Device Identity Mechanism in Smart IOT Based Building

A Dissertation submitted

for the partial fulfilment of the degree of

Bachelor of Engineering

in

Information Technology

(Session 2024-25)

Guided by

Mr. Jay Singh

Submitted By

Aditya Swamesh (21I7006)

Hrithik Chouriya (22I7092)

Nikesh Sharnagat (22I7094)

Department of Information Technology

Institute of Engineering & Technology

Devi Ahilya Vishwavidyalaya, Indore (M.P.)

(www.iet.dauniv.ac.in)

(2024-25)

Dissertation Approval Sheet

The dissertation entitled **"Device Identity Mechanism in Smart IOT Based Building"** submitted by **Aditya Swamesh (21I7006), Hrithik Chouriya (22I7092), Nikesh Sharnagat (22I7094)** is approved as partial fulfilment for the award of **Bachelor of Engineering in Information Technology** degree by **Devi Ahilya Vishwavidyalaya, Indore**.

Internal Examiner **External Examiner**

Director

Institute of Engineering & Technology

Devi Ahilya Vishwavidyalaya,

Indore (M.P.)

Recommendation

The dissertation entitled **"Device Identity Mechanism in Smart IOT Based Building"** submitted by **ADITYA SWAMESH (21I7006) , HRITHIK CHOURIYA (22I7092), NIKESH SHARNAGAT (22I7094) is** a satisfactory account of the bonafide work done under my supervision is recommended towards the partial fulfilment for the award of **Bachelor of Engineering Information Technology** degree by **Devi Ahilya Vishwavidyalaya, Indore.**

Date:

MR. JAY SINGH
Project Guide

Endorsed By:

Head, Department of Information technology

Candidate Declaration

We hereby declare that the work which is being presented in this project entitled **Device Identity Mechanism in Smart IOT Based Building** in partial fulfilment of degree of Bachelor of Engineering in Information Technology is an authentic record of our own work carried out under the supervision and guidance of **Mr. Jay Singh**, in Department of **Computer Engineering**, Institute of Engineering and Technology, Devi Ahilya Vishwavidyalaya, Indore.

We are fully responsible for the matter embodied in this project in case of any discrepancy found in the project and the project has not been submitted for the award of any other degree.

Date:

Place:

ACKNOWLEDGEMENTS

Words can never express the extent of indebtness but we still wish to express our deep sense of gratitude to all the people who helped us in the completion of this project.

I want to express my heart-felt gratitude to Mr. Jay Singh for her advises and unremitting support over the last one year. She trained me how to do research and how to write, encouraged me not to be intimidated by the difficult problems. She is so generous with her time and ideas. Her intellectual creativity, perseverance and commitment would benefit me for the rest of my life. I could never thank him enough for being an excellent mentor and a wonderful teacher.

My thanks also go to the other members of my college Dr. Vrinda Tokekar for the discussion during the course of the project, for the reading of the draft of this thesis and providing valuable feedback.

I want to thank the director of our college Dr. Vrinda Tokekar for assisting us. Therefore, my thanks go to them for making this project possible.

I also thank my team members and friends for supporting me and helping me out with the testing phase as well as rest of the project.

Finally, I want to thank my parents for giving me strength and love.

ABSTRACT

This project focuses on the development of a smart IoT-based building device identity mechanism using ESP32, RFID, and a variety of peripherals such as an LED, buzzer, and LCD display. The aim is to create a system that efficiently identifies devices and stores their identity in a database for monitoring and control purposes. RFID tags are used to identify devices, with an RFID reader connected to the ESP32 microcontroller. The system processes device information and stores it in a database while triggering visual (LED) and audio (buzzer) alerts to confirm data storage.

The project addresses the problem of secure and automated device identification in smart buildings, offering a solution that allows for the seamless integration of devices into a centralized IoT system. The system ensures that every device added to the building is recognized and stored efficiently, helping administrators maintain a well-organized and secure environment. By simulating the system using Proteus and developing the architecture and logic through StarUML and Arduino IDE, a practical, real-time system was created and tested.

The key learnings include gaining a deep understanding of IoT hardware and software integration, database handling, and real-time data processing. The solution is highly scalable, enabling further enhancements such as adding biometric verification, facial recognition, or integrating with cloud-based data storage. The project concludes that smart device management systems can be made more efficient and secure through IoT technologies, benefiting a wide range of building automation scenarios.

TABLE OF CONTENTS

Chapter-1 Introduction

1.1 Overview and issues involved

Automated, convenient, and simplified security are possible with smart IoT-based systems as homes and buildings become more technologically integrated. Device identity methods play a critical role in these kinds of systems, particularly when it comes to real-time monitoring and access control. In this particular scenario, effective procedures to identify people and devices while protecting data are necessary for maintaining device IDs in a smart building

Current Issues:

1. **Absence of Secure Authentication**: Conventional access control methods susceptible to loss, copying, or manipulation include physical keys and keypads.

2. **Unauthorized Access**: As smart IoT devices proliferate, unauthorized access becomes a major worry, particularly when devices lack sufficient authentication.

3. **Data Management and Storage**: Monitoring who entered the facility and when necessitates the use of a safe database that can hold information without being hacked.

4. **Scalability**: As the number of devices and users in a building increase, existing solutions frequently become unworkable.

5. **Device Integration**: A lot of smart systems have trouble combining different devices, such controllers, RFID, and sensors, into a unified, functional system.

1.2 Problem definition

The need for scalable and secure device identity methods for smart buildings leveraging IoT components is the main problem that this project tries to solve. In particular, the system should integrate electronic components such as RFID, ESP32 microcontrollers and peripherals such as buzzers, LEDs and LCDs to provide a reliable and user-friendly way to manage access to the facility. The existing access control techniques are too complicated for wider use in less smart buildings, or they are too basic and compromise security. This project uses current IoT technologies to create a straightforward, yet safe solution in an attempt to close the gap.

1.3 Proposed solution

The project uses an ESP32 microcontroller as the brain of the system, integrating RFID module, LED, buzzer, LCD screen and database for recording access data. When an RFID tag is scanned, the system verifies that the user is authorized by checking the stored valid ID database. If the tag is valid, the system activates an LED to indicate that access has been granted, activates a buzzer for audible feedback, and displays the user's name on the LCD screen. If the tag is invalid, the system denies access and activates an audible alarm to warn of unauthorized entry.

System Architecture & Flow Diagram

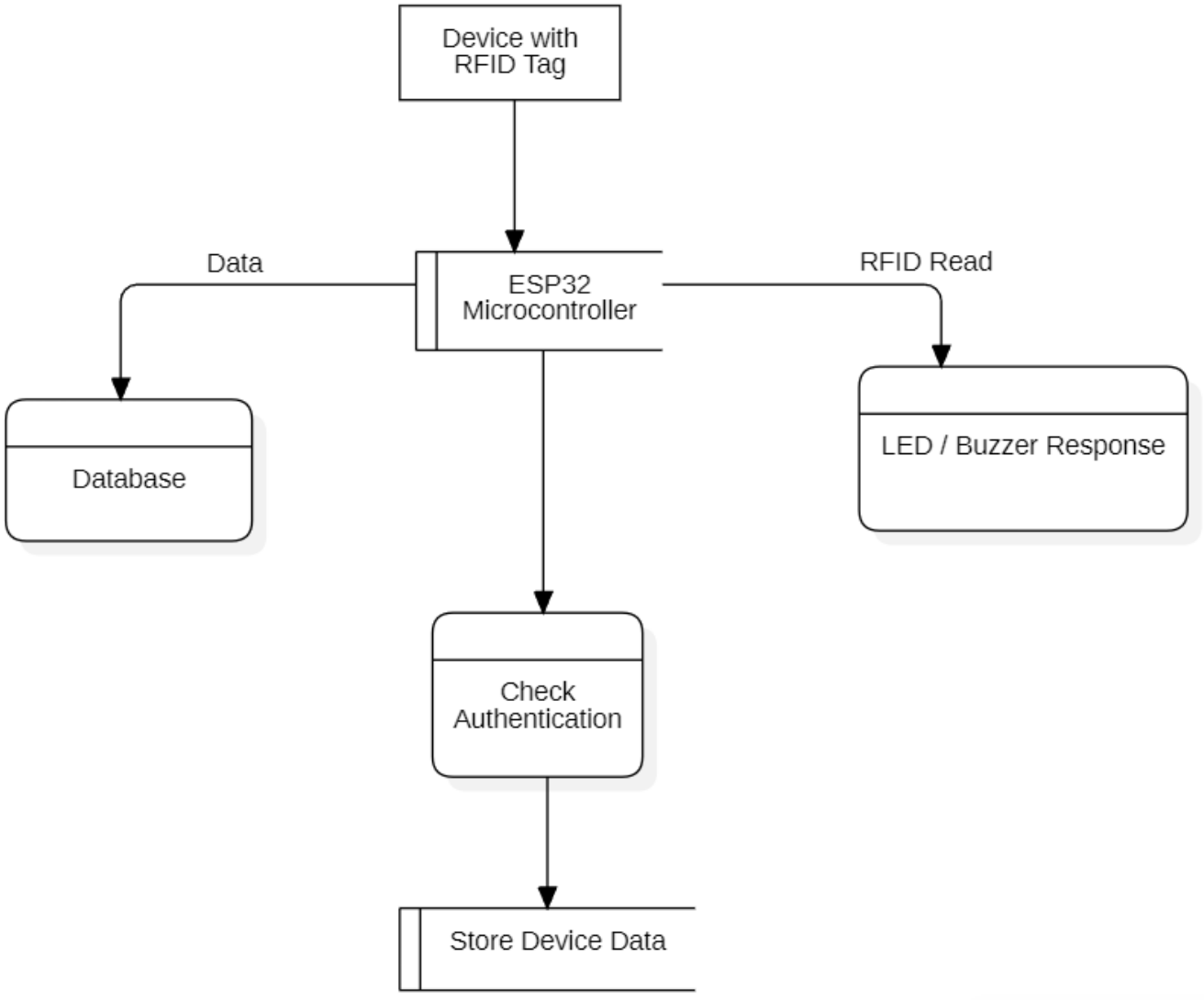


Figure 1.3.1 data flow diagram

1. RFID reader: Each device has an RFID tag, and when brought close to the reader, the RFID data (device ID) is captured.

2. It checks if the device is registered in the database and either grants or denies access.

3. Repository: The ESP32 communicates with a remote repository that lists all authorized devices. It stores the device ID and all necessary metadata for easy retrieval.

4. LED and buzzer: LED and buzzer provide visual and audible feedback. A green LED indicates a valid device ID, while a red LED and beep indicates an unauthorized or invalid device.

Functionalities Developed

• **RFID-based Device Identification**: Each device has a unique RFID tag that serves as its identity marker.

• **ESP32 as a Processing Unit**: The ESP32 reads data from the RFID tag and checks it against the database. • Secure data storage: Device IDs and associated data are securely stored in a central database, ensuring that unauthorized access does not occur.

• **Instant authentication**: The system authenticates devices immediately when they enter or leave the network. • Feedback mechanism: LED and buzzer provide immediate feedback to the user based on device status (authorized/unauthorized).

Justification of superiority over existing solutions

This project addresses several limitations of current market solutions:

1. **Increased security**: unlike systems that rely on easily falsified identifiers such as MAC addresses, our solution uses RFID tags, which are more difficult to copy and counterfeit. The database ensures that all devices are registered and provides a secure validation layer.

2. **Scalability**: With ESP32 and a database, you can easily scale. As the number of devices increases, you can add more devices to the database and the system can continue to run smoothly with real-time identification.

3. **Cost-Effectiveness**: The ESP32 board is a low-cost microcontroller with built-in Wi-Fi and Bluetooth, making it a cost-efficient solution for IoT-based systems. 4. Real-time feedback: The

integration of LEDs and buzzers provides immediate feedback to users, improving the usability of the system. 5. Low Power Consumption: ESP32 boards are known for being power efficient, making them ideal for power-sensitive IoT applications. 6. Reliable Data Processing: Using a database, the system can efficiently store, manage, and retrieve large amounts of device data, reducing delays and the possibility of data loss.

Chapter-2 Literature Survey

2.1 Methodology

This section will outline the steps involved in achieving the project's objectives, focusing on conceptual requirements, methods, and algorithms relevant to understanding the project.

Project Development Stages

The development of a device identification mechanism in a smart building based on the Internet of Things can be divided into the following key stages:

1) Identifying problems:

• Research and identify the gaps in existing solutions for identifying devices in buildings based on the Internet of Things.

• Determine the specific requirements of the proposed system, focusing on device identification, security and scalability.

2) System Design:

• Design the system architecture including hardware and software components.

• This step involves selection of appropriate technologies like ESP32, RFID reader, database etc. to form the backbone of the device ID mechanism.

3) Implementation:

• Develop an RFID based device identification system using ESP32.

• Create an internal infrastructure (database) to securely store device identifiers.

• Implement a communication system for ESP32 to interact with the database for real-time authentication.

4) Testing:

• Test the system under various conditions including authorized and unauthorized device access scenarios.

• Ensure all components (RFID, ESP32, Database, LED, Buzzer) work smoothly together.

5) Optimization:

Optimize the system for low power consumption, scalability and performance while processing large amounts of data.

Conceptual Requirements

• **Internet of Things (IoT):** IoT refers to the network of interconnected devices that communicate and share data over the internet. In a smart building, IoT devices can automate functions such as lighting, HVAC, and security.

• **RFID (Radio Frequency Identification):** RFID technology uses electromagnetic fields to automatically identify and track tags attached to objects. In this project, RFID tags are used to provide unique identification to each device. The RFID reader captures this tag and sends the data to the ESP32.

• **ESP32 Microcontroller**: The ESP32 is a low-cost, low-power SoC with built-in Wi-Fi and Bluetooth. It works as a major control unit in this project, is in charge of reading RFID data, interaction with the database, and supplying comments by LED and buzzer.

• **Database**: Database is essential to save surrounding identities safely. ESP32 communicates with a database to check device information and store records. You can use any efficient cloud or local database system (MySQL, Firebase, etc.) for this project

• **LED and buzzer mechanism**: These components provide real-time feedback to the user. Once a device is recognized, the system uses an LED to indicate success (green light) or failure (red light) and a buzzer to sound an alarm if tampering has occurred.

• **Security Mechanisms:** Authentication and secure data transfer between devices and databases are essential to ensure the integrity of the system. In this project, encryption techniques are used to ensure secure data transfer.

• **Cloud Computing**: In some cases, the database can be cloud-based to allow remote access and scalability for the IoT system. This will allow devices to be identified from multiple locations and help manage larger networks.

Key Algorithm: RFID-Based Device Authentication

The RFID-based device authentication algorithm can be summarized as follows:

Step 1: Read the RFID tag when the device enters the network.

Step 2: Send the RFID data to the ESP32 microcontroller. Step 3: ESP32 processes the data and sends it to the database for verification.

Step 4: The database checks if the device identifier is stored.

Step 5: If the device ID is valid, allow access (turn green LED on); otherwise, deny access (turn red LED on and trigger the buzzer).

2.2 Existing Solutions

We will look at some existing solutions that try to solve the problem of device identification in smart buildings. These solutions may cover some or all of the problem, and we will consider their advantages and disadvantages as well as how they compare to the proposed system.

1. MAC Address-Based Authentication

One of the current solutions that identifies devices on the IoT network is to use Mac for authentication. In many intellectual construction systems, each device is determined by the MAC address, and the network recognizes this unique identifier for communication.

advantage:

The MAC address is unique to each device.

Easy to implement with a network protocol context.

Disadvantages:

MAC addresses can be spoofed, creating security vulnerabilities.

MAC address-based systems are not flexible enough for dynamic environments where new devices are constantly being added or removed.

GAP: Our project provides a more secure alternative to MAC addresses using RFID tags, which are difficult to copy and provide physical layer security.

2. NFC (Near Field Communication)-Based Identification

Some smart buildings use NFC technology to identify devices and users. NFC is similar to RFID, but it typically has a shorter range and is more commonly used in smartphone-based systems for access control.

Advantages:

• NFC is widely supported by smartphones, making it user-friendly.

• It provides a secure way to identify nearby devices and users.

Drawbacks:

- Limited range compared to RFID.
- NFC tags are more expensive and require close contact for successful identification.

Gap: While NFC is effective, it lacks the scalability and reach that RFID offers. In large smart buildings, the longer range of RFID offers greater flexibility in identifying devices remotely.

3. Biometric Access Control

In some advanced smart buildings, we use a biometric system (for example, finger imprinting, face recognition, etc.) to identify users and devices.

Advantages:

• Biometric identification is highly reliable and unique for each person.

• These systems reduce the risk of unauthorized access.

Drawbacks:

• Biometric systems are expensive to deploy and maintain.
• Non-biometric devices cannot be easily integrated into a network.

Gap: In comparison to biometric systems, our approach is more affordable and scalable, and because it prioritizes device identity above user identity, it is more applicable to IoT networks in smart buildings.

4. ZigBee-Based Device Management

ZigBee protocols are used by certain smart building systems to identify and communicate with devices. ZigBee is a low-power, wireless mesh networking technology.

Advantages:

• Because it uses relatively little power, ZigBee is perfect for battery-powered devices

• enables communication between devices without the need for a central hub.

Drawbacks:

• The data transmission range and capabilities of ZigBee are restricted.
• Compared to RFID-based solutions, ZigBee networks' security features are less robust.

Gap: Although ZigBee is effective for small-scale networks, our concept uses ESP32 and RFID to offer increased data transmission capabilities, improved scalability, and enhanced security.

5. Wi-Fi Based Identification and Access Control

Access control systems that rely on Wi-Fi are a common feature of IoT-based smart buildings. In these systems, devices are verified according to their network permissions after connecting to the network with Wi-Fi credentials.

Advantages:

• Wide coverage throughout buildings; because to its ubiquity,

• Wi-Fi is easy to combine with a variety of devices.

Drawbacks:

• Because they need a lot of energy, Wi-Fi systems are not as effective for low-power devices and are vulnerable to hacking due to compromised network credentials.

Gap: Our system provides a more secure and energy-efficient solution by utilizing RFID for device identification instead of network-based authentication, which overcomes security concerns.

Chapter-3 Analysis

3.1 Software Requirements

A number of software components are required for the Device Identity Mechanism to be implemented successfully in Smart IoT-Based Buildings in order to guarantee that all functionality are met. The main software requirements are listed below, along with explanations for each.

1. ESP32 Microcontroller

Description:
The ESP32 is an inexpensive, low-power system on a chip (SoC) with built-in Bluetooth and Wi-Fi.

Justification for Use:

- **Versatility**: The ESP32 supports multiple communication protocols, making it ideal for IoT applications where connectivity is crucial.
- **Performance**: With a dual-core processor, the ESP32 provides sufficient processing power to handle multiple tasks simultaneously, such as reading RFID data and communicating with a database.
- **Energy Efficiency**: Its low power consumption is essential for battery-operated devices, ensuring long-term functionality in smart building environments.

2. RFID Technology

Description:
With the use of electromagnetic fields, radio-frequency identification (RFID) technology can automatically recognize and follow tags affixed to things.

Justification for Use:

- **Unique Identification**: RFID tags provide every gadget a distinct identification, improving management and security in an ecosystem of smart buildings.

- **Non-Contact Reading**: RFID makes it possible to read device identities quickly and without contact, enabling smooth system interaction.
- **Scalability**: RFID technology can readily handle more tags as the number of devices rises without requiring major infrastructural modifications.

3. Database Management System

Description:
Software for building and maintaining databases that enables data storage, retrieval, and manipulation is known as a database management system (DBMS).

Justification for Use:

- **Secure Data Storage**: The DBMS provides a secure method for storing device identities and user access records, ensuring data integrity and protection.
- **Efficient Data Management**: It makes data management more orderly and facilitates fast updates and queries when new devices are added or withdrawn.
- **Scalability**: Expanding smart building environments can benefit from a well-designed database's ability to manage an increasing number of device entries.

3.2 Hardware Requirements

Certain hardware components are needed for the Device Identity Mechanism in the Smart IoT-Based Building project in order to guarantee effective operation, communication, and overall system performance. The main hardware parts are listed below, along with an explanation of each and the reasons behind their presence.

1) ESP32 Microcontroller

Description:
With built-in Bluetooth and Wi-Fi, the ESP32 is a potent microcontroller that is well-suited for Internet of Things applications.

.

Justification for Use:

- **Connectivity:** The ESP32 can effectively connect with other devices and the database because to its compatibility for both Wi-Fi and Bluetooth. For control and data transfer to occur in real time, this link is essential.
- **Processing Power**: Its dual-core processor has the processing capability to manage database transactions, regulate LEDs and buzzers, and read RFID data, among other duties.
- **Low Power Consumption**: Because of its energy-efficient architecture, the ESP32 is perfect for battery-operated devices in smart building environments where power conservation is crucial.

2. RFID Reader

Description:

An RFID reader is a gadget that reads data from RFID tags by releasing radio waves.

Justification for Use:

- **Device Identification:** Reading the distinct identities of the gadgets within the smart building requires the use of an RFID reader. It improves security and management by making it possible to quickly and precisely identify each device.
- **Non-Contact Functionality:** RFID technology enables seamless functioning within the building environment by enabling reading without the need for physical contact..
- **Scalability:** It is simple to expand the RFID system to handle more devices without having to make major adjustments to the infrastructure.

3. RFID Tags

Description:

RFID tags are tiny chips and antenna-equipped devices that can store and send data to an RFID reader.

Justification for Use:

- **Unique Identification**: With a unique identifier assigned to each tag, it is possible to precisely trace devices inside the smart building.
- **Durability**: Because RFID tags are usually durable and resistant to a wide range of environmental factors, they can be used for a variety of purposes inside a building.

- **Cost-Effectiveness**: Since RFID tags are typically cheap, large-scale deployments can benefit financially from using them.

4. LEDs (Light Emitting Diodes)

Description:

LEDs are semiconductor devices that emit light when an electric current passes through them.

Justification for Use:

- **Visual Indicators:** LEDs improve user involvement and awareness by providing visual input for different system states, such as successful RFID readings, faults, or system alarms.
- **Low Power Consumption:** Since EDs use less energy than conventional bulbs, they can be used in smart building systems for extended periods of time.
- **Compact Size:** They do not take up much room and may be easily integrated into a variety of setups thanks to their tiny form factor.

5. Buzzer

Description:

An electromechanical device known as a buzzer emits sound when electricity is applied.

Justification for Use:

- **Auditory Alerts**: In order to guarantee that users receive notifications instantly, the buzzer offers audible feedback for user activities or alerts, such as successful device identification or cautions about faults.
- **Simple Integration**: Microcontrollers can be easily integrated with buzzers, and they can be controlled with straightforward programming, facilitating rapid development cycles.
- **Cost-Effective:** Buzzers are relatively inexpensive, making them an economical choice for providing sound feedback in the project.

6. Power Supply

Description:

A power supply is an essential component that provides the necessary voltage and current to power the ESP32 and other hardware components.

Justification for Use:

- **Stable Operation:** To avoid malfunctions or system failures, a dependable power supply makes sure that the microcontroller and any associated devices receive a steady voltage.
- **Flexibility**: The power supply can be modified to deliver the necessary voltage and current for the particular application, depending on the building's power configuration.
- **Safety**: Appropriate power supply improve system safety by providing protections against short circuits and overcurrent's.

7. Breadboard and Jumper Wires

Description:

Jumper wires are used to connect components on a breadboard, which is a reusable platform for electronic circuit prototyping.

Justification for Use:

- **Prototyping**: With the help of breadboards, circuits may be easily tested and prototyped without the need for soldering, allowing for rapid development adjustments.
- **Flexible Connections:** Jumper wires enable flexible connections between components, which simplifies the process of modifying the circuit layout during the testing stage.
- **Cost-Effectiveness**: Both breadboards and jumper wires are low-cost options that provide significant benefits during the prototyping stage.

3.3 Analysis Diagrams

3.3.1 Use Case Model

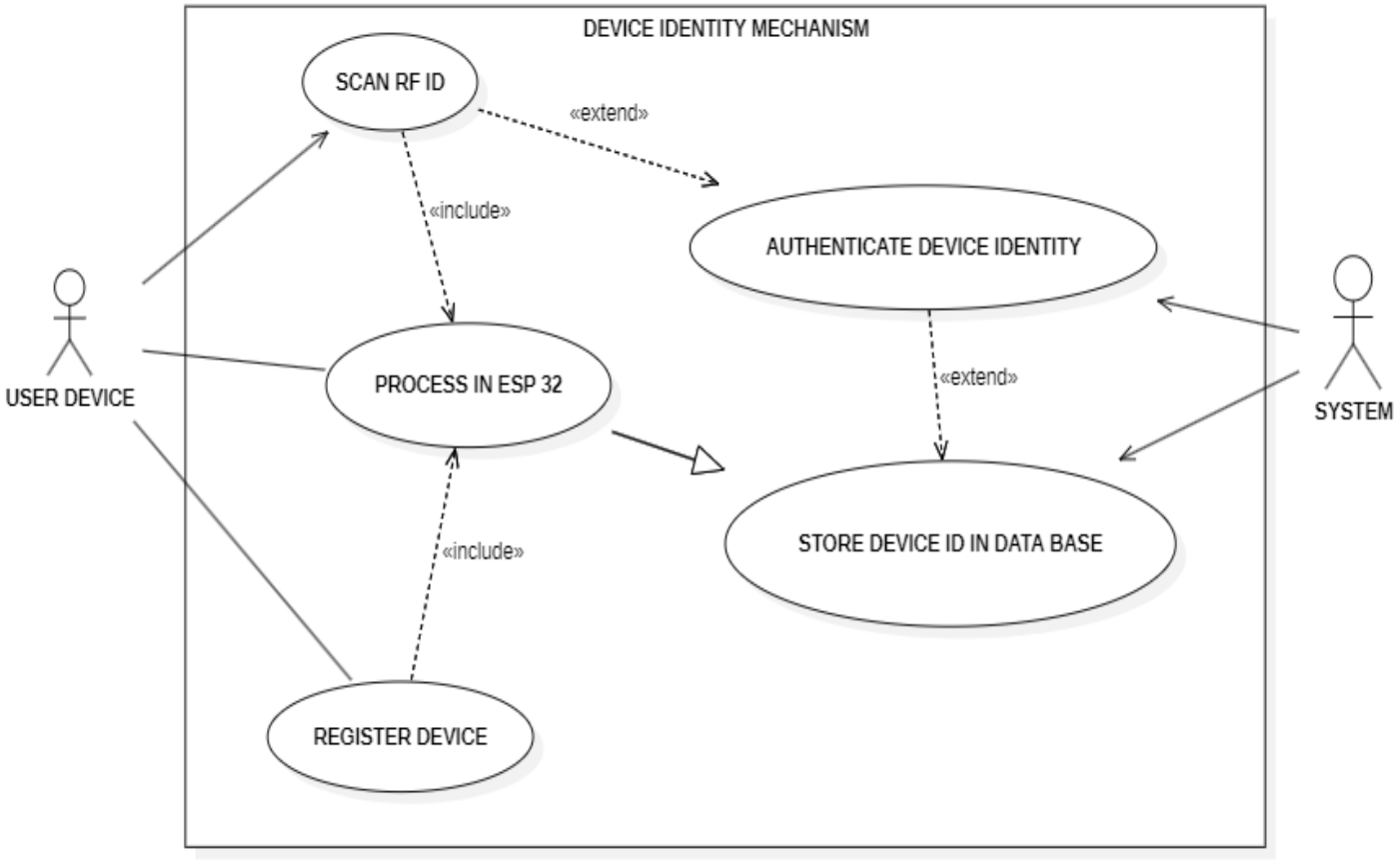


Figure 3.1 USE CASE MODEL

3.3.2 Use Case Description

Use Case 1: Scan RFID

ID: UC1

Actor(s): User Device, System

Description: The system scans the RFID to collect the device ID, which is passed to the ESP32 for further processing.

Preconditions:

- The RFID reader is functional and connected to the ESP32

Postconditions:

- The RFID data is successfully read and forwarded to the ESP32.

Main Flow:

1. 1. The user presents the RFID tag to the RFID reader.
2. The RFID reader reads the unique ID from the tag.
3. The ESP32 receives the collected ID from the system and processes it.

Alternative Flow:

- The system tries again or tells the user of the failure if the RFID cannot be read.

Special Requirements:

- The specified frequency range and protocol must be supported by the RFID reader.

Use Case 2: Process in ESP32

ID: UC2

Actor(s): System

Description: The ESP32 processes the RFID data to determine whether it's valid for further actions like authentication or storage.

Preconditions:

- RFID data is available.
- The ESP32 is connected to the database.

Postconditions:

- The RFID data is either forwarded for authentication or stored in the database.

Main Flow:

1. The ESP32 receives the RFID data.

2. The ESP32 checks if the RFID is registered or needs to be registered.
3. If the device is new, it triggers the “Register Device” use case.

Alternative Flow:

- If the data is corrupted, the system will discard it and notify the user of the error.

Special Requirements:

- Real-time processing should be fast enough to avoid delays

Use Case 2: Process in ESP32

ID: UC2
Actor(s): System
Description: The ESP32 processes the RFID data to determine whether it’s valid for further actions like authentication or storage.
Preconditions:

- RFID data is available.
- The ESP32 is connected to the database.

Postconditions:

- The RFID data is either forwarded for authentication or stored in the database.

Main Flow:

1. The ESP32 receives the RFID data.
2. The ESP32 checks if the RFID is registered or needs to be registered.
3. If the device is new, it triggers the “Register Device” use case.

Alternative Flow:

- If the data is corrupted, the system will discard it and notify the user of the error.

Special Requirements:

- Real-time processing should be fast enough to avoid delays

Use Case 4: Authenticate Device Identity

ID: UC4
Actor(s): System
Description: The system authenticates the device identity by comparing the RFID data with the stored records in the database.
Preconditions:

- The database contains valid registered devices.

Postconditions:

- The system confirms whether the device identity is valid or not.

Main Flow:

1. The system receives RFID data from the ESP32.
2. The system queries the database for a matching record.
3. If a match is found, the device is authenticated.

Alternative Flow:

- If no matching record is found, the system rejects the authentication request and triggers an alert.

Special Requirements:

- Secure database access is required to ensure data integrity.

Use Case 5: Store Device ID in Database

ID: UC5
Actor(s): System
Description: The system stores the RFID data into the database for future reference, either as a new registration or an access log.
Preconditions:

- The RFID data must be valid and authenticated.

Postconditions:

- The RFID data is successfully stored in the database.

Main Flow:

1. After authentication, the system sends the device ID and associated data to the database.
2. The database stores the data and confirms successful storage.

Alternative Flow:

- If there is a database error, the system retries or alerts the administrator.

Special Requirements:

- Database capacity must be sufficient to store all registered devices

3.4 Design Diagrams

3.4.1 Architecture Diagram

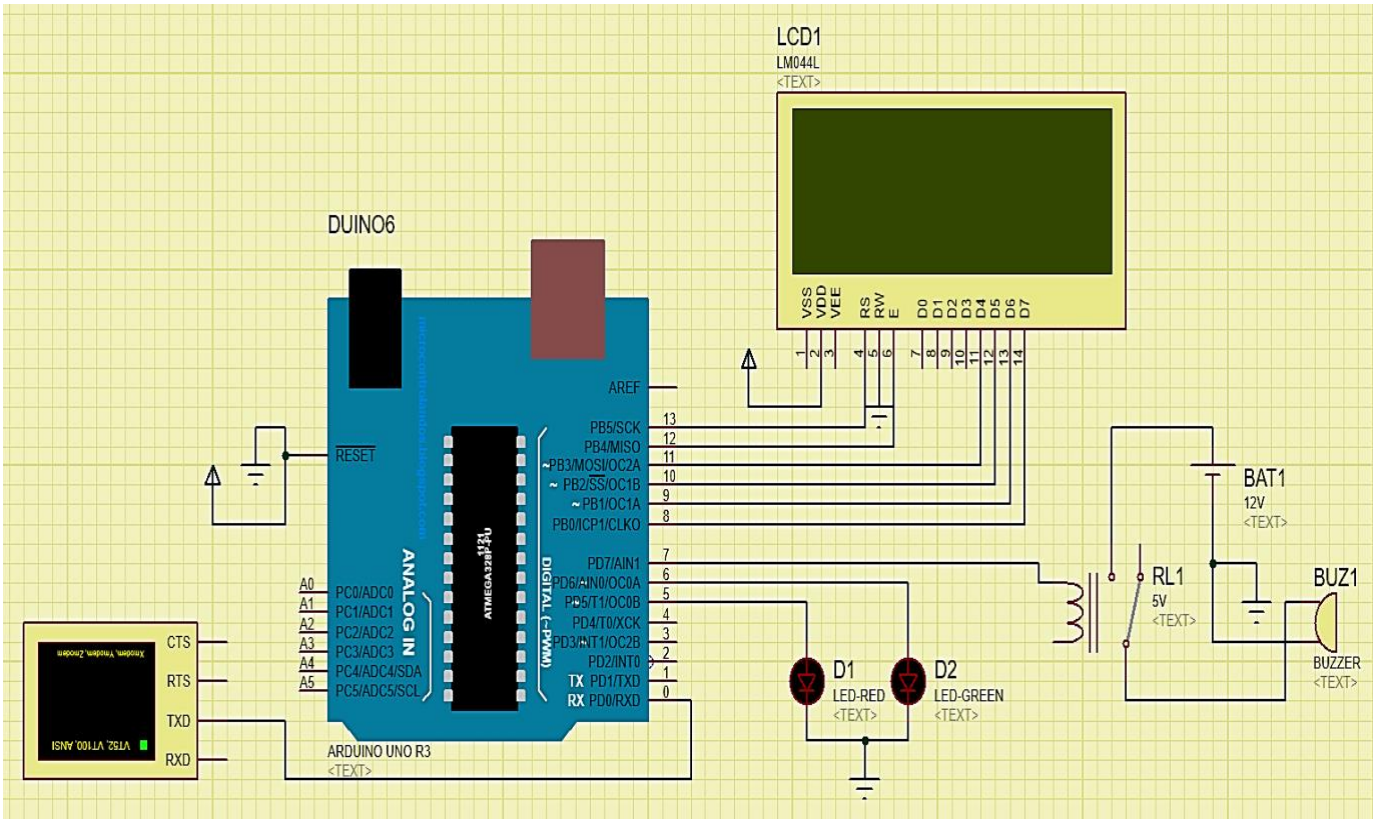


Figure 3.4.1.1 circuit diagram

The solution is cantered on an Internet of Things (IoT)-based building security system's Device Identity Mechanism. RFID technology, which is controlled by an ESP32 or Arduino microcontroller, guarantees secure and smooth identification of devices through interaction between several stakeholders (user devices, system, and database).

- **User Device Interaction**: An RFID reader attached to the ESP32 reads the tags that users scan.
- **System Processing**: The ESP32 processes the received RFID data and checks if the device is registered. If the RFID is valid, it proceeds with authentication and grants access.
- **System Output**: Depending on the results of the processing, the system can activate indicators like LEDs (red or green) or sound a buzzer to provide feedback.

Stakeholders

- **Users**: The users interact with the system by scanning their RFID tags.
- **System**: Includes the ESP32, database, and components like LEDs, buzzers, and an LCD for user feedback.
- **Administrator**: Manages the device registration process and monitors the database to ensure all devices are accounted for.

Architectural Strategy Description

The primary goal of this architecture is to ensure secure and reliable device identity management in a smart building environment. The system is designed to register devices, authenticate them using RFID tags, and store device identity information in a centralized database for monitoring and control.

Key Architectural Features:

1. **Microcontroller-Centric Design**: The ESP32 microcontroller acts as the central processing unit, coordinating all operations, including RFID scanning, data transmission, and feedback using LEDs and buzzers.
2. **RFID-Based Device Identification**: RFID tags are used to provide a unique identifier for each device. The RFID reader scans these tags and sends the data to the microcontroller for processing.

3. **Real-Time Feedback**: LEDs and buzzers are used to provide real-time feedback based on the progress of the device identification check. If a device is authenticated successfully, an LED will light up, and a buzzer will ring.
4. **Centralized Database for Storage**: For later use, device identifying data is kept in a centralized database. By doing this, the system is guaranteed to be able to track devices throughout time and keep track of device authentications and registrations.

Block diagram

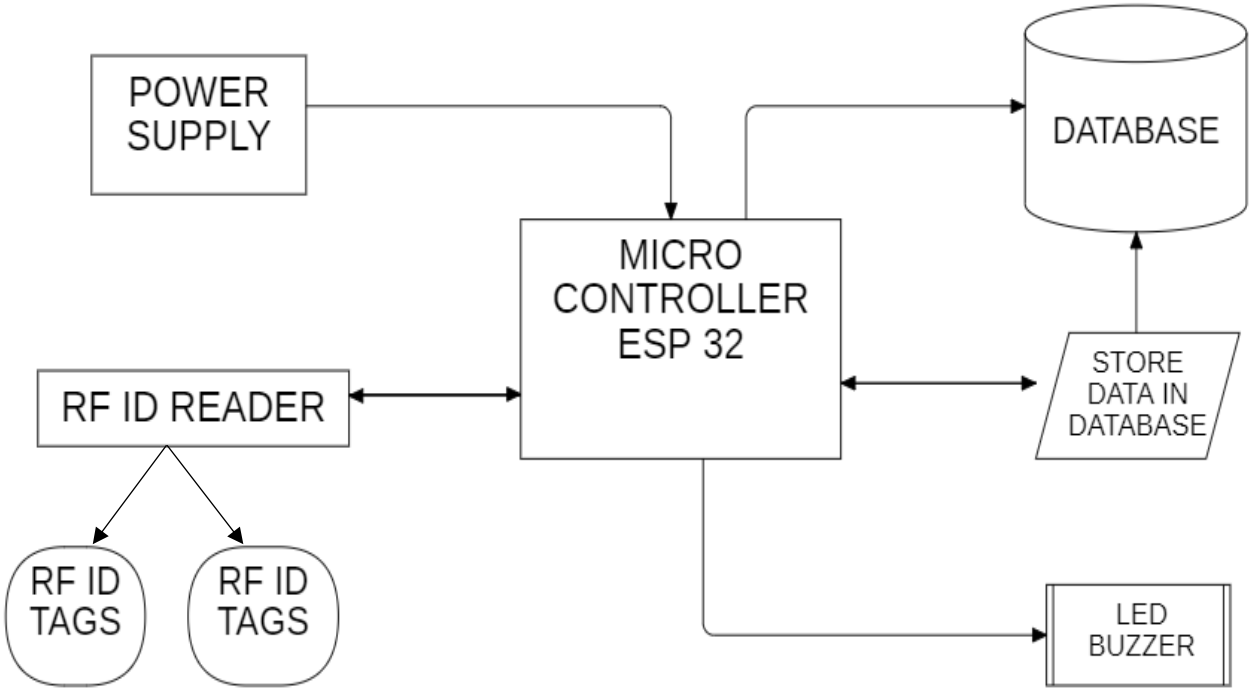


Figure 2.4.1.2 block diagram

Block Diagram of Architecture

As depicted in the provided block diagram, the architecture consists of the following components:

1. **Power Supply**: Powers the entire system, including the ESP32 microcontroller, RFID reader, LEDs, and the buzzer.

2. **Microcontroller (ESP32)**: acts as the system's brain, handling data from the RFID reader, regulating the buzzer and LED output devices, and establishing a connection with the database to save device configurations.
3. **RFID Reader** delivers the information to the ESP32 for processing after reading the RFID tags that are affixed to the gadgets. In the building, it aids in device identification.
4. **RFID Tags**: These tags are attached to different devices and contain a unique identifier. When a tag is scanned, the identifier is transmitted to the ESP32.
5. **LED and Buzzer**: Provide visual and audible feedback to the user. The LED lights up to signify successful recognition, and the buzzer sounds to confirm the activity.
6. **Database**: A centralized system for storing device data. In order to store the scanned device ID and retrieve any previously stored data for verification, the microcontroller establishes communication with the database.

Explanation of the Interaction

- The RFID reader scans the device's RFID tag as soon as it enters the building.
- The ESP32 microcontroller receives the scanned RFID data and processes it. and verifies whether the device is already registered.
- To verify whether the device is recognized, the system consults the database.
    - If recognized, the system proceeds to authenticate the device.
    - If unrecognized, the system may register the device (depending on implementation).
- The buzzer will sound and the LED will light up to indicate that the authentication was successful.
- The identity and status of the device are recorded in the database and are kept up to date.

3.4.2 Sequence diagrams.

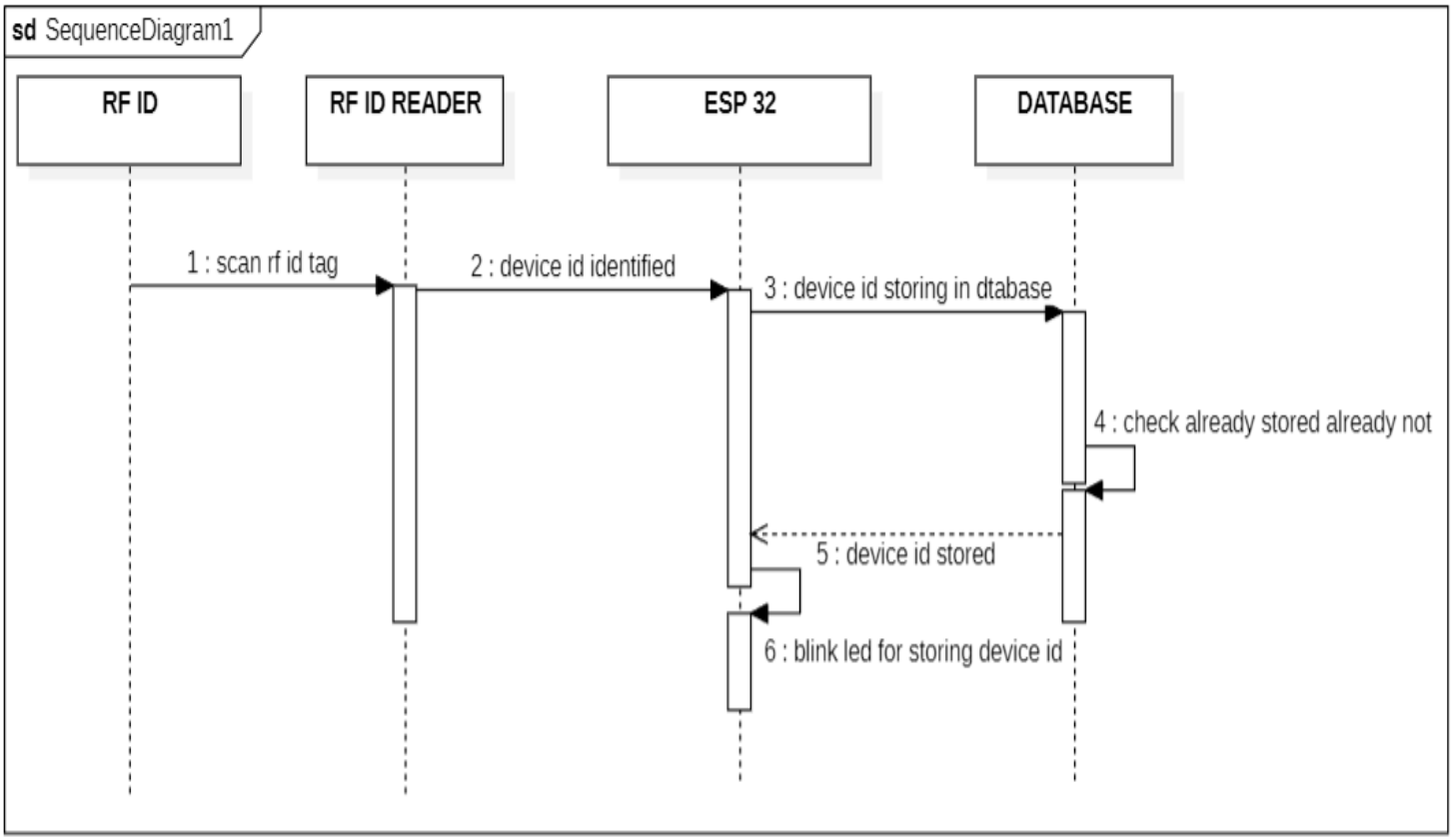


Figure 3.4.2.1 sequence diagram

Description

Scan RFID Tag (RF ID to RF ID Reader):

- The process begins when an RFID tag is scanned by the RFID reader. The tag contains a unique identifier associated with a device.
- This action is represented by the message "1: scan rf id tag" from the RFID to the RFID Reader.

Device ID Identified (RF ID Reader to ESP32):

- The RFID reader reads the unique ID from the scanned RFID tag and sends this data to the ESP32 microcontroller.
- This step is indicated by the message "2: device id identified," where the RFID Reader passes the scanned device ID to the ESP32.

Store Device ID in Database (ESP32 to Database):

- After processing the RFID data it has received, the ESP32 asks the database to record the device ID.
- This stage is illustrated by the message "3: device id storing in database," which indicates that the ESP32 talks to the database in order to store the scanned ID.

Check if Already Stored (Database to ESP32):

- The database determines if the device ID is already stored in the system before storing it. It will not store the device ID again if it already exists.
- The database verifies if the device ID exists, as indicated by the message "4: check already stored or not," which appears.

Device ID Stored (Database to ESP32):

- If the device ID is not found, the database proceeds to store the new device ID.
- The device ID has been successfully entered into the database, as indicated by the message "5: device id saved."

Blink LED for Storing Confirmation (ESP32):

- After the device ID is successfully stored in the database, the ESP32 triggers a confirmation by blinking an LED to provide visual feedback.
- The message "6: blink LED for storing device id" illustrates this operation, indicating that the user is notified of the successful storage by the LED.

Chapter – 4 Implementation and Testing

4.1 Database Design

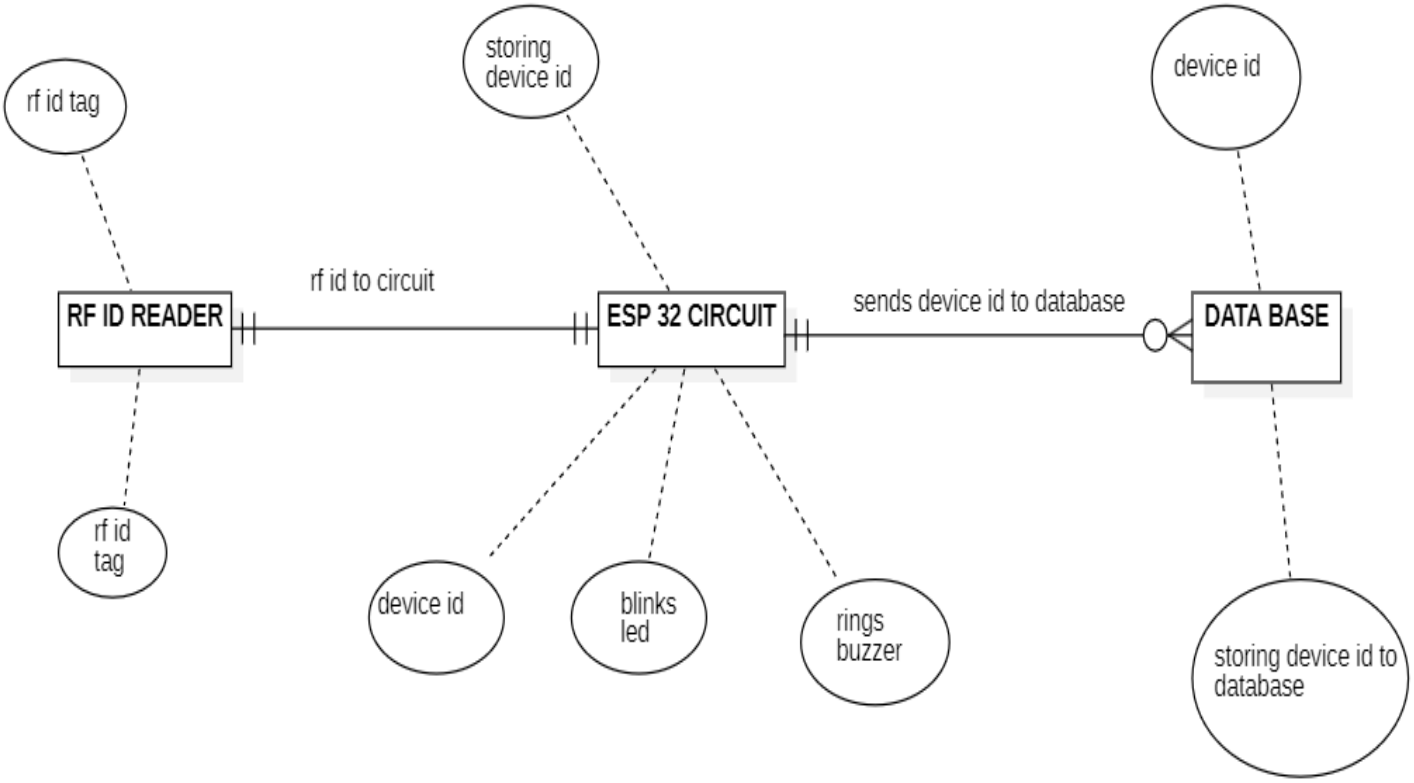


Figure 4.1.1 Entity relationship diagram

1. Entities and Attributes

RFID_Tag

Attributes: RFID_ID (PK), RFID_Code, Device_ID (FK)

Device

Attributes: Device_ID (PK), Device_Name, Device_Type, RFID_Tag (FK)

ESP32_Circuit

Attributes: Circuit_ID (PK), Device_ID (FK), Status_LED, Status_Buzzer

Database

Attributes: DB_ID (PK), Device_ID (FK), Data_Stored, Timestamp

Authentication

Attributes: Auth_ID (PK), RFID_ID (FK), Device_ID (FK), Timestamp

2. Relationships and Cardinalities

RFID_Tag - Device: One-to-One (1:1)

Each RFID tag is assigned to one specific device, and each device has only one RFID tag.

Device - ESP32_Circuit: One-to-One (1:1)

Each device is controlled by one ESP32 circuit, and one ESP32 circuit controls only one device.

Device - Database: One-to-One (1:1)

Each device's data is stored in the database, and each record in the database corresponds to one device.

Device - Authentication: One-to-Many (1)

A device can have multiple authentication events, but each event corresponds to one specific device.

ESP32_Circuit - LED/Buzzer: One-to-Many (1)

One ESP32 circuit controls multiple peripherals like LEDs and buzzers.

Snapshot of Each Table Structure

1. RFID_Tag Table

Field Name	Data Type	Description
RFID_ID	INT (PK)	Unique identifier for RFID tag
RFID_Code	VARCHAR	Code associated with RFID tag
Device_ID	INT (FK)	Foreign key to Device table

Description: This table stores all RFID tags and associates them with specific devices.

3. Device Table

Field Name	Data Type	Description
Device_ID	INT (PK)	Unique identifier for the device
Device_Name	VARCHAR	Name of the device
Device_Type	VARCHAR	Type/category of the device
RFID_Tag	INT (FK)	Foreign key to RFID_Tag table

Description: This table stores information about each device, including its name, type, and the associated RFID tag.

4. ESP32_Circuit Table

Field Name	Data Type	Description
Circuit_ID	INT (PK)	Unique identifier for the circuit
Device_ID	INT (FK)	Foreign key to Device table
Status_LED	BOOLEAN	Status of the LED (on/off)
Status_Buzzer	BOOLEAN	Status of the buzzer (on/off)

Description: This table stores data for the ESP32 circuits, including the device they control and the statuses of the LED and buzzer.

4. Database Table

Field Name	Data Type	Description
DB_ID	INT (PK)	Unique identifier for the database record
Device_ID	INT (FK)	Foreign key to Device table
Data_Stored	TEXT	Data related to the device
Timestamp	DATETIME	Time when the data was stored

Description: This table stores data related to devices, including their current status and the time of storage.

5. Authentication Table

Field Name	Data Type	Description
Auth_ID	INT (PK)	Unique identifier for authentication event
RFID_ID	INT (FK)	Foreign key to RFID_Tag table
Device_ID	INT (FK)	Foreign key to Device table
Timestamp	DATETIME	Time of the authentication event

Description: This table stores each authentication event, associating it with the corresponding RFID tag and device.

4.2 Class diagram

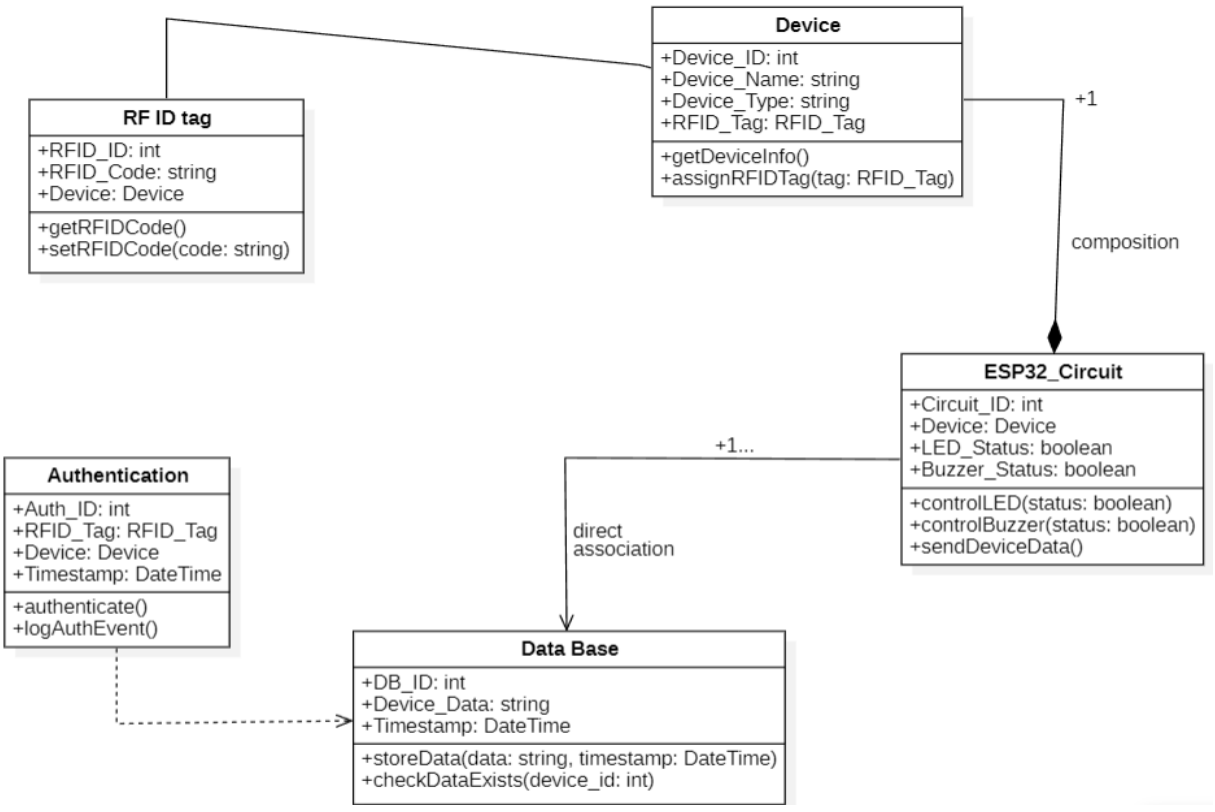


Figure 4,2,1 class diagram

Class Diagram (Description)

1. RFID_Tag Class

- **Purpose**: Represents an RFID tag, which is used for device identification.

Attributes	Type	Description
RFID_ID	int	Unique identifier for RFID tag
RFID_Code	string	Code embedded in RFID tag
Device	Device	Associated device object

Methods:

- getRFIDCode() -> string: Returns the RFID code.
- setRFIDCode(code: string): Updates the RFID code.

Explanation:

- **getRFIDCode()**: This method retrieves the current RFID code associated with the tag. It takes no parameters and returns the RFID code.
- **setRFIDCode(code)**: This method is used to update the RFID code. It takes a string as input (code), representing the new RFID code for the tag.

2. Device Class

- **Purpose**: Models a device connected to the system. Each device has a corresponding RFID tag.

Attributes	Type	Description
Device_ID	int	Unique identifier for the device
Device_Name	string	Name of the device
Device_Type	string	Type/category of the device
RFID_Tag	RFID_Tag	Associated RFID tag

Methods:

- getDeviceInfo() -> string: Returns the device information as a string.
- assignRFIDTag(tag: RFID_Tag): Associates an RFID tag with the device.

Explanation:

- **getDeviceInfo()**: This method returns a string containing the device's details (name, type, ID). It does not require any parameters.
- **assignRFIDTag(tag)**: This method links an RFID tag to the device. It accepts an instance of the RFID_Tag class as its parameter.

3. ESP32_Circuit Class

- **Purpose**: Represents the ESP32 microcontroller, which interacts with the device and performs actions such as controlling LEDs and buzzers.

Attributes	Type	Description
Circuit_ID	int	Unique identifier for the ESP32
Device	Device	The device controlled by this ESP32
LED_Status	boolean	Status of the LED (on/off)
Buzzer_Status	boolean	Status of the buzzer (on/off)

Methods:

- controlLED(status: boolean): Turns the LED on or off.
- controlBuzzer(status: boolean): Turns the buzzer on or off.
- sendDeviceData() -> string: Sends the device data to the database.

Explanation:

- **controlLED(status)**: This method controls the LED. It takes a boolean parameter (status) that represents whether to turn the LED on (true) or off (false).

- **controlBuzzer(status)**: Similar to the controlLED method, this method controls the buzzer. It also takes a boolean (status) as input to activate or deactivate the buzzer.
- **sendDeviceData()**: By using this technique, the database receives and stores the device's current data. It returns a confirmation message (as a string) regarding the data transfer and does not require any inputs.

4. Database Class

- **Purpose**: Represents the system's database, responsible for storing device data, including information obtained from RFID scans.

Attributes	**Type**	**Description**
DB_ID	int	Unique identifier for the database
Device_Data	string	Data stored in the database
Timestamp	DateTime	Time of data entry

Methods:

- storeData(data: string, timestamp: DateTime): Stores the provided data in the database.
- checkDataExists(device_id: int) -> boolean: Checks if the data for a device is already in the database.

Explanation:

- **storeData(data,timestamp)**:By using this technique, the database receives and stores the dev ice's current data.It returns a confirmation message (as a string) regarding the data transfer an d does not require any inputs.
- **checkDataExists(device_id)**: This technique determines if the database already contains the data for a specific device. It returns a boolean (yes if the data exists, false otherwise) after receiving the device_id as input.

5. Authentication Class

- **Purpose**: Handles the authentication process when an RFID tag is scanned.

Attributes	Type	Description
Auth_ID	int	Unique identifier for authentication
RFID_Tag	RFID_Tag	The RFID tag used for authentication
Device	Device	The device being authenticated
Timestamp	DateTime	Time of authentication event

Methods:

- authenticate() -> boolean: Authenticates the RFID tag and device.
- logAuthEvent(): Logs the authentication event in the database.

Explanation:

- **authenticate()**: Using this method, you may verify that the RFID tag is legitimate and linked to the right device. If the authentication process is successful, it returns true; if not, it returns false.
- **logAuthEvent()**: The authentication event (RFID tag, device, and timestamp) is logged in the database using this method. It captures the pertinent data but does not require any settings.

4.3 Test Cases

1. Testing Techniques

the following testing methodologies have been used:

- **Testing Units:**
  focuses on specific parts, like the ESP32 circuit functions, database interface, and RFID scan ning. makes sure every class and method functions as intended when used alone.

Integrity Checking:

- tests the communication between several parts, including the ESP32 and the database and the RFIDreader.confirms that the modules function properly when combined with the entire syste m.

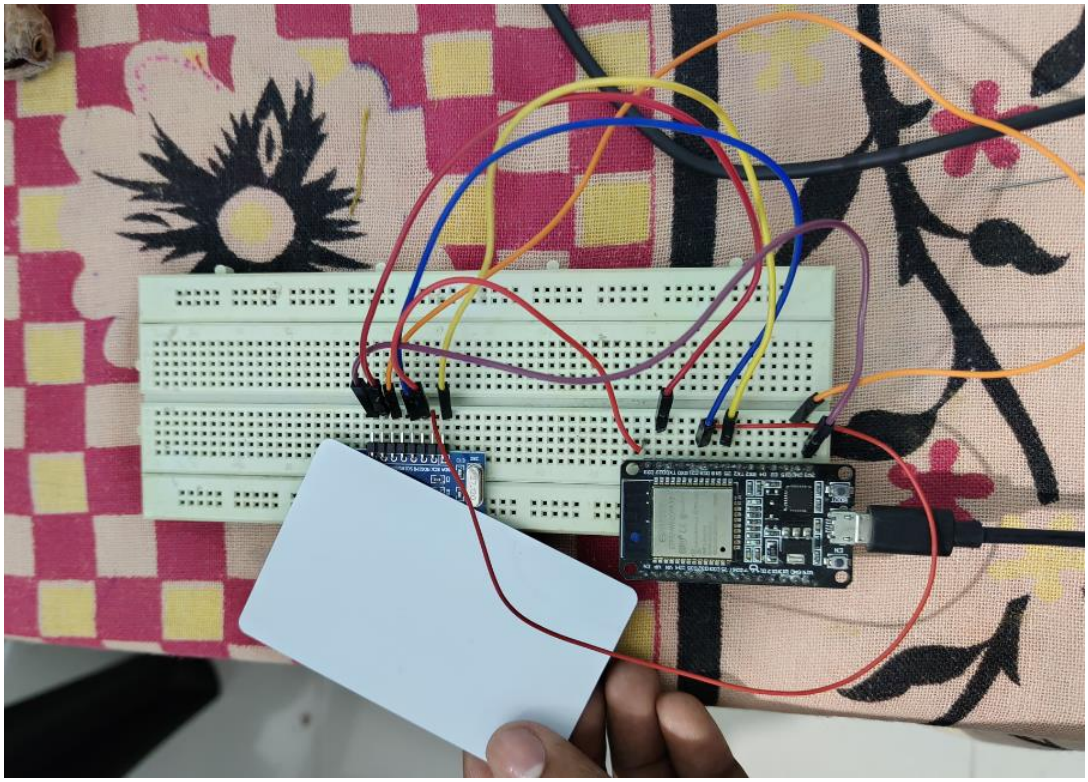
Testing the System:

- assesses the system's overall performance, taking into account both its software and hardware guarantees that, in a variety of circumstances, the IoT system operates as intended.

Testing for user acceptance (UAT):

- makes ensuring the system satisfies the needs of the end user.
  To mimic actual user interactions with the system, tests are carried out.

Snapshots



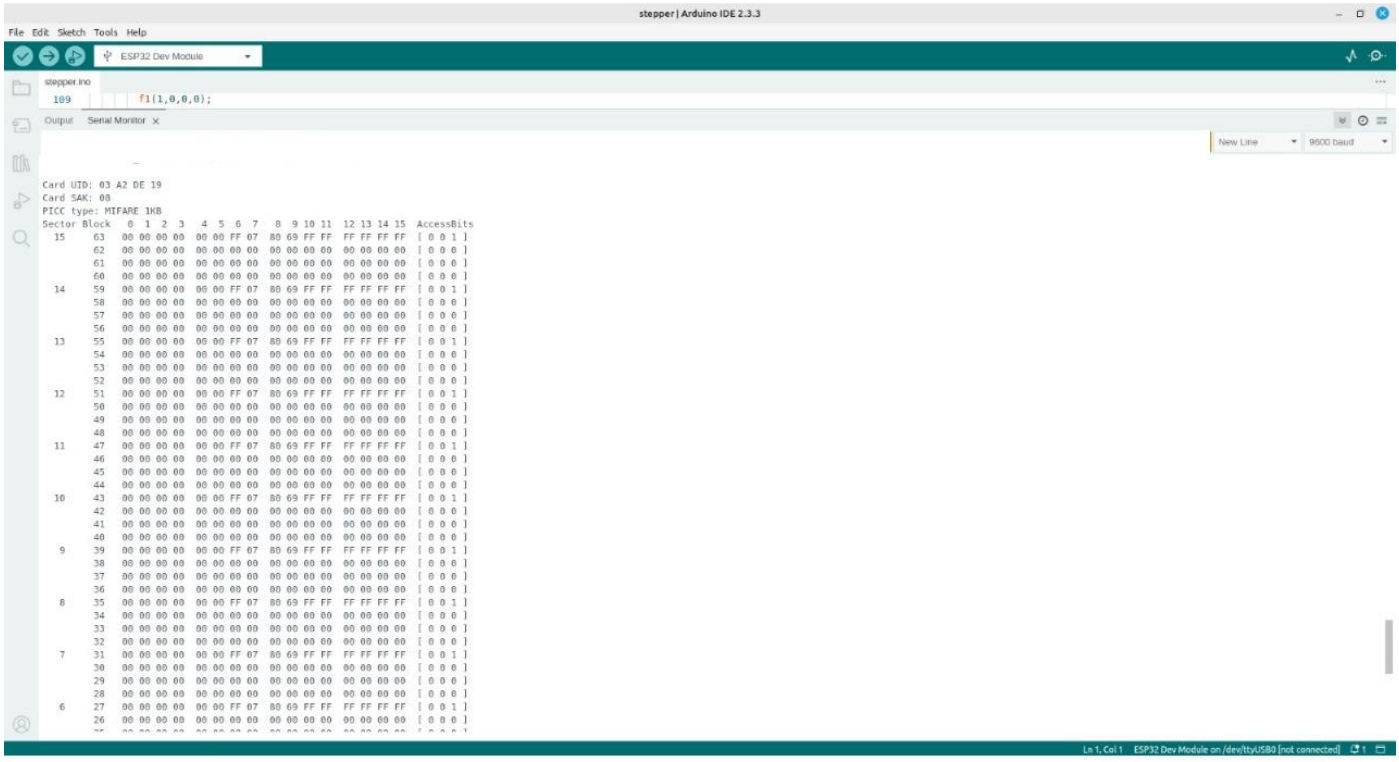
Test case 1

This snapshot displays the hardware setup with an ESP32 microcontroller, an RFID reader, and an RFID tag being scanned. The RFID tag is used to identify a device, which is then processed by the ESP32. The wiring connects the RFID reader to the ESP32 for communication, and the ESP32 is connected to a power source through USB.

Test 01: Successful RFID Tag Scan

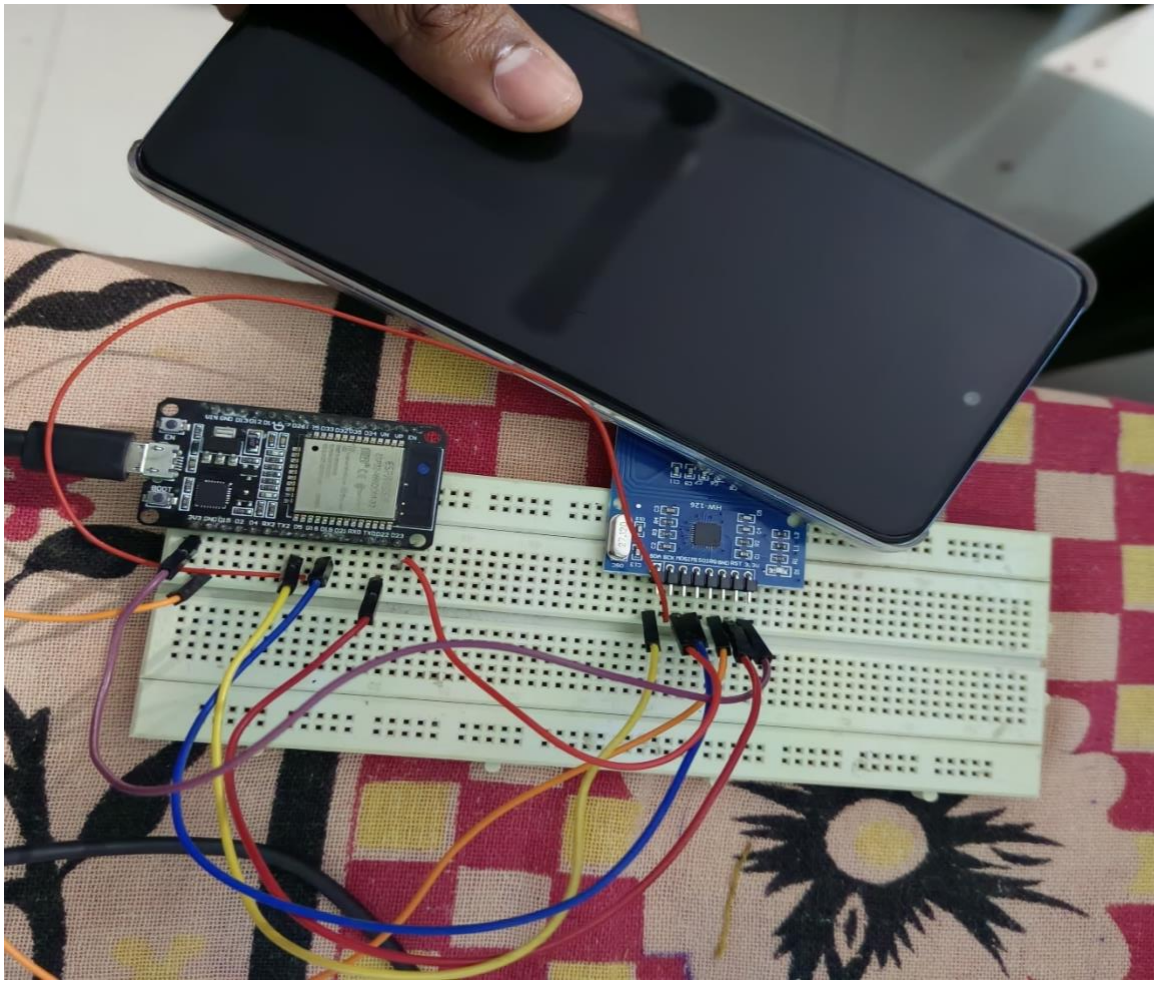
- **Input**: Valid RFID tag is placed near the reader.
- **Expected Output**: The system reads the RFID tag and successfully identifies the tag ID, sending it to the ESP32.
- **Actual Output**: The tag ID is displayed on the connected serial monitor, and the LED blinks.
- **Status**: PASS

Output



Output

Test case 2

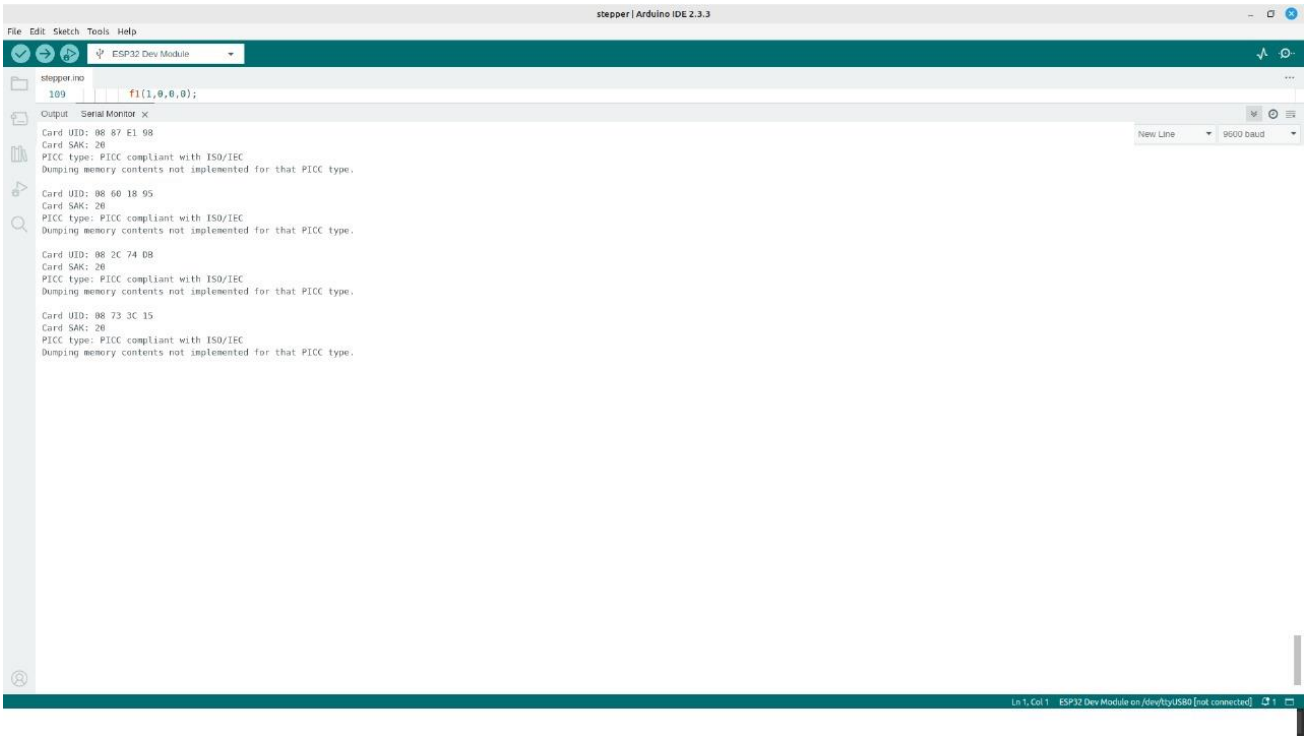


Test case 2

Test 02: Successful device Scan

- **Input**: Valid device tag is placed near the reader.
- **Expected Output**: The system reads the deice tag and successfully identifies the tag ID, sending it to the ESP32.
- **Actual Output**: The tag ID is displayed on the connected serial monitor, and the LED blinks.
- **Status**: PASS

Output



Output

Code for implementation

Signal	MFRC522 Reader/PCD Pin	Arduino Uno/101 Pin	Arduino Mega Pin	Arduino Nano v3 Pin	Arduino Leonardo/Micro Pin	Arduino Pro Micro Pin
RST/Reset	RST	9	5	D9	RESET/ICSP-5	RST
SPI SS	SDA(SS)	10	53	D10	10	10
SPI MOSI	MOSI	11 / ICSP-4	51	D11	ICSP-4	16
SPI MISO	MISO	12 / ICSP-1	50	D12	ICSP-1	14
SPI SCK	SCK	13 / ICSP-3	52	D13	ICSP-3	15

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN         22          // Configurable, see typical pin layout above
#define SS_PIN          5         // Configurable, see typical pin layout above
MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance
void setup() {
  pinMode(2, OUTPUT);
        Serial.begin(9600);               // Initialize serial communications with the PC
        while (!Serial);                  // Do nothing if no serial port is opened (added for Arduinos based on
ATMEGA32U4)
        SPI.begin();                            // Init SPI bus
        mfrc522.PCD_Init();                     // Init MFRC522
        delay(4);                               // Optional delay. Some board do need more time after init to
be ready, see Readme
        mfrc522.PCD_DumpVersionToSerial(); // Show details of PCD - MFRC522 Card Reader details
        Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
}
void loop() {
// Reset the loop if no new card present on the sensor/reader. This saves the entire process when idle.
        if ( ! mfrc522.PICC_IsNewCardPresent()) {
    digitalWrite(2, LOW);
                return;  }
// Select one of the cards
        if ( ! mfrc522.PICC_ReadCardSerial()) {
                digitalWrite(2, LOW);
    return;         }
  digitalWrite(2, HIGH);    // Dump debug info about the card;
 PICC_HaltA() is automatically called
        mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
  }
```

Chapter – 5 Conclusion

5.1 Learnings and Achievements

The project, titled "Device Identity Mechanism in Smart IoT-Based Building," provided valuable practical experience in creating an IoT-based system for device identification using the ESP32 microcontroller and RFID technology. By integrating various hardware components, such as RFID readers, tags, LEDs, buzzers, and a database for device information storage, I gained the following insights:

- **IoT System Development**: I learned how to design and build an efficient, scalable IoT system capable of uniquely identifying devices.
- **Microcontroller Programming**: The project enhanced my knowledge of programming microcontrollers like ESP32 and interfacing them with external components through the Arduino IDE.
- **Database Integration**: I developed skills in managing device information, which involved transmitting data from the ESP32 to a database for secure storage and validation.
- **Circuit Design and Simulation**: By using Proteus software, I learned how to simulate circuits before physically constructing the system, which helped eliminate design errors.
- **UML Diagrams and Documentation**: The project improved my ability to document the system using UML diagrams, such as sequence diagrams, class diagrams, and E-R diagrams, which provided a clear representation of the workflow and relationships between components.

5.2 Project Limitations

Although the project successfully implemented the core features, it faced a few constraints:

- **RFID Reader Range**: The RFID reader had a limited range, which reduced its ability to identify devices over larger distances.
- **Real-time Data Processing**: While the system could store and verify device data, there is potential for improvement in the speed of real-time data processing and response.

- **Database Scalability**: The current database implementation is basic and may struggle to handle a significant increase in the number of devices, affecting performance if not optimized or migrated to a more robust platform.
- **Network Reliance**: The system depends on a stable network connection for data transfer between the ESP32 and the database. Any network failure can hinder system operations.

5.3 Future Enhancements

Several potential enhancements can extend the project's capabilities in the future:

- **Extended RFID Range**: Implementing long-range RFID readers or using alternative technologies, such as NFC, could improve identification accuracy and range.
- **Mobile App Integration**: Developing a mobile application for real-time monitoring and management of device identities could significantly enhance user convenience.
- **Cloud-based Database**: Migrating the database to a cloud platform would improve scalability, data security, and allow for remote control and monitoring.
- **Real-time Alerts**: Adding features like real-time alerts through SMS or email for new device registrations or unauthorized access attempts would improve system security.
- **Machine Learning for Security**: Incorporating machine learning algorithms could detect unusual device behaviour or access attempts, enhancing the overall security and reliability of the system.

REFRENCES

Books

1. Alasdair Allan, *Programming the ESP32: Getting Started with the Espressif ESP32 Development Board*, O'Reilly Media, 2019.
2. John S. Huggins, *RFID Handbook: Applications, Technology, Security, and Privacy*, Springer, 2017.
3. Raj Kamal, *Internet of Things: Architecture and Design Principles*, McGraw Hill Education, 2020.

Research Papers

1. S. Srivastava, A. Sinha, "A Study on IoT Device Identity Management and Security," *International Journal of Computer Applications*, Vol. 180, Issue 47, June 2023.
2. M. Gupta, V. Aggarwal, "Enhancing RFID Authentication Mechanisms for IoT-Based Systems," *IEEE Conference on Internet of Things*, Vol. 10, Issue 2, March 2022.

Websites

1. Arduino.cc, "ESP32 – Getting Started with the ESP32 Board," https://www.arduino.cc, Accessed September 2024.
2. Espressif Systems, "ESP32 Technical Reference Manual," https://www.espressif.com, Accessed August 2024.
3. Tutorials Point, "IoT Device Identity Mechanisms," https://www.tutorialspoint.com, Accessed July 2024.
4. CircuitDigest, "Interfacing RFID with ESP32 for IoT Applications," https://circuitdigest.com, Accessed July 2024.
5. Instructables, "How to Use RFID with ESP32 for Smart Building Projects," https://www.instructables.com, Accessed August 2024.
6. Random Nerd Tutorials, "ESP32 RFID Reader with Database Integration," https://randomnerdtutorials.com, Accessed September 2024.
7. ElectronicsHub, "ESP32 with RFID and Database Connectivity," https://www.electronicshub.org, Accessed October 2024.

8. Hackster.io, "Smart IoT-Based Building with ESP32 and RFID," https://www.hackster.io, Accessed October 2024.
9. Microcontrollers Lab, "ESP32 RFID Tutorial with Circuit Diagram and Code," https://microcontrollerslab.com, Accessed October 2024.
10. IoT Design Pro, "RFID with ESP32 in IoT-Based Applications," https://iotdesignpro.com, Accessed October 2024.
11. Maker Pro, "IoT Smart Building Project Using ESP32 and RFID," https://maker.pro, Accessed October 2024.
12. Espressif Systems, "ESP32 RFID Integration for IoT Projects," https://www.espressif.com, Accessed October 2024.
13. Tinkercad Blog, "Prototyping IoT Projects with ESP32 and RFID," https://blog.tinkercad.com, Accessed October 2024.

Appendix: Abbreviation List

1. **IoT** – Internet of Things
2. **RFID** – Radio Frequency Identification
3. **ESP32** – Espressif Systems Microcontroller
4. **LED** – Light Emitting Diode
5. **LCD** – Liquid Crystal Display
6. **IDE** – Integrated Development Environment
7. **UML** – Unified Modelling Language
8. **DB** – Database