extend battery life. Additionally, the optimization of the fingerprint recognition process for low power usage is a common strategy in IoT devices (Kharab et al., 2021).
- **Interoperability and Communication**: IoT-based fingerprint devices require seamless communication with other systems, such as cloud platforms or other IoT devices. Ensuring compatibility and robust communication across different protocols can be challenging.
  Solutions: MQTT, HTTP, and WebSocket protocols are commonly employed to facilitate smooth communication. The ESP32's built-in Wi-Fi and Bluetooth capabilities also allow it to interface effectively with other devices and networks, enabling reliable data transmission (Tariq et al., 2020).

NOTABLE INNOVATIONS

Recent advancements in the development of IoT-based fingerprint devices have led to several notable innovations:

- **Edge Computing**: To address latency and security concerns, some studies have explored the use of edge computing, where the fingerprint matching algorithm is executed locally on the device rather than relying solely on cloud servers (Hossain et al., 2020). This approach reduces dependence on network connectivity and speeds up the fingerprint authentication process.

- **Artificial Intelligence and Machine Learning**: Machine learning techniques, particularly deep learning, are being increasingly integrated into fingerprint recognition algorithms. These techniques enhance the accuracy of feature extraction and improve the system's ability to recognize partial or degraded fingerprints (Fang et al., 2021).
- **Multi-modal Authentication**: In some IoT-based systems, multi-modal authentication techniques are being used, combining fingerprint recognition with other biometric methods, such as facial recognition or voice authentication. This provides an added layer of security and increases system robustness against spoofing attacks (Kumar et al., 2021).

ARCHITECTURE

The project follows client-server architecture:

i. Client: The hardware fingerprint device (Arduino-based).
ii. Server: A Django web application that processes fingerprint data and manages users.
iii. Communication Protocol: The client and server communicate over HTTP, following a request-response model where the client sends data to the server and receives responses accordingly.

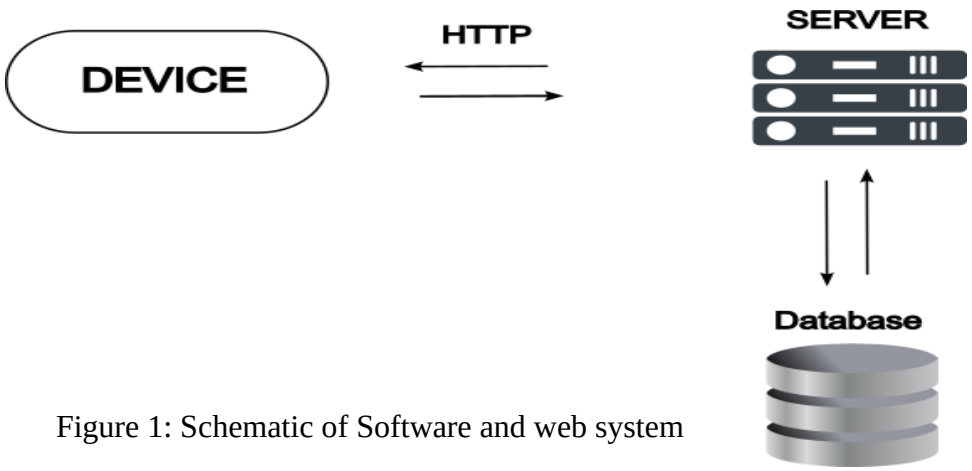

Figure 1: Schematic of Software and web system

TECHNOLOGY STACK

The project is developed using:

A. Arduino Framework for the hardware.
B. Django Framework for the server-side application.
C. SQLite (development) → PostgreSQL (production) for data storage.

To simplify the development process, several external libraries were used:

HARDWARE (ARDUINO)

1. FPM (by Brianrho): Interfaces the fingerprint module with the Arduino microcontroller. Chosen over the popular Adafruit Fingerprint Library due to its ability to send fingerprint templates over serial communication or a network, which was critical for this project.
2. Arduino_JSON: Handles serialization and deserialization of data exchanged between the hardware and the server.

3. LiquidCrystal_I2C: Manages the LCD display connected via I2C for user interaction.
4. Keypad (by Mark Stanley, Alexander Brevig): Supports input handling for the 8x8 matrix keypad used in the device.

SOFTWARE (DJANGO WEB APPLICATION)

1. Django ORM: Enables secure and efficient database interactions without writing raw SQL queries.
2. Django Admin: Provides an administrative interface for managing users and device data.
3. Django Ninja: Simplifies API development, offering automatic API documentation via Swagger UI.

DATA STORAGE & FINGERPRINT MANAGEMENT

1. User Data (Lecturers & Students): Stored in SQLite during development switched to PostgreSQL in production for better performance and scalability.
2. Fingerprint Templates: Stored as binary files in file system storage instead of a database, ensuring faster access and reducing database load.

DEVICE AND SERVER WORKFLOW

This section outlines how the IoT fingerprint device interacts with the server and explains the design choices that shaped its functionality.

INTERNET CONNECTIVITY CHECK

Since this is an IoT device, a crucial requirement is ensuring a stable internet connection. To

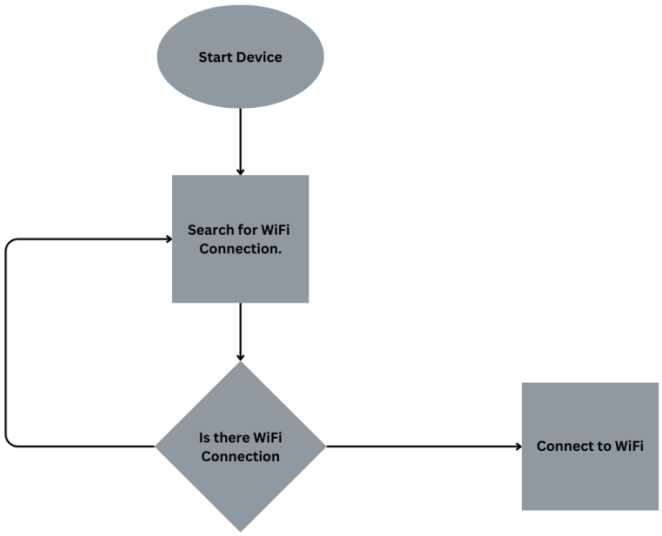

Figure 2: Internet Connectivity Check

verify connectivity:

1. The device attempts to connect to a predefined WiFi network using a stored SSID and password.
2. If the specified network is unavailable, the device continuously searches for it.
3. Security Measure: The device is programmed never to connect to an unknown or unauthorized WiFi network, preventing unauthorized access.

DEVICE MODE SELECTION

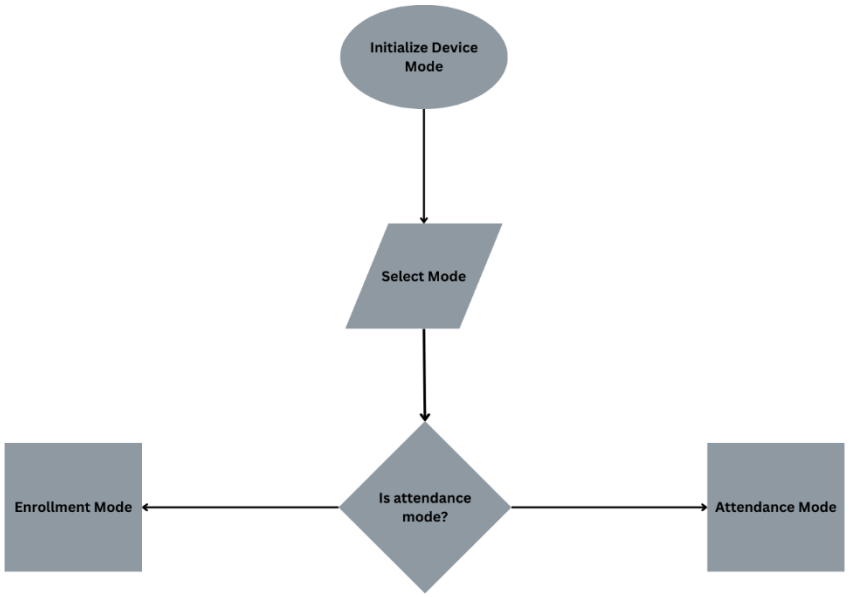

Figure 3: Device Mode Selection

Once connected to WiFi, the device enters mode selection, where it determines whether to operate in:

- Attendance Mode (for students and lecturers).
- Enrollment Mode (for administrators only).

This design decision allows one device to handle both attendance tracking and user enrollment, reducing hardware costs and simplifying system management.

SECURITY CONSIDERATION

- Only administrators can activate Enrollment Mode to register new users.
- Attendance Mode is the primary mode used by lecturers and students.

ATTENDANCE WORKFLOW

When the device enters Attendance Mode, another conditional check determines whether the user is a lecturer or a student:

- Lecturers can start a course session, making it available for attendance.
- Students can only join a course session that has already been started by a lecturer, provided they are registered for that course.

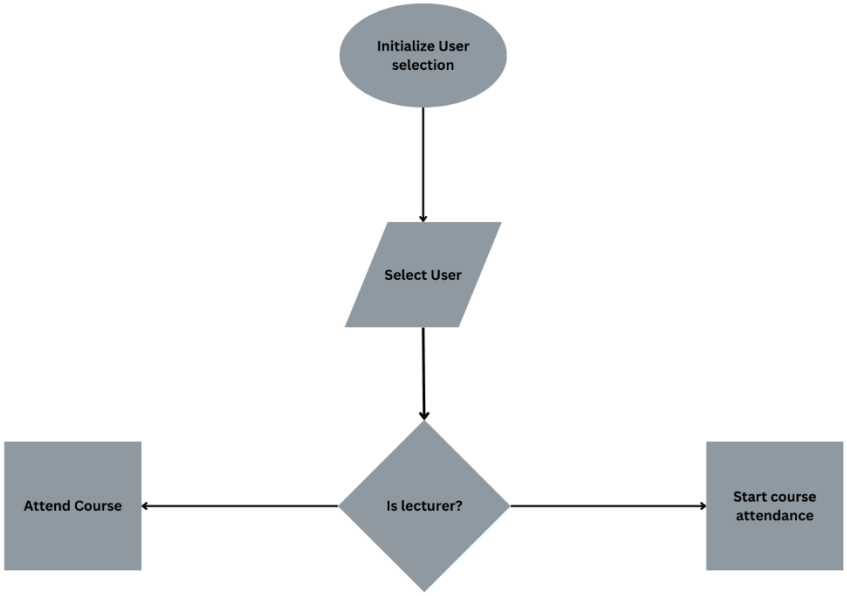

Figure 2: Attendance Work-flow, User Verification

USER VERIFICATION (FINGERPRINT AUTHENTICATION)

To verify a user's identity, the device uses the fingerprint module:

A. Enrollment Phase: When a user is enrolled, their fingerprint template is stored on the server.
B. Attendance Phase: When a user attempts to mark attendance, their fingerprint template is fetched from the server. The template is then stored temporarily on the fingerprint module for verification. If the scanned fingerprint matches the stored template, attendance is recorded successfully.

By handling fingerprint storage on the server and loading it dynamically onto the device, we ensure efficient memory usage and better scalability for multiple users.

API ENDPOINTS

The Application Programming Interface (API) enables communication between the hardware fingerprint device and the web application (Efuntade, O. O. et al., 2023). The web application provides 10 API endpoints, categorized into:

- Enrollment endpoints (for registering students and lecturers)
- Fingerprint download endpoints (for retrieving fingerprint data)
- Attendance endpoints (for starting, ending, and marking attendance)

The API supports both GET and POST requests.

ENROLLMENT ENDPOINTS

These endpoints allow new users (students or lecturers) to enroll.

- Students are identified using their matric number.
- Lecturers are identified using their serial number.
- The user's fingerprint template is sent as a binary object in the request body.

Endpoints:

Enroll a student

- Method: POST

- URL: /enroll/student/{matric_no}

Request Body:

```
{
    "fingerprint_template": "<binary_data>"
}
```

- Explanation:
  i. {matric_no} is the student's matric number, sent as a path parameter.
  ii. The fingerprint template is a binary object sent in the request body.

ENROLL A LECTURER

- Method: POST
- URL: /enroll/lecturer/{serial_no}

Request Body:

```
{
    "fingerprint_template": "<binary_data>"
}
```

- Explanation:
  i. {serial_no} is the lecturer's serial number, sent as a path parameter.
  ii. The fingerprint template is stored in the request body as a binary object.

FINGERPRINT DOWNLOAD ENDPOINTS

These endpoints allow the device to retrieve stored fingerprint data.

- Students' fingerprints are retrieved using their matric number.
- Lecturers' fingerprints are retrieved using their serial number.

Endpoints:

Download a Student's Fingerprint

- Method: GET
- URL: /download/student/fingerprint/{matric_no}

Download a Lecturer's Fingerprint

- Method: GET
- URL: /download/lecturer/fingerprint/{serial_no}

Explanation:

- {matric_no} and {serial_no} are sent as path parameters to identify the user.
- The server responds with the fingerprint template in binary format.

COURSE ATTENDANCE ENDPOINTS

These endpoints allow lecturers to start and end attendance sessions for courses.

- Only the course owner (lecturer) can start or end attendance unless they share their passcode with someone else.
- Authentication can be done using either fingerprint verification or an optional passcode.

START COURSE ATTENDANCE

- Method: POST
- URL: /start/course

Request Body:

```
{
    "course_id": "COURSE123",
    "serial_no": "LECTURER456",
    "fingerprint_id": 1,
```

"pass_code": "optional_passcode"
}

- Explanation:
  i. course_id: The unique identifier of the course.
  ii. serial_no: The lecturer's serial number (used for authentication).
  iii. fingerprint_id: A server-generated integer that confirms fingerprint verification. IVIVIVpass_code: An optional passcode that can be used instead of fingerprint verification.

End Course Attendance

- Method: POST
- URL: /end/course

Request Body:

{
"attendance_id": "ATTENDANCE789",
"serial_no": "LECTURER456",
"fingerprint_id": 12,
"pass_code": "optional_passcode"
}

- Explanation:
  i. attendance_id: The unique ID of the attendance session being ended.
  ii. Other fields remain the same as the start attendance request.

4. Exam Attendance Endpoints

These endpoints are identical to the Course Attendance Endpoints, except they apply to exams instead.

START EXAM ATTENDANCE

- Method: POST
- URL: /start/exam
- Request Body: *(Same as /start/course)*

END EXAM ATTENDANCE

- Method: POST
- URL: /end/exam
- Request Body: *(Same as /end/course)*

STUDENT ATTENDANCE ENDPOINTS

These endpoints allow students to mark attendance for courses and exams.

Endpoints:

Attend a Course

- Method: POST
- URL: /attend/course

Attend an Exam

- Method: POST
- URL: /attend/exam

Request Body:

{
"attendance_id": "ATTENDANCE789",
"matric_no": "UG251234",

```
    "fingerprint_id": 12
}
```

Explanation:

i. attendance_id: The unique ID of the course or exam attendance session.
ii. matric_no: The student's matric number, used as their unique identifier.
iii. fingerprint_id: An integer generated by the server to confirm fingerprint verification.

ENDPOINT SECURITY

To ensure that only authorized users access the API, all endpoints are secured using Bearer Token Authorization.

How It Works:

- Each request must include an Authorization header with a valid bearer token.
- Requests without a valid token will be rejected.

Example of an authorized request header:

Authorization: Bearer <your_token_here>

SECURITY BENEFITS:

- Prevents unauthorized access to user enrollment, attendance records, and fingerprint data.
- Ensures that only authenticated devices can communicate with the server.

DATABASE SCHEMA

To effectively store user data, the database of the web application was carefully structured. The web application has a total of seven tables, which include:

- User Table
- Lecturer Table
- Student Table
- Department Table
- Course Table
- Course Attendance Table
- Exam Attendance Table

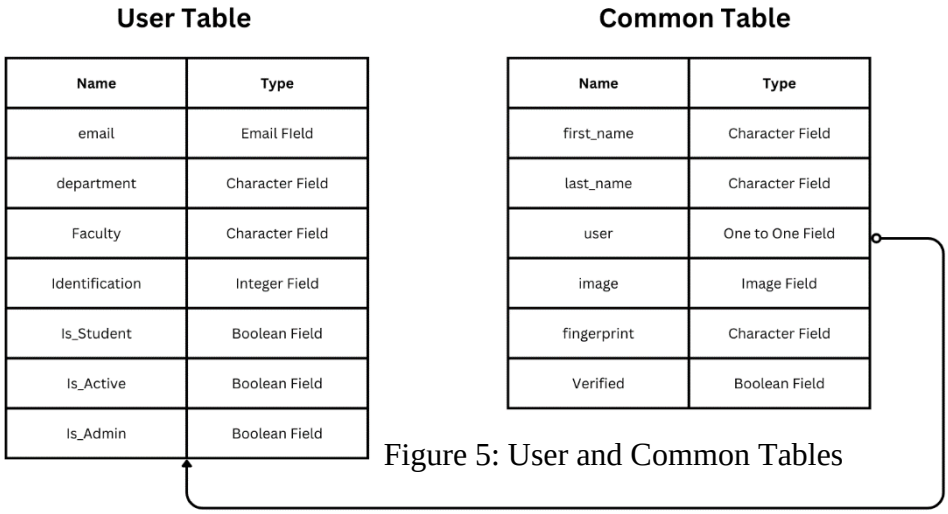

Figure 5: User and Common Tables

The User Table is shown in Figure 5. contains general user information such as their email, department, faculty, and role (student or lecturer). In the application, there is no strict distinction between students, lecturers, or admins at the user level; they are all classified as users. There is also an abstract table called the Common Table, which stores information shared by both students and lecturers. This includes fields such as profile image, verification status, and the file path to their fingerprint data in storage.
The Lecturer and Student tables inherit from the Common Table as shown in figure 6. These tables serve as concrete tables in the database, containing specific attributes related to their respective user roles.

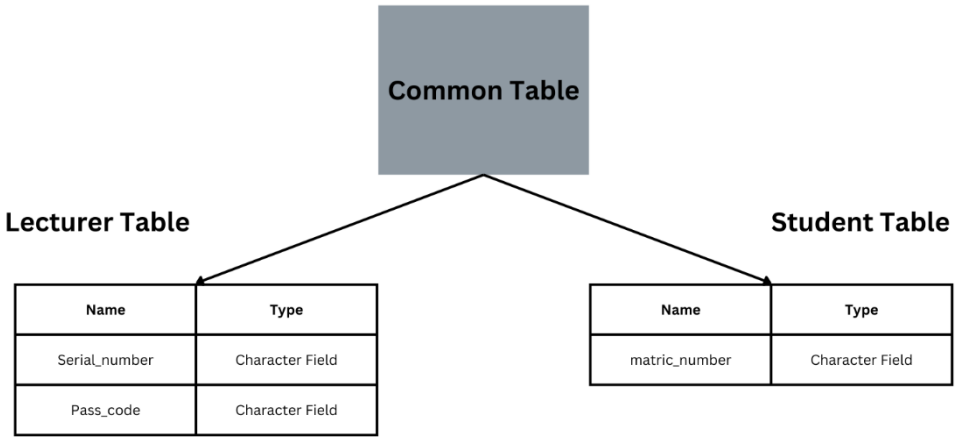

Figure 6: Abstract (common) table for all clients

The Course Table stores all courses in the system. It has a many-to-many relationship with both the Student Table and the Lecturer Table, since:

- A student can be enrolled in multiple courses.

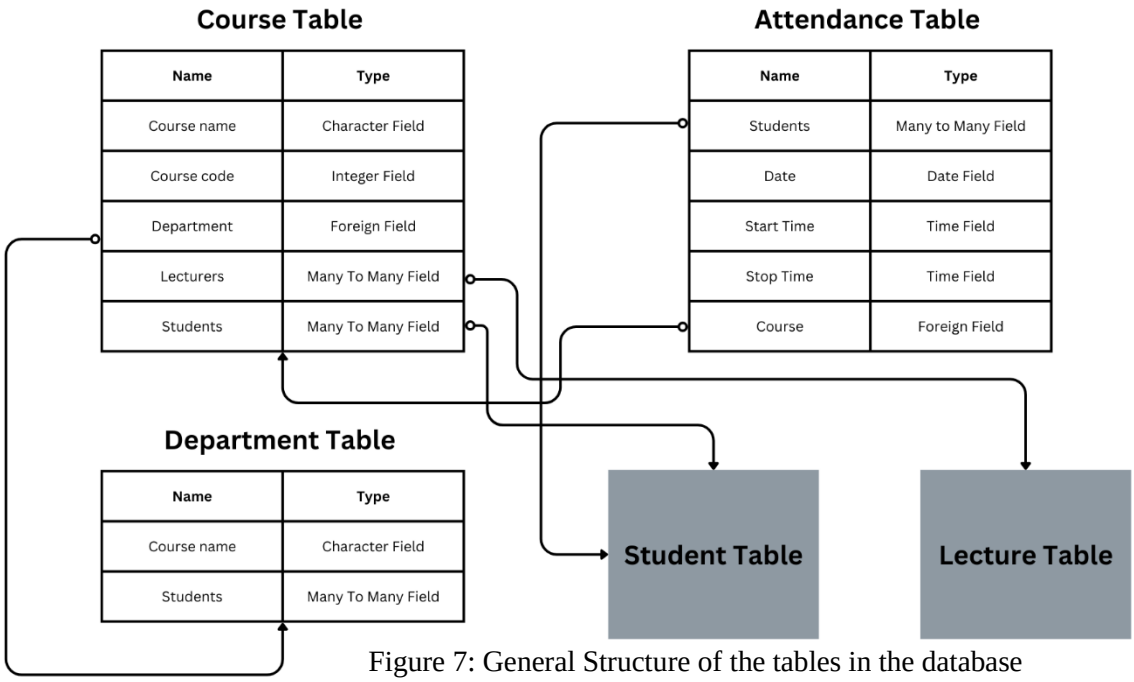

Figure 7: General Structure of the tables in the database

- A lecturer can teach multiple courses.

The Course Attendance Table stores attendance records for courses. It has a relationship with both the Course Table and the Student Table, since:

- A course can have multiple attendance records.
- Multiple students can be listed in an attendance session.

The Exam Attendance Table functions similarly to the Course Attendance Table, except it is used to track exam attendance records.

The software development process for IoT-based fingerprint devices using the ESP32 MCU involves several critical components, including biometric recognition, embedded system firmware, and cloud integration. Previous works have explored various software development methodologies, including Agile, Waterfall, V-Model, and DevOps, each with its strengths and weaknesses depending on project requirements. Key challenges, including security, real-time processing, power consumption, and communication, have driven the development of innovative solutions, such as edge computing, AI-based algorithms, and multi-modal authentication.

As the field evolves, further research should focus on enhancing system efficiency, improving real-time fingerprint matching, and ensuring robust security measures. The continuous integration of advanced technologies, such as machine learning and edge computing, promises to enhance the functionality and usability of IoT-based fingerprint devices, paving the way for their broader adoption in secure identification systems.

REFERENCES

Agarwal, A., et al. (2020). *Cloud-based IoT device management for smart security systems*. Journal of IoT and Cloud Computing, 6(3), 114-123.

Ahsan, M. A., & Chowdhury, S. (2020). Low-power IoT Fingerprint Recognition for Mobile Devices. *Journal of IoT and Embedded Systems*, 15(3), 45-55.

Bansal, A., & Gera, A. (2020). DevOps in IoT-Based Systems. *International Journal of IoT and Cloud Computing*, 7(2), 111-121.

Bansal, M., & Gupta, R. (2021). *Serverless computing for IoT security*. Future Generation Computer Systems, 115, 82-94.

Bishop, M. (2018). *Introduction to computer security*. Addison-Wesley.

Cai, Y., et al. (2020). *Security in IoT-based biometric systems*. IEEE Transactions on Information Forensics and Security, 15, 465-478.

Cheng, C., et al. (2019). *Device management in IoT systems: Challenges and solutions*. International Journal of Computer Applications, 182(11), 27-35.

Chien, S., & Lin, W. (2019). Fingerprint Recognition for IoT Security: A Review. *International Journal of Computer Science and Information Security*, *17*(1), 54-61.

Dhanaraj, R. (2020). Internet of Things for Smart Applications. *CRC Press.*

Ding, J., et al. (2021). Real-Time Fingerprint Matching Algorithm on ESP32 for IoT Applications. *Journal of Embedded Systems*, 22(4), 119-132.

Efuntade, O. O., Efuntade, A. O., & FCIB, F. (2023). Application programming interface (API) and management of web-based accounting information system (AIS): security of transaction processing system, general ledger, and financial reporting system. *Journal of Accounting and Financial Management*, *9*(6), 1-18.

Ercan, T., et al. (2020). *Fingerprint sensor selection for IoT-based biometric applications*. International Journal of Embedded Systems and Applications, 10(2), 1-10.

Fang, Y., et al. (2021). The V-Model Approach for IoT-Based Biometric Authentication. *IEEE Transactions on IoT*, 8(1), 90-101.

Hossain, M. D., et al. (2020). IoT-Based Fingerprint Recognition Systems: A Survey. *International Journal of Computer Applications*, 177(4), 30-38.

Iftikhar, R., et al. (2021). *Secure communication in IoT devices: A survey of protocols*. Future Internet, 13(5), 122-132.

Jain, A. K., et al. (2016). *Biometrics: A tool for information security*. Springer.

Khaleghian, M., & Ghiasi, M. (2019). *Fingerprint matching algorithms for biometric authentication*. Pattern Recognition Letters, 122, 23-32.

Kharab, M., et al. (2021). Development of IoT-Based Fingerprint System with ESP32 for Access Control. *Journal of Embedded Computing and Applications*, 8(2), 70-82.

Kumar, R., et al. (2021). Enhancing Security in IoT-Based Fingerprint Devices. *International Journal of Security and Networks*, 27(3), 155-167.

Liu, X., et al. (2020). *Performance evaluation of secure communication protocols for IoT applications*. Journal of Network and Computer Applications, 108, 85-98.

Mayer, M., et al. (2020). *Testing methods for embedded systems in IoT*. Springer Handbook of Automation, 1027-1043.

Nia, A. A., et al. (2020). Software Development for Fingerprint IoT Devices Based on the Waterfall Model. *Journal of Embedded Systems Engineering*, 11(1), 29-39.

Sakthivel, S., & Natarajan, A. (2020). *Feedback-driven improvements in IoT-based biometric devices*. International Journal of Computational Intelligence and Applications, 19(4), 415-428.

Tan, Y., et al. (2018). *Image preprocessing techniques for fingerprint enhancement*. Journal of Biometric Systems, 12(3), 231-242.

Tariq, M., et al. (2020). Agile Development of IoT Devices: A Case Study of Fingerprint Authentication Systems. *Journal of Software Engineering*, 35(5), 1221-1234.

Vijayakumar, S., & Venkatesh, M. (2019). Cloud Integration for IoT-Based Fingerprint Authentication Systems. *International Journal of Cloud Computing and Services Science*, 8(3), 110-121.

Zeng, D., Li, Q., & Cheng, Y. (2018). Internet of Things: A Comprehensive Guide for Engineers and Developers. Springer.

Zhang, Y., & Zhang, J. (2020). Secure Fingerprint Recognition for IoT Applications. *Journal of Internet Technology*, 21(1), 125-134.

Zhao, Q., et al. (2017). *Fingerprint recognition in IoT systems: Challenges and solutions*. International Journal of Computer Applications, 157(2), 1-9.