



University of Bahrain
College of Information Technology
Department of Computer Engineering

IoT SMART LOCK WITH AWS INTEGRATION

Prepared by

Najim Abdulkarem Alfutini	202003215
Waleed Saleh Ali	202006448
Maged Hussain Aljashoobi	202004484

For

ITCE 497

Senior Project

Academic Year 2023-2024-Semester 2

Project Supervisor: Dr. Ali Hasan Sayed

Date of Submission: 14/5/2024

Abstract

IoT-based cloud smart lock, that seamlessly combines security and convenience with the integration of Amazon Web Services (AWS). The system leverages the power of AWS such as including AWS IoT Core, DynamoDB, and Lambda to provide a secure and scalable cloud platform for managing IoT devices. This smart lock system achieves a robust and secure access control solution, it was made using an ESP32 module for the lock operations and a fingerprint sensor, keypad, and RFID for unlocking the lock. In addition, a dedicated webpage facilitates admin permissions and management. The project combines the convenience of cloud connectivity and centralized management with the reliability and scalability provided by AWS which ensures a seamless and efficient user experience while maintaining the highest standards of security.

Acknowledgments

We would like to express our sincere gratitude to our project supervisor, Dr. Ali Hasan Sayed Ali Ebrahim, for his invaluable guidance and support throughout the project. His expertise, advice, and feedback have played a pivotal role in shaping our project and ensuring its successful completion.

We would also like to express our appreciation to Dr. Mohamed A. Almeer, Mr. Ali Al-Junaid, Mr. Ahmed Alsaigh, faculty members, and staff of the University's Computer Engineering Department for providing a nurturing academic environment, excellent facilities, and essential resources that have supported our project's successful completion.

Additionally, we would like to thank our friends and family for their unwavering belief in our abilities and their valuable feedback. Their support and encouragement have a major impact on our academic journey.

We sincerely appreciate the opportunity to work on this project and the support received from all individuals involved. Their collective contributions have been integral to the project's achievements, and we are grateful for their guidance, expertise, and unwavering support.

Table of Contents

ABSTRACT	II
ACKNOWLEDGMENTS.....	III
LIST OF FIGURES.....	VI
CHAPTER 1 INTRODUCTION.....	7
1.1 PROBLEM STATEMENT	7
1.2 PROJECT OBJECTIVES	8
1.3 RELEVANCE/SIGNIFICANCE OF THE PROJECT.....	9
1.4 REPORT OUTLINE	9
CHAPTER 2 BACKGROUND AND LITERATURE REVIEW	11
CHAPTER 3 SYSTEM DESIGN	12
3.1 HARDWARE COMPONENTS	12
3.1.1 ESP32	12
3.1.2 Solenoid Lock.....	13
3.1.3 RFID- RC522.....	14
3.1.4 LCD	14
3.1.5 Relay.....	15
3.1.6 Power Supply Model	16
3.1.7 Fingerprint Sensor	16
3.2 SCHEMATIC DIAGRAM.....	17
3.3 SOFTWARE COMPONENTS	17
3.3.1 IoT Core	18
3.3.2 Lambda	18
3.3.3 DynamoDB	18
3.3.4 S3 (Simple Storage Service)	18
3.3.5 API Gateway & Rest API	18
3.3.6 CloudFront.....	19
3.3.7 Certificate Manager	19
3.3.8 Route 53	19
3.4 ADMIN INTERFACE DESIGN	20
CHAPTER 4 SYSTEM IMPLEMENTATION	21

4.1 SOFTWARE SETUP AND DEVELOPMENT.....	21
4.1.1 <i>Platforms</i>	21
4.1.2 <i>Setup a Component Configuration & Libraries</i>	22
4.1.3 <i>Initialize Amazon DynamoDB</i>	22
4.1.4 <i>AWS IoT Core Configuration & secret.h</i>	23
4.1.5 <i>AWS Hosting & Admin Interface Configuration</i>	23
4.2 TOKEN AND ACCESS VALIDATION PROCESSING.....	23
4.2.1 <i>User Registration</i>	24
4.2.2 <i>Log-in Authentication</i>	24
4.3 ESP32 CONNECTION TO AWS AND WIFI	25
4.4 FINGERPRINT PROCESSING	25
4.5 RFID PROCESSING	26
4.6 AWS LAMBDA FUNCTIONS.....	26
4.6.1 <i>(UsersData) & ESP32 System</i>	27
4.6.2 <i>(AdminUI) lambda</i>	28
4.7 ADMIN INTERFACE & REACT APPLICATION	29
CHAPTER 5 TESTING AND RESULTS	30
5.1 OPENING THE LOCK FROM THE WEB PAGE BY THE ADMIN	30
5.2 REGISTER USER.....	31
5.3 OPENING THE LOCK USING FINGERPRINT	35
5.4 OPENING THE LOCK USING RFID AND PASSWORD	36
5.5 TESTING TOKEN VALIDATION.....	37
CHAPTER 6 CONCLUSION AND FUTURE WORK.....	41
REFERENCES	43

List of Figures

Figure 1 Adafruit huzzah32 esp32 feather. [3].....	13
Figure 2 Solenoid Lock 9Volt	13
Figure 3 RFID-RC522.....	14
Figure 4 I2C for Liquid Crystal Displays.....	15
Figure 5 LCD 16×2	15
Figure 6 5V Four-Channel Relay Module	15
Figure 7 Jbtek Breadboard Power Supply Module 3.3V/5V for Arduino Board Solderless Breadboard ..	16
Figure 8 Adafruit Optical Fingerprint Sensor. [6]	16
Figure 9 Hardware Setup - Main Circuit Diagram & Interfacings	17
Figure 10 Admin Interface design- serverless architecture.	20
Figure 11 Integrated Smart Lock System Architecture Diagram	21
Figure 12 UsersData and Esp32 Dataflow	27
Figure 13 AdminUI lambda Function & REST API	28
Figure 14 Admin Interface (Open Lock)	30
Figure 15 LCD welcoming Admin after opening the lock.....	30
Figure 16 Admin interface (Add New User).....	31
Figure 17 Admin Interface New User Information	31
Figure 18 Admin Interface (Add Finger ID 2)	32
Figure 19 Registering Finger print.	32
Figure 20 Admin Interface (Add RFID).....	33
Figure 21 Registering RFID Tag.....	33
Figure 22 Admin Interface (Add Password)	34
Figure 23 Registering a New Password	34
Figure 24 Admin Interface (Add Token).....	35
Figure 25 Users Table	35
Figure 26 Opening the Lock Using Fingerprint.....	36
Figure 27 Opening the Lock using RFID and Password.....	37
Figure 28 Users For Token Testing.....	37
Figure 29 User "Adnan", Daily Token	38
Figure 30 User "Mohammed" To This Date and Time Token.....	38
Figure 31 User "Luai" Between Two Dates Token	39

Chapter 1

Introduction

In a world where connectivity and convenience are essential, the implementation of Internet of Things (IoT) technology has transformed many parts of our lives. One example is access control systems, where traditional locking options frequently fail in terms of security, management, and user satisfaction. To solve these drawbacks, this report introduces an IoT-based cloud-based smart lock solution that seamlessly integrates security and convenience using Amazon Web Services (AWS).

The IoT-based cloud smart lock system takes advantage of AWS technologies such as AWS IoT Core, DynamoDB, S3, API Gate Way, and Lambda to provide a secure and scalable cloud platform for managing IoT devices. This platform serves as the smart lock system's backbone, ensuring that the lock and the cloud communicate reliably and efficiently. It supports complex functionalities such as remote lock operations, access management, entry logs, and real-time monitoring, all while maintaining strong security standards. Moreover, to improve the user experience, a separate webpage has been created to enable admin permissions and management. Authorized users (admin) can use this user-friendly interface to remotely control and monitor their locks, modify access rights, manage users, and receive real-time updates. The seamless integration of cloud connectivity and centralized management means that users may easily interact with the smart lock system, resulting in a genuinely convenient and efficient access control solution.

1.1 Problem Statement

In the current smart lock landscape, where several other solutions exist, our project aims to design and implement an IoT-based cloud smart lock system that provides distinct advantages while also addressing the limitations of existing solutions. While other smart locks have certain functions, they frequently suffer from common issues that limit their effectiveness and user experience. Our project will prioritize security, scalability, and user convenience by focusing on the following key subproblems:

Enhanced Security Features: Many existing smart locks have security flaws or use outdated authentication methods. Our smart lock system will include strong security features like biometric (fingerprint) and RFID authentication to ensure secure access control and prevent unauthorized entry attempts.

Seamless Integration with AWS: While some smart locks may have integration capabilities, our system will use the power of AWS IoT Core, DynamoDB, and Lambda services to ensure that the smart lock and cloud platform communicate seamlessly. This integration will allow for real-time monitoring, efficient data management, and scalability as the system grows.

Advanced Access Control Management: Existing smart locks may not have comprehensive access control management, making it difficult to grant temporary access, revoke access remotely, or provide detailed permissions. Our smart lock system will include a user-friendly admin page that simplifies access permission management using tokens by allowing for easy user registration, access revocation, and activity logs.

1.2 Project Objectives

The main objective of the project is to create an IoT-based cloud smart lock system that seamlessly integrates security and convenience using Amazon Web Services (AWS). To accomplish that goal, the following objectives have been established to steer the project's progress:

1. Develop a cloud smart lock system based on the Internet of Things (IoT) that works with Amazon Web Services (AWS) to offer strong security and sophisticated access control features.
2. Using AWS services, develop a scalable and dependable cloud platform that will enable smooth communication between the cloud and the smart lock devices.
3. Use a variety of authentication techniques, such as RFID technology, a keypad, and a fingerprint sensor, to give users flexibility and convenience when opening smart locks.
4. Create a user-friendly web-based admin interface for convenient and centralized management of the smart lock system. This interface should allow authorized users (admins) to remotely control and monitor locks and manage access permissions.
5. Ensure the highest levels of security by incorporating encryption protocols and secure data transmission.

6. Conduct testing and evaluation of the system's performance, reliability, and security to validate its effectiveness and identify any potential flaws or opportunities for improvement.
7. Look into the possibility of future enhancements and developments, to expand the functionality and value of the smart lock system.

1.3 Relevance/Significance of the project

The significance of this project arises from its capacity to break through the drawbacks of the present lock systems. The IoT-based cloud smart lock system combines better security, convenience, scalability, and centralized management by capitalizing on the potential of IoT and cloud computing. It not only overcomes the flaws of traditional lock systems, but it also creates new opportunities for smart home security, business applications, and a variety of industries where access control is critical.

1.4 Report Outline

- Chapter 1: Introduction

This chapter explains an introduction to the project by providing an overview of the problem statement, project objectives, project relevance/significance, and project limitations. The subsequent sections of the report will discuss these aspects as follows:

- Chapter 2: Background and Literature Review

A comprehensive review of previous research articles and studies relevant to IoT smart lock systems, access control mechanisms, biometric authentication, RFID technology, and cloud computing.

- Chapter 3: System Design

Explains the techniques and strategies employed in designing our IoT smart lock system. We will discuss the hardware and software components, schematic diagrams, and the design of the web page.

- Chapter 4: System Implementation

This chapter will focus on the implementation details of the smart lock system. Outlines the software implementation including the packages, tools, and platforms. Additionally, it discusses the hardware components, particularly the integration with ESP32. process.

- Chapter 5: Testing and Results

Presents the testing procedures performed on our IoT smart lock system. providing pictures and descriptions of the tested scenarios, along with the results in terms of security, scalability, connectivity, authentication, and performance.

- Chapter 6: Conclusion and Future Work

A conclusion summarizing the overall implementation process of the IoT smart lock system. Discussing the objectives achieved and limitations. Additionally, mentioning future potential enhancements and recommendations for future work.

Chapter 2

Background and Literature Review

Smart locks and access control systems have advanced significantly over time. Researchers and practitioners have tried a variety of approaches to improve security while maintaining convenience. Early smart locks relied heavily on keypad-based entry systems, which faced issues such as vulnerability, limited authentication methods, and scalability. However, the introduction of biometric authentication, such as fingerprint recognition, transformed smart locks by providing greater security and user convenience. With the advent of the Internet of Things (IoT), smart locks began to connect to cloud platforms, enabling centralized management, remote access, and scalability. In the present era, researchers are investigating various authentication methods for smart locks. These include fingerprint recognition, RFID (Radio Frequency Identification), and mobile apps. Each method has strengths and weaknesses. For example, biometric sensors provide dependable and unique identification, but privacy concerns remain. RFID tags and cards provide seamless access, but threats must be handled. Mobile apps allow for remote control and monitoring, but battery life optimization remains vital.

One important component of smart lock systems is IoT security. Researchers prioritize robust encryption (ECC and AES) and secure communication protocols (like MQTT and HTTPS) for Internet of Things devices. It's critical to build trust between devices and cloud services. A major part of edge computing is played by edge devices such as the ESP32, which process data locally before sending it to the cloud. Critical analysis of existing literature reveals conflicting viewpoints. Some focus on usability, while others prioritize security. Researchers discuss the trade-offs between convenience and robust security. Arguments for and against different smart lock designs, security mechanisms [2], and cloud architectures are everywhere. Viewing relevant studies and experiments allows us to draw conclusions about gaps in current research and areas where our project can help.

Chapter 3

System Design

The system has hardware and software components, the hardware components are the main components of the system design and the functionality of the system. The system has different hardware components and sensors. The microcontroller that is used in the system is ESP32, and the input devices are RFID, fingerprint sensor, and keypad. The lock that is used in the system is a solenoid lock. In addition, LCD I2C is used to display information to the users.

For the software component, the AWS is used to process the user input information and validate the user inputs, The validation process is done by forwarding the data from the esp32 to the cloud by using the MQTT protocol. Then the IoT Core service will receive the data and send it to the Lambda function by using the IoT rule.

3.1 Hardware Components

The Hardware components are chosen based on their compatibility and functionality to ensure that the system is consistent and has a higher performance. The following subsections will provide an overview of each component, that gives features and how these components are used on the system.

3.1.1 ESP32

The ESP32 is a powerful microcontroller module used in several IoT applications. It has a built-in USB-to-Serial converter, automatic bootloader reset, and a Lithium Ion/Polymer charger. In our system, the esp32 is the main component. It's responsible for processing data that it receives from the users and communicating with the AWS IoT core by using the MQTT protocol. The ESP that we use in our project is 'Adafruit Huzzah32 - ESP32 Feather'. This ESP has many features, for example it has a dual-core Tensilica LX6 microcontroller with 600 DMIPS and runs on 240 MHz. In addition, it has a 4MByte flash and integrated dual-mode Bluetooth and Wi-Fi module [3]. And the esp32 is powered by a Lipoly battery 3.7V.

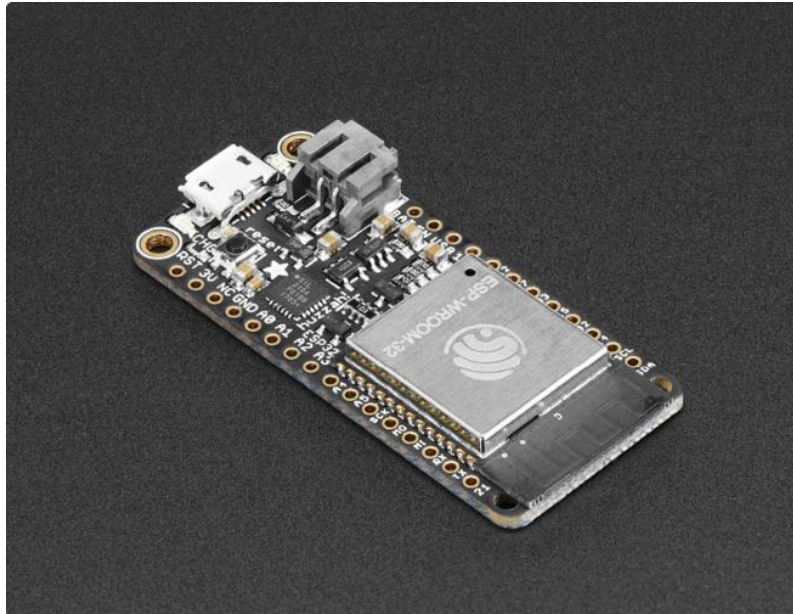


Figure 1 Adafruit huzzah32 esp32 feather. [3]

3.1.2 Solenoid Lock

The solenoid lock is an electromagnetic lock and it is used to secure access. It is made of a coil, wire, and metal plate. When the current is passing through the solenoid it generates a magnetic field. In the system, the lock provides physical security and it must open after the system authorizes the user. The solenoid lock is powered by a 9-12V Sealed Lead Acid battery and it draws 650mA at 12V, and 500 mA at 9V when activated. [4]



Figure 2 Solenoid Lock 9Volt

3.1.3 RFID- RC522

RFID stands for (Radio Frequency Identification) It is a technology that uses electromagnetic fields for communications, the communication and sending of data must be within a limited distance of about 3cm or less. The Reader RC522 sends radio waves, and when the card or the tag comes in the range it sends the data, and the data is captured by the reader. The RC522 operates on 2.5-3.3V, and the operating frequency is 13.56MHz. So it can only read the cards with a frequency of 13.56MHz. The tags are 1KB of memory and divided into 16 sectors, each sector is divided into 4 blocks and each block can store 16 bytes of data. The Uid (Unique Identifier) is 8 bytes and is stored in the sector 0 and block 0 [5]. In our System, RFID is used for authentication and identification, and each user must have one card or tag.



Figure 3 RFID-RC522

3.1.4 LCD

The LCD stands for (Liquid Crystal Display) which is used to display text and information. In the system, the I2C (Inter-Integrated circuit) is used to manage the communication between the chip and the microcontroller with the two-wire serial protocol. The LCD is used as a user interface that displays the system state, lock state, and error messages.

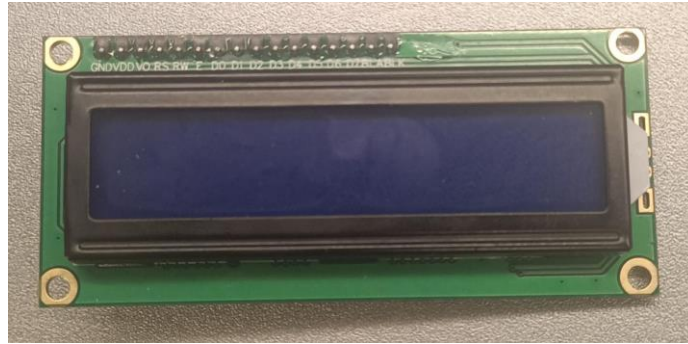


Figure 5 LCD 16x2

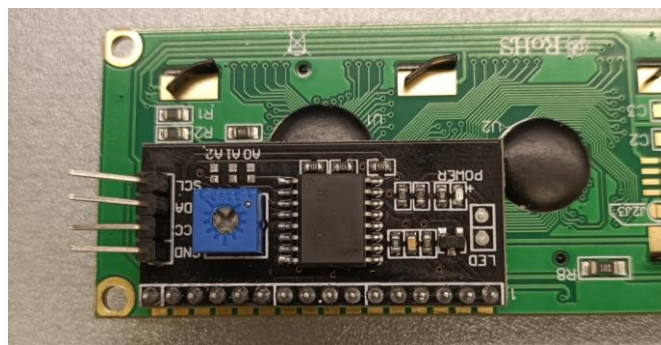


Figure 4 I2C for Liquid Crystal Displays

3.1.5 Relay

The relay is an electronic device that operates like a switch that allows a low-power signal to control a high-power circuit. In the system, the relay is used to control the power of the solenoid lock. The relay that is used in the system is a low-level trigger, and it needs 5V power. And the maximum output is 30V/10A DC, 250V/10A AC.

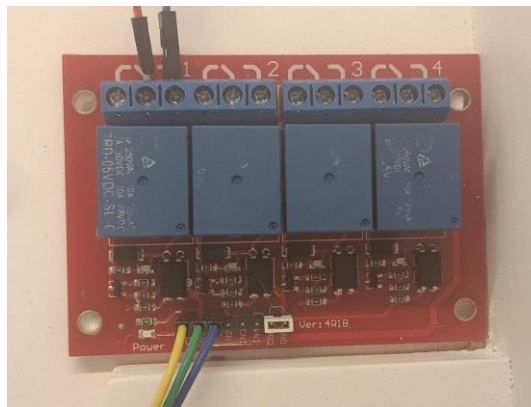


Figure 6 5V Four-Channel Relay Module

3.1.6 Power Supply Model

The power supply MB102 is used in the system to provide electrical power to the system components. It applies to the MB102 breadboard, and it converts the power from the input source battery 9V or USB into 3.3V or 5V. In our system, the power supply model ensures the power of the fingerprint sensor, relay, and LCD (each 5V) is stable.

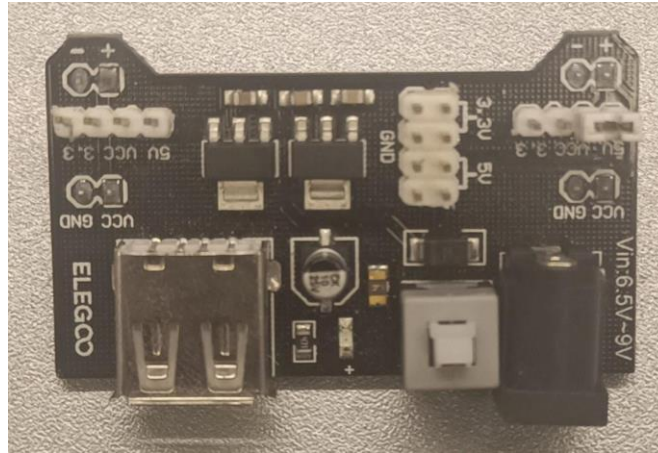


Figure 7 JBtek Breadboard Power Supply Module 3.3V/5V for Arduino Board Solderless Breadboard

3.1.7 Fingerprint Sensor

The fingerprint sensor that is used in the system is an optical scanner sensor is one method for the user to open the lock. The sensor will capture and verify the user's fingerprint. With this method, it ensures secure authentication to open the lock. this sensor can store up to 127 user fingerprints, and the sensor needs 3.6-6.0VDC. [6]

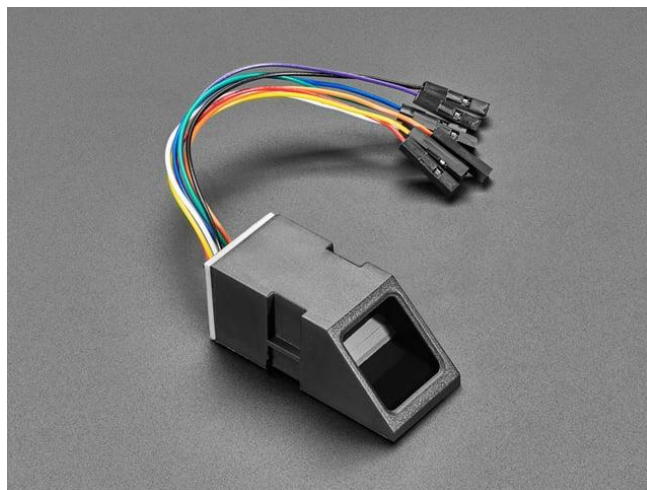


Figure 8 Adafruit Optical Fingerprint Sensor. [6]

3.2 Schematic Diagram

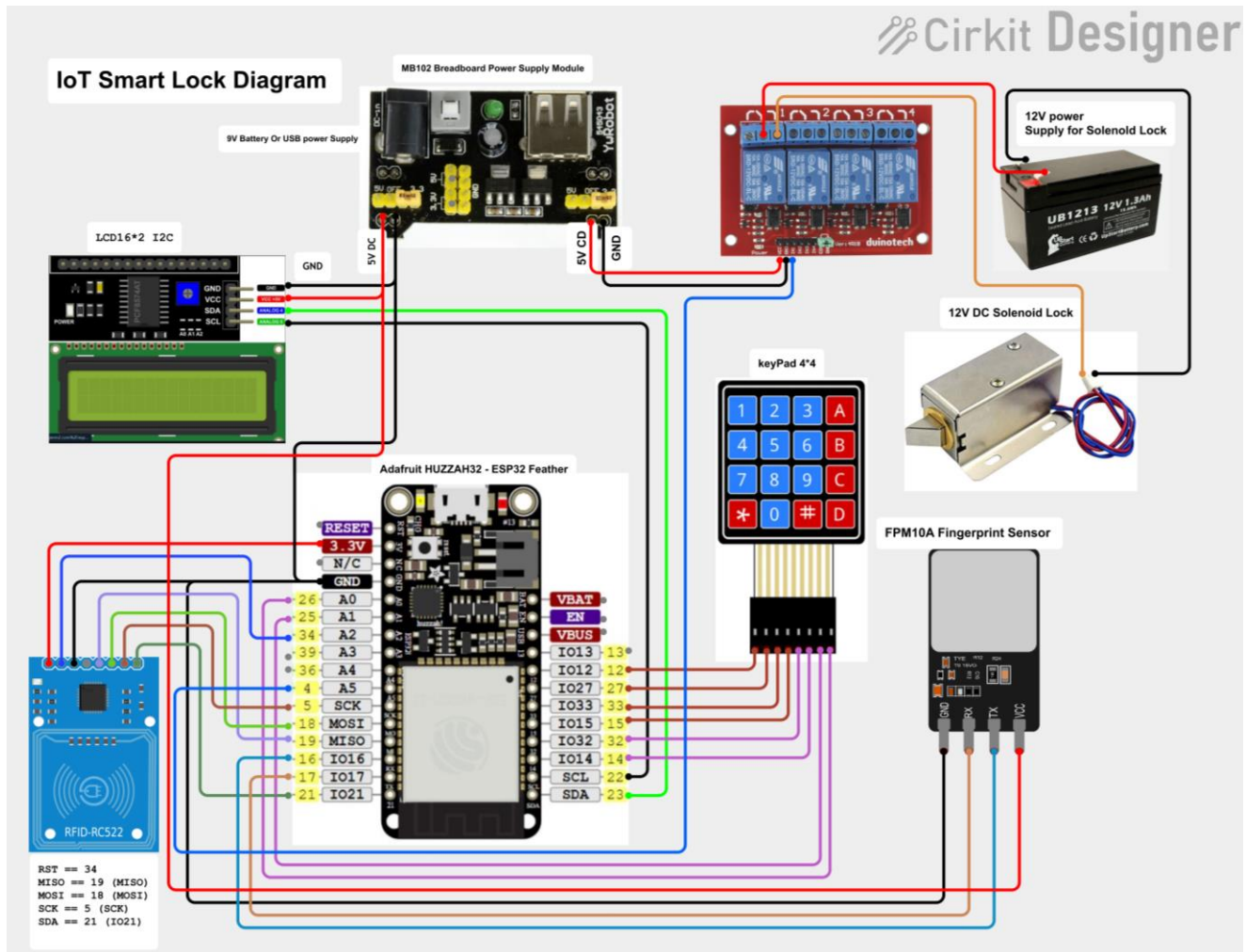


Figure 9 Hardware Setup - Main Circuit Diagram & Interfacings

The above schematic diagram (Figure 1) illustrates the connection between the hardware components and the esp32 microcontroller. It provides an overall hardware setup to understand the system architecture. In addition, it shows different communication interface connections: GPIO pins, I2C, SPI, and UART. Furthermore, it highlights the power supply connection for all the hardware components and ensures each has the necessary power.

3.3 Software Components

The following sub-sections illustrate the AWS services that are used in the system. The system used different AWS services: IoT, computing, networking, database, and storage services.

3.3.1 IoT Core

The IoT Core service is a very useful service with IoT applications. It allows you to connect the IoT device easily and securely to the cloud with low latency and high throughput. In addition, it gives the user the chance to choose the preferred communication protocol between MQTT, HTTPS, and LoRaWAN. In our system, this service is used with MQTT protocol to send the user data to the AWS and take action based on the validation process. [7]

3.3.2 Lambda

The lambda service is a serverless computer service that runs code without thinking about managing the servers or clusters. The key features of this service are its automatic scaling and its Pay-as-you-go pricing, which means you only pay for what you use. The lambda function can be triggered by over 200 AWS services and SaaS (Software As A service) applications. It is used in the system to run the Python code that handles the validation of user data and access the database tables in the DynamoDB [1].

3.3.3 DynamoDB

DynamoDB is a Non-relational database and NoSQL database service that provides a fast and scalable database, so the tables can be scaled up or down based on the requirement. And it offers encryption at rest. It allows the user to create full backups of all the tables, and it gives the option to delete the expired items for the tables to help reduce the costs of storing data. This database service has high availability that automatically replicates the data in multiple AZs (Availability Zones) [8].

3.3.4 S3 (Simple Storage Service)

Amazon Simple Storage Service (Amazon S3) is a storage service that provides object storage scalability, high security, data availability, and high performance. S3 can store different types of objects, such as websites, applications, comprehensive data analytics, mobile applications, backups, and restores. Since there are many objects, S3 provides management services that help optimize access to objects and provide availability to meet customer's requirements [9].

3.3.5 API Gateway & Rest API

Amazon API Gateway is a fully managed service that Amazon Web Services (AWS) provides. It helps developers build, deploy, and secure APIs at any scale. It also serves as a

gateway for applications to access website data or applications from server services. API Gateway supports RESTful and WebSocket APIs, enabling real-time communication applications. However, what enhances trust is its efficient traffic management, which handles so many tasks, messages, and calls [10].

REST (Representational et al.) is an architectural technique for distributed hypermedia systems. It is a flexible architectural type. Based on the HTTP protocol, REST APIs enable stateless communication between the system's front and back end. They provide standard HTTP methods such as PUT, GET, DELETE, POST, and PATCH. The main advantage of REST is its statelessness, where each request must include all the information necessary to process it. REST APIs, intriguingly, do not require any server-side sessions [11].

3.3.6 CloudFront

Amazon CloudFront is a service that helps deliver web content to clients, improving access to dynamic web content, such as web applications, frontend development, and image files. CloudFront delivers web content using edge locations, providing low latency and high data availability, thus enhancing the client's experience [12].

3.3.7 Certificate Manager

AWS Certificate Manager (ACM) is a service that helps renew certificates and create and store public and private SSL/TLS keys that protect website applications. ACM can issue protection certificates for all types of names in the Domain Name System (DNS) [13].

3.3.8 Route 53

Amazon Route 53 is a service that offers scalable and high-availability Domain name systems. It checks domain registration and DNS routing. Route 53 connects user requests to internet applications running on AWS or on-premises. It translates human-readable names into numeric IP addresses that computers use to connect, ensuring low latency and high data availability [14].

3.4 Admin Interface Design

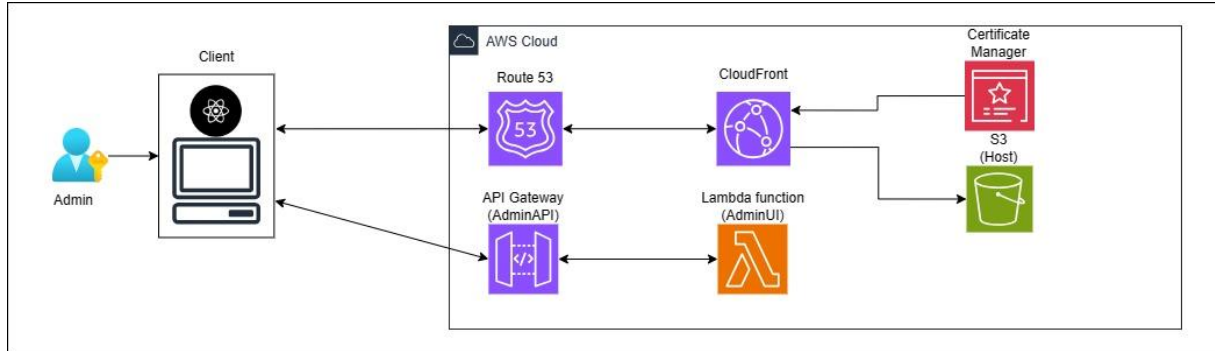


Figure 10 Admin Interface design- serverless architecture.

The admin interface allows the admin to add inexperienced users and give each user permission to open the lock. The interface system is built using serverless architecture, eliminating the need to manage infrastructure and all server operations on Amazon Web Services. The lambda function is used as the backend or server, and the API gateway links the backend with the front end using Rest API. The frontend design uses the React library, which provides optimization, Scalability, and Flexibility, improving User Experience. S3 hosts and stores the data of the interface application, and the client can access the interface using Route 53, specifying the domain name and redirecting the client to CloudFront. CloudFront will deliver the web content from S3 to clients in a secure connection using a Certificate manager [15].

Chapter 4

System Implementation

The system is a smart lock that integrates with Amazon Web Services (AWS). It identifies users, shows user logs activities, and provides an admin interface for user management and log access and add users. The system leverages AWS's serverless architecture for the management interface, capitalizing on its scalability, flexibility, and cost-effectiveness. All hardware components and software applications work cohesively to form a complete system. In addition, AWS provides a combination of services, making the system reliable for users.

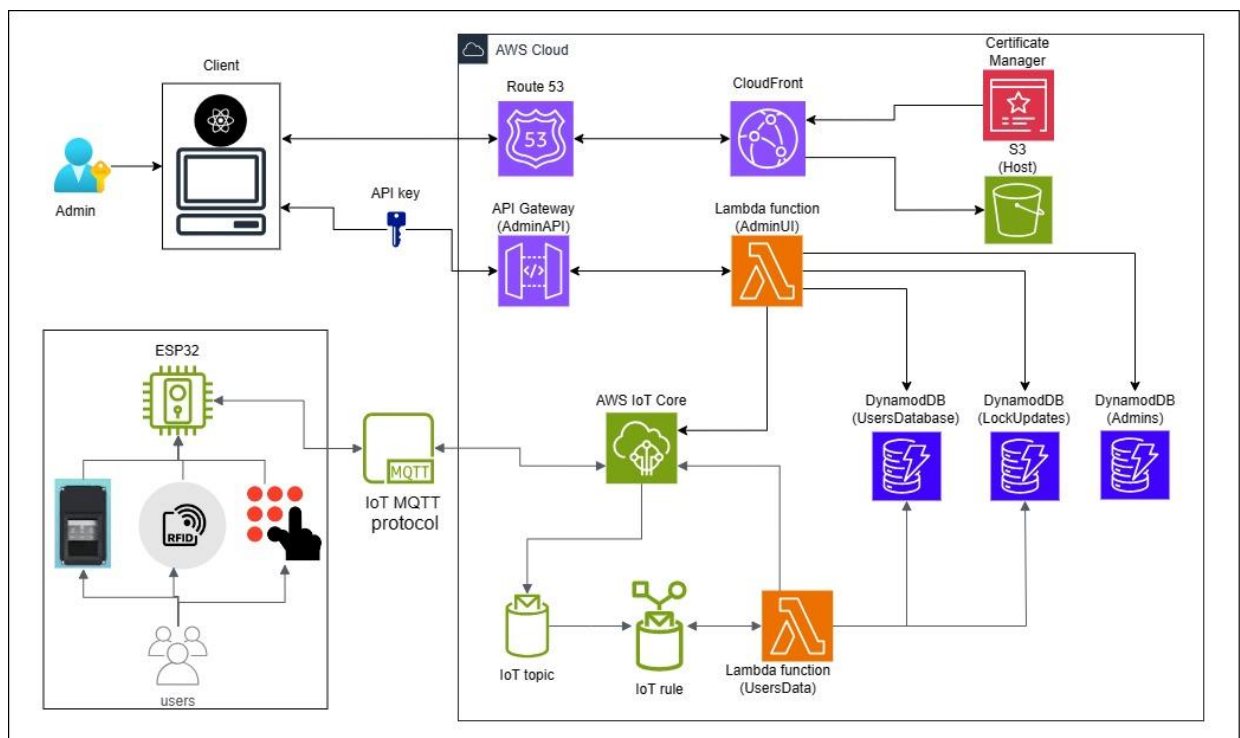


Figure 11 Integrated Smart Lock System Architecture Diagram

4.1 Software Setup and Development

4.1.1 Platforms

Arduino Software (IDE) is a platform that helps users write code in a text editor, process and compile different languages, such as C, C++, or header files, provide easy access to libraries, and handle errors. It is used to upload code and sketches to ESP32 and maintain the hardware component [16].

Visual Studio is a developer tool with multiple features. An integrated development environment contains a text editor that supports a wide range of programming languages using a library and provides flexibility for handling and debugging the errors used to write the Python code, upload it to the lambda function, and write the frontend and backend software [17].

4.1.2 Setup a Component Configuration & Libraries

The setup of the ESP should be done by organizing and distributing the pins on the ESP with components and synchronizing the data flow with the components.

The system configuration starts with the selection of NTP (Network Time Protocol) servers for time synchronization. Two servers are used: Pool.ntp.org and time.nist.gov, and the Greenwich Mean Time (GMT) offset is set to 10800 seconds, which corresponds to Bahrain's time zone (+3 GMT). The Adafruit Fingerprint library is configured to handle fingerprint sensor operations. Additionally, the MFRC522 library is configured for an RFID reader, and the LiquidCrystal_I2C library is used to manage the LCD. The keyboard configuration includes specifying the number of rows, columns, keys, and pins for the rows and columns. The keyboard library is initialized with these parameters. The WiFiClientSecure and PubSubClient libraries are configured for network operations to handle secure Wi-Fi and MQTT connections. The JSON Web Token (JWT) configuration consists of configuring the CustomJWT library with a secret key that saves in secret.h file to decrypt the JWT tokens.

4.1.3 Initialize Amazon DynamoDB

Create three DynamoDB tables to handle the requirement or system. The first table, "LockUpdates", is designed to track lock updates. It uses 'timestamp' as the primary key to sort and retrieve updates based on their event and a sort key called 'sort' to further organize the data. The second table, "Admins", is intended to store information about system administrators. It uses the Administrator ID "AID" as the primary key, uniquely identifying each administrator. The third table, "UsersDatabase" stores user information. The "ID" is the primary key to identify each User uniquely. Three secondary indicators enable efficient

querying: "FingerprintID-index", "Password-index", and "RFID-index" These indexes facilitate quick searches based on the User's fingerprint ID, password, and RFID.

4.1.4 AWS IoT Core Configuration & secret.h

Setting up AWS IoT Core starts with creating a "thing" called "SmartLock". This "thing" is linked to the policies that govern the ability to connect, publish, subscribe, and receive messages, ensuring secure and efficient communication. The automatically generated certificate provided by IoT Core is used for authentication, and the endpoint is created for communication purposes.

The Secret.h file is critical in storing the credentials and configuration details necessary for Thing to securely interact with AWS IoT Core. This includes a Wi-Fi SSID and password to access the network, an AWS IoT endpoint to connect to the IoT Core service, MQTT topics to publish and subscribe to messages, and certificates for secure communication. And the secret key for JWT encoding. These elements allow "SmartLock" to function effectively within the AWS IoT Core environment.

4.1.5 AWS Hosting & Admin Interface Configuration

Set up a domain name using domain registration from AWS Route 53. The domain "smartlockaws.xyz" is registered, providing Admins access to the web application. This process also creates a hosted zone for the registered domain. A new bucket is created on AWS S3. The bucket is named identically to the domain "smartlockaws.xyz". A getObject policy is added to the bucket, and a static website on Amazon S3 is hosted using the static website hosting feature. Then, a new public SSL/TLS certificate is requested from the AWS Certificate Manager (ACM). This certificate is linked to the Route 53 hosted zone and the CloudFront distribution point, ensuring secure and efficient content delivery. The distribution point is created by the CloudFront service and linked with the static host endpoint in S3, and the custom SSL certificate is created by the Certificate Manager. All these companies provide a secure and low-latency connection between the client and the server hosting.

4.2 Token and Access Validation Processing

Different types of tokens are used to securely transmit information and for authorization and authentication. and For example, JWT (JSON Web Token), SWT(Simple Web Token), and

SAML (Security Assertion Markup Language). The JWT has many benefits such as, its being more compact which means when the JWT is encoded it has a small size and it's more secure than other tokens by using a public and private key. [18]

In our system, the JWT is used to validate the user's access to the smart lock. The JWT structure has 3 parts: Header, Payload, and Signature. The header contains the algorithm used and the type of token. In the payload we can add any type of data for example in our system we add the user ID, expiration time/data, and state. In the signature, it hashes the header and the payload by using the `base64UrlEcnode` with the secret key. Moreover, the JWT structure is used for authentication and authorization of the admin interface session.

4.2.1 User Registration

In the user registration, the admin can give each user different access limits to the lock. The payload message contains the following fields:

- "id": A unique identifier for the user.
- "exp": The expiration time or time range during which the token is valid.
- "state": A state indicator that determines the type of validation to perform.

For example, if the state is '1' the token is valid up to the date on the exp. If the stat is '2' the token is valid between the date that in exp. If the state is '3' the token will be valid daily between the time that is written in the exp. In addition, the admin can give an always valid token to the user. By doing this the admin has more flexibility to give access limits to each user based on their needs.

4.2.2 Log-in Authentication

The session token is created after the admin ID and password are authenticated. It is based on HS256 encryption and contains a payload and secret key, JWT KEY, which is initialized as an environment variable in the lambda configuration.

The payload message contains the following fields:

'adminId': Admin identification,

'exp': session expiration time by default is one hour.

After every event on the client side, the admin interface sends the admin ID and the token to verify the token's validity by decrypting the payload using JWT_KEY and checking the

admin ID with the payload's admin ID, expiration time, and token validation. The session ends with a logout, expired token, invalid admin, or invalid token.

4.3 ESP32 Connection to AWS and WIFI

To enable communications between ESP32 and the AWS platform a secure Wi-Fi connection was established and it was done by several steps. The "connectAWS()" function is called when the esp32 starts. Within this function, the Wi-Fi mode is set to "WIFI_STA" and the esp32 will connect to the network with WIFI_SSID and WIFI_PASSWORD. And during the process, it will check the Wi-Fi state by using "WiFi.status()". While the connection is processed the LCD will display a message of connection with the network name. Once the esp32 is connected to the Wi-Fi successfully the function will configure the WiFiClientSecure with the AWS IoT device credentials. These credentials include the client certificate (CRT), the Certificate Authority (CA) certificate, and the private key. They are essential for establishing secure communication with the AWS IoT platform.

After the WiFi connection, the MQTT will be configured by the AWS IoT endpoint and MQTT broker's port "8883" This will establish bidirectional communication. In addition, to handle the incoming messages from the AWS the callback function is set by "client.setCallback()" function. If the connection to the AWS IoT is successful, it subscribes to the specific topic using "client.subscribe()" function, the topic is "smartLock/sub". By doing so, a real-time interaction and data exchange between AWS and ESP32 is established successfully.

The MQTT (Message Queuing Telemetry Transport) protocol is designed for M2M (machine-to-machine) communication. This communication uses TCP/IP protocols, and it can transfer the data in different forms like text, XML, and JSON. In addition, this protocol uses a publish/subscribe model instead of a client-server model. [19]

4.4 Fingerprint Processing

The Fingerprint It's one of the methods to open the lock in the IoT smart lock system. The FingerPrint() function verifies the user's identity based on the fingerprint that reads it from the sensor. It will check if the fingerprint matches to preregistered in the database.

The function started by calling the 'getFingerPrint()' function to read the user fingerprint if the returned value is -1 which indicates invalid input or an error occurred during the fingerprint capture. If a valid fingerprint is captured then it will return the ID of the user in the sensor storage. In addition, the user ID must be checked if it's registered in the database so the "checkFingerPrint(fingerPrintID)" function will process this by sending an MQTT message to the AWS and it will get a response if it's valid or not. Then to validate the user Token, the token will be obtained by calling the "getUserToken()" function. Lastly, To display the user name on the LCD screen the "getUserName()" function is used.

If the token is not valid or the read fingerprint in the sensor is not found on the database the access will be denied and the "Access is Denied" message will be displayed on the LCD.

4.5 RFID Processing

RFID is another method to open the smart lock on the system. It is responsible for verifying the user's identity based on the RFID cards or tags. Since the card is vulnerable to loss, the user must enter the correct password as an additional condition to open the lock. So the user can access the lock only if he has a valid RFID card, correct password, and valid token.

The function started by calling the "getRFID()" function to detect the card that is present on the sensor and check if it's a valid card. If it's a valid card and it is found in the system database the user must enter the password by calling the "verifyPassword(UserPassword)" function. After that, that last condition must be valid which is the token, if all these conditions are satisfied then the lock will open.

If the token or the password is not valid then the access is denied and this message will be displayed on the LCD.

4.6 AWS Lambda Functions

The system relies on two AWS lambda functions (UserData) that enable it to handle all the requests and responses coming from the ESP32. The lambda function (AdminUI) handles and responds to all messages from the admin interface. Both functions must be provided with full access to DynamoDB and IoT Core by granting permissions to the two functions. All lambda functions are built using "python 3.8".

4.6.1 (UsersData) & ESP32 System

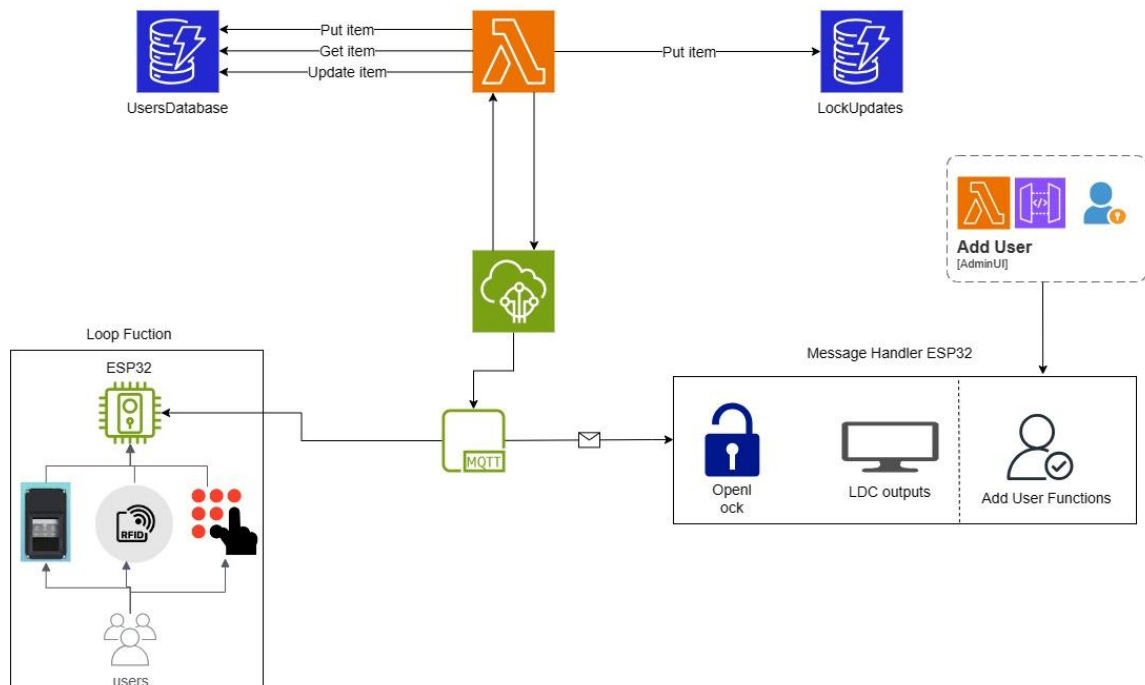


Figure 12 UsersData and Esp32 Dataflow

After each attempt to open the lock, whether it is successful or not, Esp32 sends a message to the lambda function, which in turn processes the message and compares it with the user information on DynamoDB using the `get_item` function from the `boto3` library and a response is built according to the situation and sent to Esp32.

All messages coming from the lambda function to the Esp32 are handled using the "message handler ()" function. This function determines the type of question and the tasks required, such as opening the lock, printing sentences on the LCD, or adding data for a new user (Fingerprint, RFID, Password, and Token).

On each open lock operation, a message will be sent from Esp32 to the lambda function (UserData). The payload message contains the username, the method of unlocking, and the timestamp for unlocking.

(UserData) use the `put_item` function to insert a new item in the LockUpdates table in DynamoDB.

4.6.2 (AdminUI) lambda

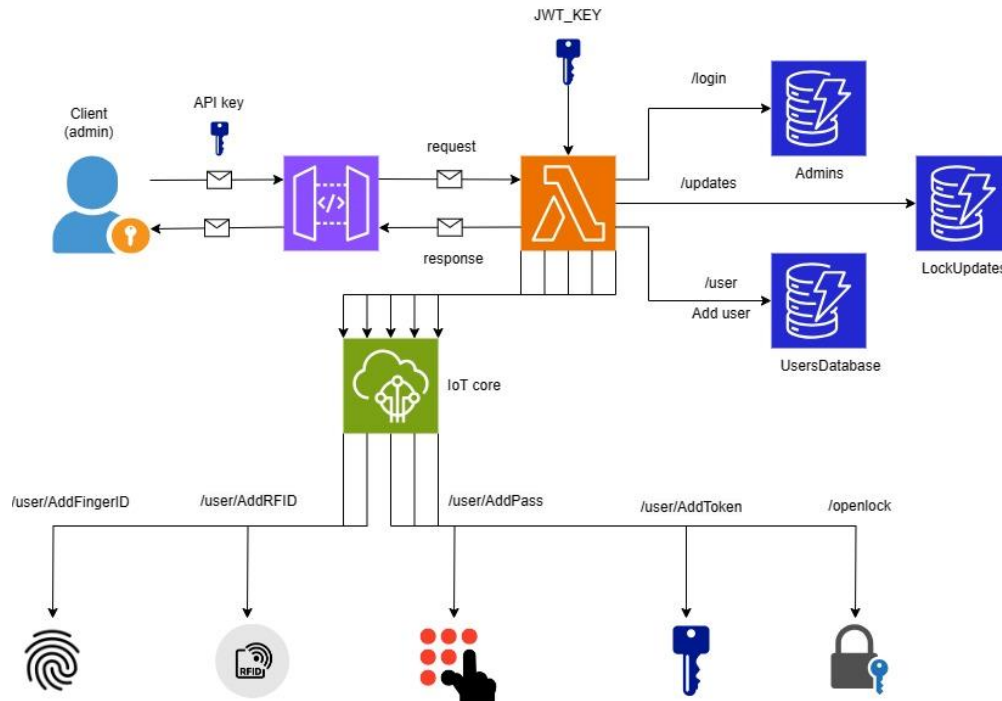


Figure 13 AdminUI lambda Function & REST API

Deploy REST API on AWS API Gateway with the name "AdminAPI", providing seven main paths and four sub-paths. The API Gateway Creates an OPTIONS method that allows all origins, all methods, and several common headers by enabling CORS (Cross Origin Resource Sharing) on all paths.

The AdminUI lambda function acts as a server for the client side, managing all requests and responses between the admin and author services. This setup enhances access to all data and manages users anytime and anywhere.

The administrator authentication login page sends a POST request to Api_Url/login via the AdminUI through the REST API Gateway (AdminAPI). To improve the connection's security, The AdminAPI denies access and requests without an API key. AdminUI uses the get_item function to retrieve a project from DynamoDB (Admins) and matches the requested data against the database data, such as ID and Password. All sensitive data are encrypted using a one-way hashing function.

Adding a User on DynamoDB using the AdminUI Lambda and ESP32 involves a sequential five-stage process to ensure a comprehensive and secure registration of a new user. Initially, the "save_user" function stores the User's basic data in DynamoDB, handling any errors that might occur during the process. Following this, the "save_FingerID" function checks and registers the User's fingerprint ID, ensuring it's within valid ranges and not duplicated, then

publishes this information to the MQTT topic "smartLock/sub". Similarly, the "save_RFID" function registers the User's RFID tag, and the "save_Password" function handles the password registration, both using MQTT for consistent system updates. Finally, the "save_Token" function updates the User's access token in DynamoDB based on its expiration and publishes this message through MQTT.

The delete_user function matches if a user exists and then attempts to delete their record from DynamoDB. Using the ID of the User to identify and search on DynamoDB to find a record that matches the ID of the User. If the User exists, then delete the User from the User's table. Otherwise, returns an error message.

4.7 Admin Interface & React Application

The React application hosted on Amazon S3 and delivered through Amazon CloudFront distribution and Route 53 identifies the domain name, with domain security managed by Amazon Certificate Manager (ACM), integrates JavaScript codes to handle administrator interactions and AWS service integrations efficiently.

The login.js component is responsible for administrator authentication, recording the administrator ID and password, hashing the password using MD5, and sending a POST request to a Lambda function. This process either starts an administrator session or handles errors based on data validation.

The admin_console.js component serves as the administrative interface, offering several functionalities: sending a POST request to the ESP32 to unlock the door (Open Lock), retrieving lock activity logs (Get Logs), fetching user data from the 'UsersDatabase' table in DynamoDB (Get All Users), providing user registration through sequential POST requests (Add User), deleting users from the database (Delete User), and terminating sessions (Log Out).

AuthenticatedApp.js, PrivateRoute.js, and PublicRoute.js ensure that administrative functions are protected and only accessible by authenticated users, while public routes are available to all users. The AuthService.js component is crucial for managing session storage, and continuity is essential for maintaining secure and persistent user sessions across the application.

Chapter 5

Testing and Results

This chapter provides the result of all scenarios of opening the lock, opening from the web page by the admin, opening the lock using fingerprint, opening the lock using RFID and password, test the validation of tokens of user and how will the lock manage the access.

5.1 Opening the Lock from the web page by the admin

The admin can open the lock using one button in the webpage.

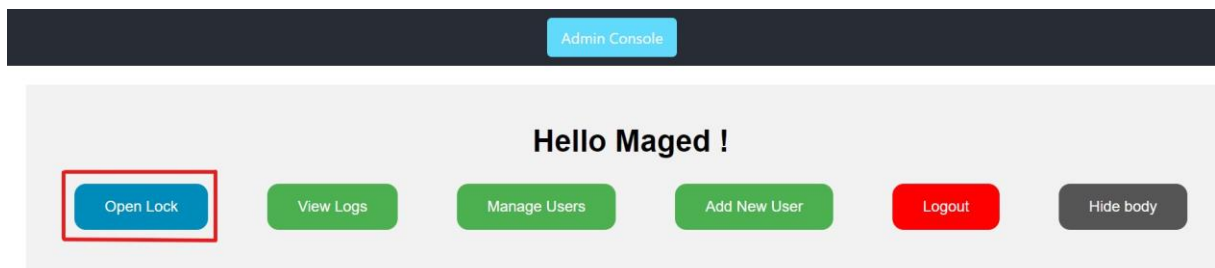


Figure 14 Admin Interface (Open Lock)

When pressed the lock is opened instantly without any delay, with a messaged shown saying “Welcome (Admin name)”



Figure 15 LCD welcoming Admin after opening the lock.

5.2 Register User

The admin can register the user and attach fingerprint or RFID to him using the admin interface.

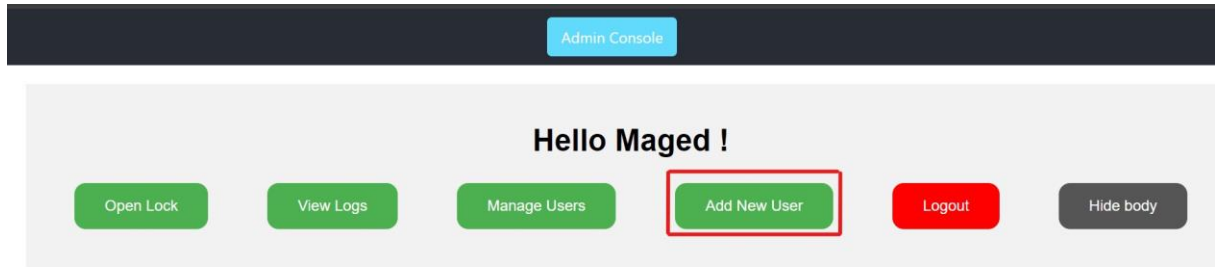


Figure 16 Admin interface (Add New User)

Then filling out the user information form:

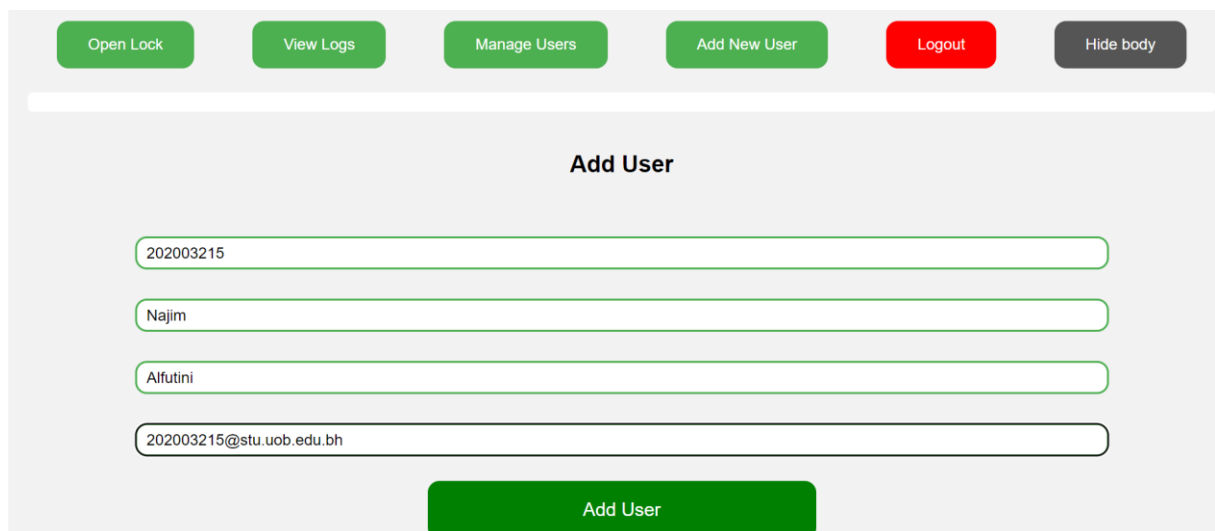
The image shows the "Add User" form in the Admin Interface. At the top, there is a row of six buttons: "Open Lock" (green), "View Logs" (green), "Manage Users" (green), "Add New User" (green), "Logout" (red), and "Hide body" (dark gray). Below this, the main area has a light gray background. It starts with the text "Add User" in bold. Below this, there are four input fields with green borders, each containing a value: "202003215", "Najim", "Alfutini", and "202003215@stu.uob.edu.bh". At the bottom, there is a green button labeled "Add User".

Figure 17 Admin Interface New User Information

- Adding Fingerprint for the user:

After pressing “Add User”, a “SUCCESS” notification will appear if the user was added successfully.

Hello Maged !

Open Lock

View Logs

Manage Users

Add New User

Logout

Hide body

Cancel Registration

User ID: 202003215

Add Finger ID

2

Add Finger ID

Figure 18 Admin Interface (Add Finger ID 2)

Adding Fingerprint ID = 2. Then the user will be asked to put his finger twice in the fingerprint sensor.

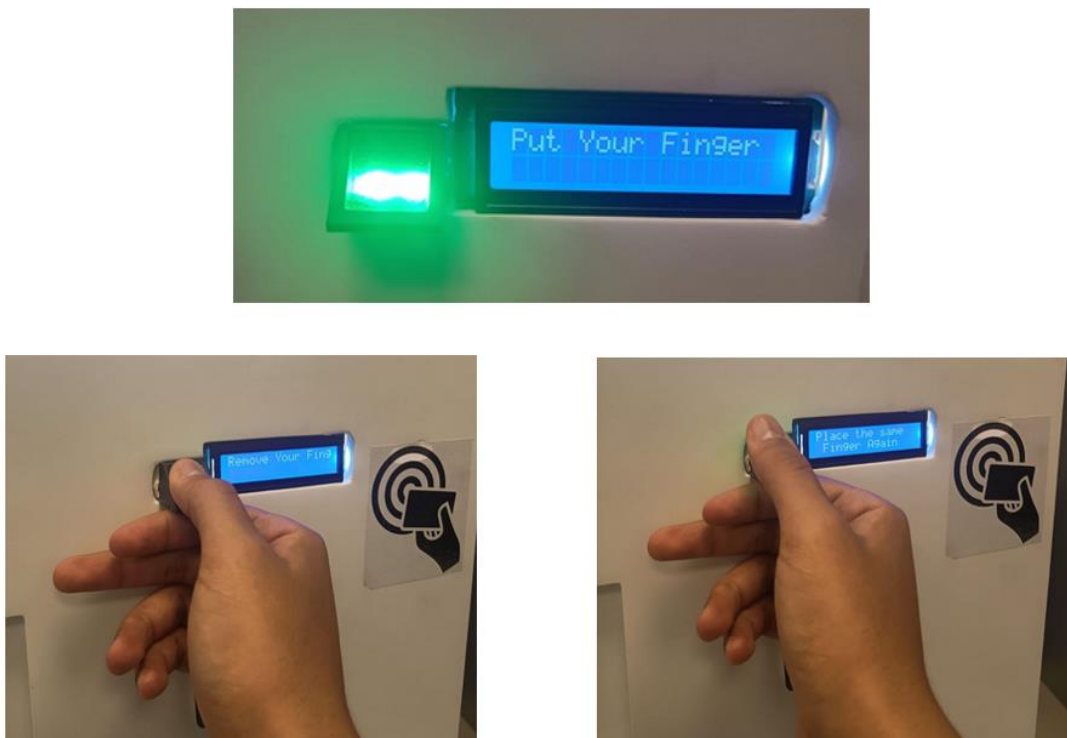


Figure 19 Registring Finger print.

- Adding RFID for the user:

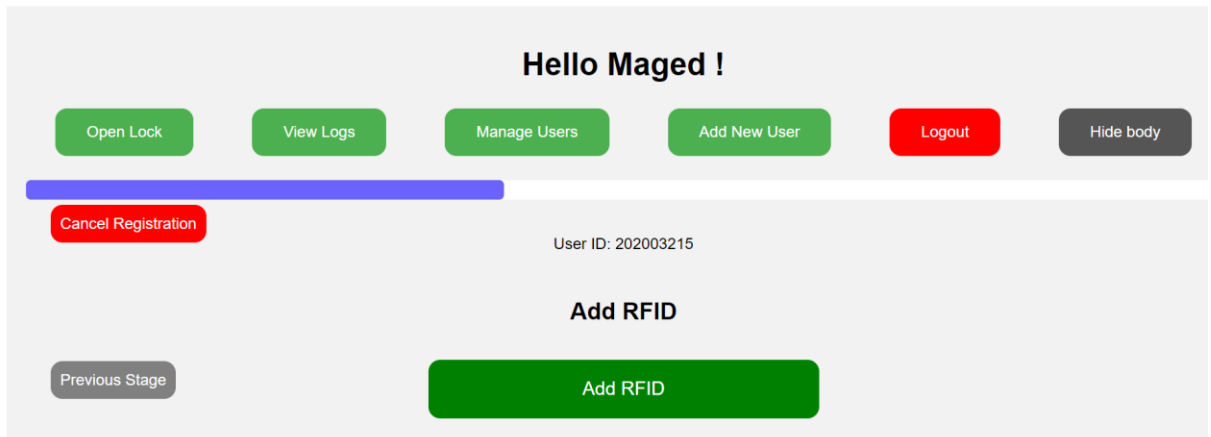


Figure 20 Admin Interface (Add RFID)

After pressing “Add RFID” in the webpage the user must put his RFID tag in the RFID sensor range.



Figure 21 Registering RFID Tag

- Adding Password for the user’s RFID.

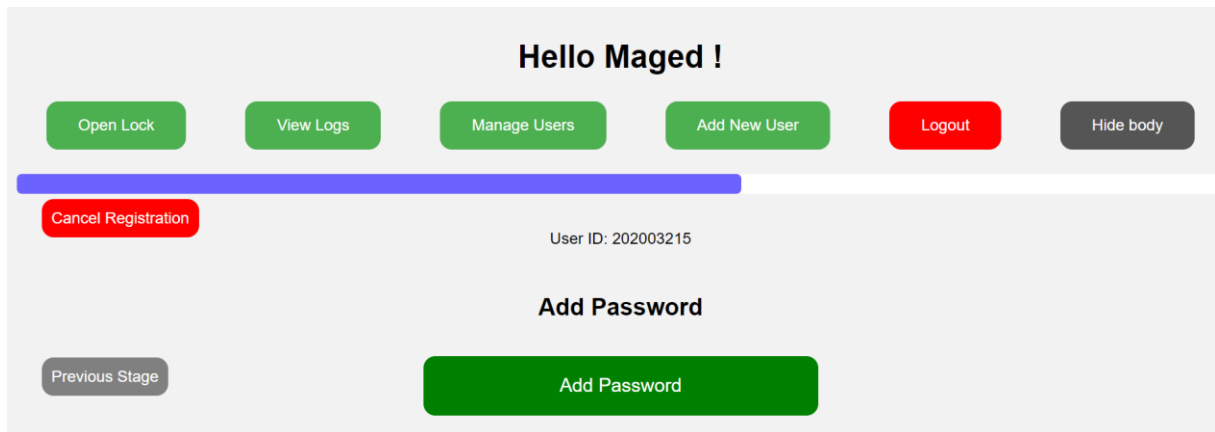


Figure 22 Admin Interface (Add Password)

After pressing “Add Password”, the user will be asked to put his password in the keypad and then press “#” to confirm the password.



Figure 23 Registering a New Password

- Adding Token for the user:

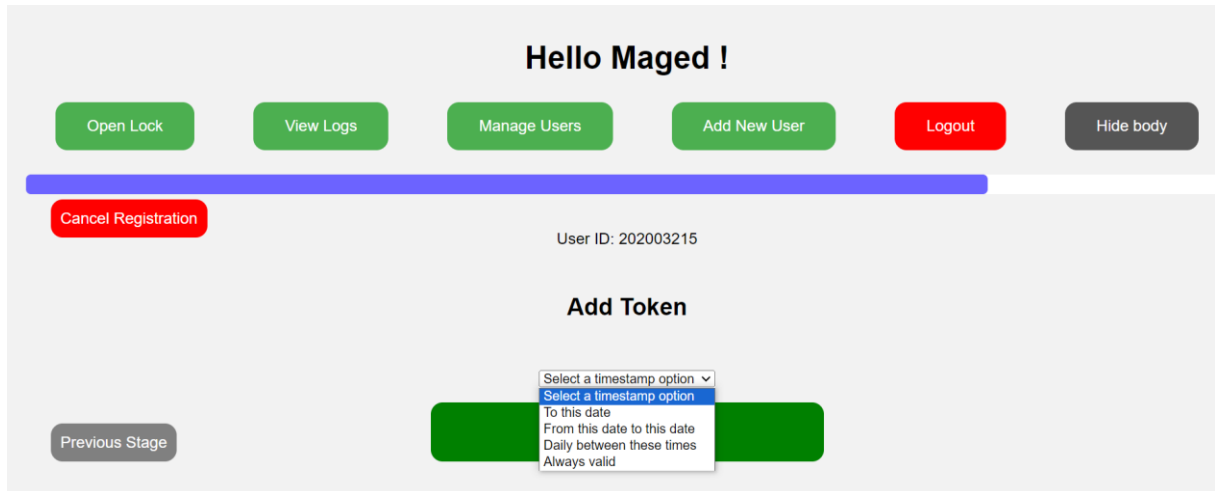


Figure 24 Admin Interface (Add Token)

We put the token as always valid for the user “Najim”, so he can access the lock anytime.

After doing all these steps the user was added successfully as we can see in the picture:

The screenshot shows the 'Users Table' in the Admin Interface. At the top, there's a greeting 'Hello Maged !' and a navigation bar with buttons: 'Open Lock', 'View Logs', 'Manage Users', 'Add New User', 'Logout', and 'Hide body'. Below the navigation bar, there's a table with the following data:

ID	First Name	Last Name	Email	FingerPrint	RFID	Access Token	Action
1	WALEED	SALEH	202006448@stu.uob.edu.bh	1	50770723	02:13:00 02:17:00	
202003215	Najim	Alfutini	202003215@stu.uob.edu.bh	2	94f87a85	Always	

Figure 25 Users Table

5.3 Opening the Lock Using Fingerprint

So, now we are going to try to open the lock using the fingerprint as the new user that we added “Najim”.



Figure 26 Opening the Lock Using Fingerprint

The Lock opened successfully, and it took the fingerprint sensor to authenticate my finger and open the lock around 1.4 seconds.

5.4 Opening the Lock using RFID and Password

We tried opening the lock using the RFID associated to the user “Najim” and it asked for the password as we expected, after putting the correct card and correct password the lock was opened instantly with a delay of 200ms to 500ms.



Figure 27 Opening the Lock using RFID and Password

5.5 Testing Token Validation

We made Three new users to test all three options of token.

- User “Adnan”: with token of daily access from 3PM to 4PM.
- User “Mohammed” with token from today till date 13/5/2024 last time 4PM.
- User “Luai” with token that gives access between two dates: from 12/5/2024 at 8:00 AM to 13/5/2024 at 4:00 PM

ID	First Name	Last Name	Email	FingerPrint	RFID	Access Token	Action
1	WALEED	SALEH	202006448@stu.uob.edu.bh	1	50770723	02:13:00 02:17:00	
202003215	Najim	Alfutini	202003215@stu.uob.edu.bh	2	94f87a85	Always	
5	Luai	Adnan	Luai@gmail.com	5		12/05/2024 08:00:00 - 13/05/2024 16:00:00	
4	Mohammed	Abdulkarim	mohammed@gmai.com	4		13/05/2024 16:00:00	
3	Adnan	Abdulkarim	adnan@gmail.com	3		15:00:00 16:00:00	

Figure 28 Users For Token Testing

- User “Adnan”:



Figure 29 User "Adnan", Daily Token

User “Adnan” with token of the daily type, he has access to the lock everyday from 3:00PM till 4:00PM, in the experiment, when “Adnan” tried to access the lock at the given time 3:49PM, the lock opened successfully, and when he tried to access it at 4:16PM, he got “Invalid Token, Access Denied” message from the LCD and the lock didn’t open.

- User “Mohammed”:



Figure 30 User "Mohammed" To This Date and Time Token

User “Mohammed” with token of the “To this date and time” type, he has access to the lock from the date and time the token was generated 12/5/2024 1:00PM till the chosen date and

time which was 13/5/2024 at 4:00PM, in the experiment, when “Mohammed” tried to access the lock at 13/5/2024 3:52PM, the lock opened successfully, and when he tried to access it at 13/5/2024 4:21PM, he got “Invalid Token, Access Denied” message from the LCD and the lock didn’t open.

- User “Luai”



Figure 31 User "Luai" Between Two Dates Token

User “Luai” with token of the “between two dates and time” type, he has access to the lock between two specific dates like shown in the picture above from 12/5/2024 at 8AM till 13/5/2024 at 4PM, in the experiment, when “Luai” tried to access the lock at the given time 13/5/2024 3:55PM, the lock opened successfully, and when he tried to access it at 13/5/2024 4:16PM, he got “Invalid Token, Access Denied” message from the LCD and the lock didn’t open.

So, from the experiments all token types working flawlessly without any issues demonstrating a high level of functionality and compatibility across the tested range. The result ensures that the system is reliable and able to manage various types of tokens comfortably. These findings are important for many reasons. Firstly, the smooth operation of every type of token ensures a great user experience since the admin can use their preferred token with confidence without

worrying about system errors. Second, the system is more applicable in various contexts and user scenarios due to its capacity to manage a variety of token types.

Furthermore, a high level of stability and robustness in the system's design and implementation is indicated by the lack of any problems or failures during the experiments. This validates the system's efficacy in practical settings and bolsters its dependability in managing and securing access and authentication procedures.

Chapter 6

Conclusion and Future Work

During the course of this project, we successfully designed, implemented, and tested an IoT-based smart lock system that works in the cloud using AWS and combines multiple access control mechanisms, cloud computing, and secure communication protocols. The system provides remote access control, centralized management, and scalable deployment, which improves user security and convenience.

Our findings show that incorporating biometric authentication, such as fingerprint recognition, significantly improves the security of the smart lock system over traditional keypad-based entry systems. The use of secure communication protocols, such as MQTT and HTTPS, protects the confidentiality and integrity of data exchanged between smart lock devices and the cloud platform. Moreover, our system offers a user-friendly interface using a web page, allowing administrators to easily manage access permissions, monitor lock entry log, manage users, and adding users. The system's compatibility with various access control methods, such as RFID cards, fingerprinting, and web pages, provides flexibility and convenience to users with varying preferences. Throughout the project, we encountered several limitations and challenges that must be acknowledged. First and foremost, while biometric authentication provides strong security, concerns about privacy and biometric data storage must be addressed. Second, the system's dependence on internet connectivity is a potential drawback. Users may have difficulty accessing the smart lock system remotely in the event of a network outage or disruption. Implementing offline access mechanisms or backup authentication methods would help to address this issue. Another limitation is that smart lock devices are power dependent. Battery-powered locks may require frequent battery replacements, limiting the system's usability and maintenance. Exploring energy-efficient solutions and alternative power sources can help improve system sustainability.

While this project has reached significant milestones, there are many opportunities for future work and development in this project. Some possible areas for future research include:

- **Enhanced Privacy Measures:** Additional research into privacy-preserving techniques for biometric data storage and authentication methods to address privacy concerns.
- **Advanced Authentication Methods:** Investigate emerging authentication technologies, such as facial recognition or voice recognition, and integrate them into the smart lock system for increased security and user convenience.
- **Edge Computing and Edge AI:** Research into edge computing architectures and edge AI algorithms to enable local processing and decision-making in smart lock systems, reducing reliance on cloud services and increasing system responsiveness.
- **Integration with Smart Home Ecosystems:** The smart lock system is integrated with other smart home devices and platforms, allowing for seamless automation and interoperability across a larger smart home ecosystem.
- **Usability and User Experience:** Conducting user research and evaluations to improve the smart lock system's user interface, mobile application, and overall user experience by adding more options to the admin like, editing user information and change token.

References

- [1] B. Choudhary, C. Pophale, A. Gutte, A. Dani, and S. S. Sonawani, "Case Study: Use of AWS Lambda for Building a Serverless Chat Application," *Proceeding of International Conference on Computational Science and Applications*, p. 237–244, 2020.
- [2] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart Locks: Lessons for Securing Commodity Internet of Things Devices," in *in Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, Xi'an, China, 2016.
- [3] l. ada, "Adafruit HUZZAH32 - ESP32 Feather," Adafruit Industries, 2024.
- [4] M. 73, "What Is a Lock Solenoid?," 2024.
- [5] H. Technology, "RC522 RFID Development Kit," Handson Technology, 2020.
- [6] l. ada, "Adafruit Optical Fingerprint Sensor," Adafruit Industries, 2024.
- [7] I. AWS, "AWS IoT Core. AWS. Retrieved from <https://aws.amazon.com/iot-core/>".
- [8] "What is Amazon DynamoDB?," Amazon Web Services, Inc., 2024. [Online]. Available: <https://docs.aws.amazon.com/dynamodb/>.
- [9] "Amazon Simple Storage Service Documentation," Amazon Web Services, Inc., [Online]. Available: Amazon Web Services, Inc..
- [10] "Amazon API Gateway Documentation," Amazon Web Services, Inc, [Online]. Available: <https://docs.aws.amazon.com/apigateway/>.
- [11] S.-P. Ma, M.-J. Hsu, H.-J. Chen, and C.-J. Lin, "RESTful API Analysis, Recommendation, and Client Code Retrieval," *Electronics*, vol. 12, no. 5, p. 1252, 2023.
- [12] "Amazon CloudFront Documentation," Amazon Web Services, Inc, [Online]. Available: <https://docs.aws.amazon.com/cloudfront/>.
- [13] "AWS Certificate Manager Documentation," Amazon Web Services, Inc, [Online]. Available: <https://docs.aws.amazon.com/acm/>.
- [14] "Amazon Route 53 Documentation," Amazon Web Services, Inc., [Online]. Available:

<https://docs.aws.amazon.com/route53/>.

- [15] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, "Survey on serverless computing," *Journal of Cloud Computing*, vol. 10, p. 1, 2021.
- [16] "Arduino Integrated Development Environment (IDE) v1," Arduino , 17 1 2024. [Online]. Available: <https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics/>. [Accessed 5 5 2024].
- [17] "Visual Studio IDE documentation," Microsoft, [Online]. Available: <https://learn.microsoft.com/en-us/visualstudio/ide/?view=vs-2022>.
- [18] Fu, Jamie Liu, Richard Wong, Anna Liu, Katherine, "JWT Web Security," 2022.
- [19] R. A. Atmoko, R. Riantini, and M. K. Hasin, "IoT real-time data acquisition using MQTT protocol," *IOP Conf. Series: Journal of Physics: Conf. Series*, vol. 853, no. 1, p. 012003, 2017.