

AUTOMATIC DOOR LOCK/UNLOCK SYSTEM



PROJECT REPORT

Submitted by

DHARSHINI R (9517202109014)
SANDHIYA DEVI B (9517202109046)
SHRI SUBIKSHA S T (9517202109048)

in

19AD691 –PRINCIPLES OF SOFTWARE ENGINEERING

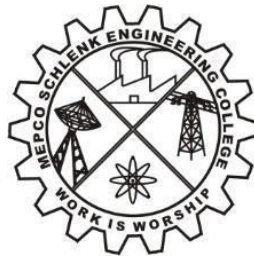
DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE
SIVAKASI

APRIL 2024

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI
AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE



BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **DHARSHINI R (9517202109014), SANDHIYA DEVI B (9517202109046), SHRI SUBIKSHA S T(9517202109048)** for the mini project titled **“AUTOMATIC DOOR LOCK/UNLOCK SYSTEM”** in 19AD691 –**PRINCIPLES OF SOFTWARE ENGINEERING** during the sixth semester December 2023 –April 2024 under my supervision.

SIGNATURE

Dr.P.Thendral, M.E.,Ph.D
Associate Professor,
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

SIGNATURE

Dr. J. Angela Jennifa Sujana, M.E.,Ph.D
Professor & Head,
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi.

Software Requirements Specification

for

Automatic Door Lock/Unlock Using Mobile App

Prepared by

Dharshini(21BAD008)

Sandhiya Devi B(21BAD022)

Shri Subiksha S T(21BAD024)

Mepco Schlenk Engineering College

Table of Contents

Table of Contents.....	ii
1. Introduction.....	1
1.1 Purpose.....	1
1.2 Document Conventions.....	1
1.3 Intended Audience and Reading Suggestions	1
1.4 Product Scope	2
1.5 References.....	2
2. Overall Description.....	2
2.1 Product Perspective.....	2
2.2 Product Functions	2
2.3 User Classes and Characteristics.....	3
2.4 Operating Environment.....	3
2.5 Design and Implementation Constraints	3
2.6 User Documentation	3
2.7 Assumptions and Dependencies.....	3
3. External Interface Requirements	4
3.1 User Interfaces	4
3.2 Hardware Interfaces	4
3.3 Software Interfaces	5
3.4 Communications Interfaces	5
4. System Features	6
4.1 System Feature 1	6
4.2 System Feature 2 (and so on).....	7
5. Other Nonfunctional Requirements.....	9
5.1 Performance Requirements	9
5.2 Safety Requirements	9
5.3 Security Requirements	9
5.4 Software Quality Attributes	10
5.5 Business Rules	10
6. Other Requirements	10
7. Glossary	10
8. Analysis Models.....	13
9. Complexity Analysis... ..	16
10. Blackbox Testing.....	21
11. Outputs	26
12. Conclusion	76
13. References.....	76
Appendix :Implementation	40

1.Introduction

The door locking system project introduces an innovative approach to access control by leveraging Arduino and a Bluetooth module. This initiative addresses the growing need for secure and convenient door lock/unlock operations from close proximity via a mobile app. By integrating these technologies, the project aims to revolutionize traditional lock and key systems, providing users with a modern and adaptable solution for managing door access.

1.1. Purpose

The purpose of the door locking system project is to develop a cutting-edge solution that allows users to control door access remotely. The integration of Arduino and the Bluetooth module, along with the mobile app developed using Android Studio, serves to create a secure, efficient, and user-friendly system. This project aims to offer an alternative to traditional physical keys, enhancing security measures while providing flexibility and convenience for users to manage access control through their smartphones.

1.2. Document Conventions

To ensure clarity and consistency in communication, this project adheres to specific document conventions. Bold text is used for section headings, italicized text emphasizes key terms, and code font is employed for technical elements. These conventions streamline communication among team members, ensuring a standardized approach to the project's development.

1.3. Intended Audience and Reading Suggestions

The door locking system project caters to developers, project managers, and stakeholders. Developers are encouraged to focus on the technical aspects of integrating Arduino, Bluetooth module, and app using Android Studio. Project managers should prioritize understanding timelines, resource requirements, and milestones, while stakeholders are urged to explore the overall functionality and benefits of the developed system.

1.4. Product Scope

The product scope of the door locking system project encompasses the comprehensive development of a secure, scalable, and user-friendly access control solution. Integrating Arduino and a Bluetooth module, the system will enable users to remotely lock and unlock doors via a mobile app. The use of Android Studio ensures a seamless interface, making the solution adaptable for various settings, from residential homes to commercial establishments.

1.5. References

Key references for the project include the official Arduino Documentation, the Bluetooth Module Datasheet from the manufacturer, and the Android Studio Documentation references serve as essential resources for understanding the technical specifications, compatibility requirements, and integration guidelines necessary for the successful implementation of the door locking system.

2. Overall Description

2.1. Product Perspective

The Automatic Door Lock/Unlock System serves as a standalone product, providing a secure and technologically advanced solution for access control. It operates independently, without relying on or being a part of any existing product family. However, it does interface with external components like Arduino, Bluetooth modules, and the app using Android Studio, ensuring seamless communication for effective remote control of door access.

2.2. Product Functions

The primary function of the system is to enable users to lock and unlock doors remotely using a mobile app. This includes features such as real-time door status updates, secure authentication mechanisms, and emergency unlocking procedures. The system functions as an alternative to traditional physical keys, offering a modern and efficient way for users to manage door access effortlessly.

2.3. User Classes and Characteristics

The system caters to three main user classes: end-users (residents or employees), administrators, and maintenance personnel. End-users utilize the mobile app for door control, administrators manage access permissions and system configurations, while maintenance personnel handle troubleshooting and system maintenance. Each user class has distinct characteristics, with end-users prioritizing ease of use, administrators focusing on security configurations, and maintenance personnel emphasizing system reliability.

2.4. Operating Environment

The software operates in various environments, including residential spaces, commercial buildings, and industrial settings. It is compatible with common hardware platforms, such as Arduino boards, and supports multiple operating systems for the mobile app. The operating environment encompasses diverse scenarios, ranging from everyday home use to more complex industrial applications.

2.5. Design and Implementation Constraints

Design and implementation constraints include adherence to specific hardware interfaces for Arduino compatibility, utilization of Bluetooth module communication protocols, and compliance with Android Studio standards for mobile app development. These constraints guide the development team to ensure seamless integration and optimal functionality.

2.6. User Documentation

User documentation includes manuals for end-users, administrators, and maintenance personnel. The documentation covers installation guides, app usage instructions, security protocols, and troubleshooting procedures. The goal is to provide comprehensive support for users at every level, ensuring a positive and user-friendly experience.

2.7. Assumptions and Dependencies

Assumptions include the availability of a stable internet connection for Bluetooth module communication and the proper functioning of external components like Arduino. Dependencies include third-party libraries and

tools for mobile app development, as well as compliance with relevant security and privacy regulations during system implementation. These assumptions and dependencies form the foundation for successful project execution.

3.External Interface Requirements

3.1. User Interfaces

The mobile app will serve as the primary user interface for interacting with the door unlocking and locking system. The logical characteristics of this interface include:

Screen Layout: The app will have a simple and intuitive layout with distinct buttons for "Open" and "Close" operations.

Buttons and Functions: The main functionalities will include "Open" and "Close" buttons. Additionally, a help option and status indicators (showing whether the door is currently locked or unlocked) may be included.

GUI Standards: The app will adhere to common mobile app design standards for the selected platform (iOS or Android) to provide a consistent and user-friendly experience.

Error Messages: Clear and concise error messages will be displayed in case of any issues, providing guidance to the user on how to resolve problems.

Keyboard Shortcuts: As this is a mobile app, keyboard shortcuts may not be applicable. However, touch gestures and actions should be intuitive.

For further details on the user interface design, a separate user interface specification document will be created.

3.2. Hardware Interfaces

The software product will interface with the following hardware components:

Arduino Board: The logical interface will include the commands to control the locking and unlocking mechanism connected to the Arduino board.

Bluetooth Module: Interaction with the Bluetooth module will involve communication protocols for sending and receiving signals related to door status (locked or unlocked).

Door Locking Mechanism: Logical characteristics will define how the software interacts with the physical locking mechanism, ensuring proper synchronization and control.

3.3. Software Interfaces

The software product will interact with the following software components:

Arduino Software: The system will communicate with the Arduino software to send commands and receive status updates.

Mobile Operating System: Interaction with the mobile operating system to utilize standard functionalities like network access and user interface rendering.

3.4. Communications Interfaces

The software product will require the following communication functions:

Bluetooth Communication: Define the communication protocols for Bluetooth, specifying the format of messages exchanged between the mobile app and the Arduino-Bluetooth setup.

Data Transfer Rates: Specify the expected data transfer rates to ensure timely and efficient communication between the mobile app and the hardware components.

Security: Define any security measures, such as encryption, to secure communication between the mobile app and the Arduino-Bluetooth system, preventing unauthorized access or control.

By addressing these external interface requirements, the mobile app and hardware components can seamlessly interact to unlock and lock the door as intended.

4.System Features

4.1. Remote Unlocking

4.1.1 Description and Priority

Allowing users to unlock their doors remotely using the mobile app is a critical feature for enhancing accessibility and security. With a priority rating of High, this feature ensures users can conveniently grant access to their premises from anywhere, offering peace of mind and flexibility in managing door access.

4.1.2 Stimulus/Response Sequences

- User taps "Unlock" in the app interface.
- App transmits unlock command securely to Arduino via Bluetooth module.
- Arduino receives and processes the unlock command.
- Upon successful processing, Arduino activates the door unlock mechanism.
- Confirmation of door unlocking is sent back to the app.
- User receives real-time notification of the successful unlock.
- App interface updates to reflect the unlocked status.

4.1.3 Functional Requirements

REQ-1: Implement a user-friendly interface with a prominent "Unlock" option.

REQ-2: Establish a secure and reliable connection between the app and Arduino via Bluetooth module.

REQ-3: Ensure user authentication before allowing door unlock commands.

REQ-4: Transmit unlock commands securely to Arduino to prevent unauthorized access.

REQ-5: Arduino must accurately receive and process unlock commands to activate the door unlock mechanism.

REQ-6: Provide real-time feedback to the user about the door unlocking status.

REQ-7: Implement mechanisms to handle potential errors or interruptions during the unlocking process.

4.2. In-App Notifications

4.1.1 Description and Priority

Implementing in-app notifications for lock/unlock events is crucial for keeping users informed about the status of their doors. With a priority rating of Medium, this feature ensures users receive timely updates about door activities, enhancing overall security awareness and user experience.

4.1.2 Stimulus/Response Sequences

- Detection of door lock/unlock event.
- App generates an in-app notification.
- Notification is sent to the user's device.
- User receives and views the notification.
- Notification includes relevant details such as door status and timestamp.
- User acknowledges or dismisses the notification within the app.
- App interface updates to reflect the latest notification status.

4.1.3 Functional Requirements

- REQ-1: Implement real-time event detection mechanisms within the app.
- REQ-2: Automatically generate in-app notifications upon detecting lock/unlock events.
- REQ-3: Ensure prompt delivery of notifications to the user's device.
- REQ-4: Include clear and concise information about door status in the notifications.
- REQ-5: Allow users to customize notification settings according to their preferences.
- REQ-6: Maintain a log of past notifications for user reference.

- REQ-7: Provide the option for users to view and manage past notifications within the app interface.

4.2.1 Feature 3: Remote Monitoring

4.1.1 Description and Priority

Enabling remote monitoring of door status via the mobile app enhances user convenience and security. With a priority rating of High, this feature allows users to check the status of their doors from anywhere, ensuring they remain informed and in control of their premises' security at all times.

4.1.2 Stimulus/Response Sequences

- User accesses the door status monitoring feature within the app.
- App sends a query to the Arduino via the Bluetooth module.
- Arduino retrieves the current door status.
- Arduino sends the door status information back to the app.
- App receives and displays the door status to the user.
- User views the door status information on the app interface.
- App allows users to refresh or update the door status as needed.

4.1.3 Functional Requirements

REQ-1: Implement a dedicated feature for remote monitoring of door status within the app.

REQ-2: Ensure a reliable and secure connection between the app and Arduino for data retrieval.

REQ-3: Arduino must accurately process and respond to door status queries from the app.

REQ-4: Present door status information clearly and intuitively within the app interface.

REQ-5: Provide users with the option to manually refresh or update door status information.

REQ-6: Implement robust error handling mechanisms to handle interruptions or connection issues gracefully.

REQ-7: Support push notifications to alert users of significant changes in door status, ensuring timely updates and enhanced security awareness.

5. Other Nonfunctional Requirements

5.1. Performance Requirements

Response Time: The mobile app should have a response time of less than 2 seconds for all user interactions to ensure a smooth user experience.

Concurrent Users: The system should support a minimum of 100 concurrent users without significant degradation in performance to accommodate peak usage times.

Door Unlocking Time: The door unlocking process should take no longer than 5 seconds from the user's command to the door being successfully unlocked.

5.2. Safety Requirements

Fail-Safe Mechanism: The door unlocking mechanism must include fail-safe features to prevent unauthorized access in the event of system failure or tampering.

Compliance: The system should comply with safety regulations regarding electronic door locks to prevent accidents or injuries.

5.3. Security Requirements

User Authentication: User authentication must be secure, utilizing encryption protocols for data transmission and storage to protect user credentials.

Access Control: The system must implement role-based access control to ensure that only authorized individuals can unlock doors and access sensitive features.

Privacy Compliance: Compliance with GDPR and other privacy regulations regarding the collection and use of user data must be ensured.

5.4. Software Quality Attributes

Reliability: The mobile app should prioritize reliability and robustness to minimize downtime and errors.

Usability: Usability is a key quality attribute, with an emphasis on intuitive user interfaces and clear feedback mechanisms.

Maintainability: Maintainability of the system codebase is crucial for future updates and enhancements, with well-documented code and modular architecture.

5.5. Business Rules

Administrative Privileges: Only users with administrative privileges can grant guest access or modify access permissions.

User Authentication: Users must authenticate themselves before performing any action related to door unlocking or access control.

6. Other Requirements

Multilingual Support: The system should support multiple languages to cater to users from diverse linguistic backgrounds.

Internationalization and Localization: Internationalization and localization features should be included to adapt the system to different regions and cultures.

Legal Compliance: Legal requirements related to electronic access control systems in the project's jurisdiction must be adhered to.

Scalability: The system should be designed with scalability in mind to accommodate future expansion and growth.

7. Glossary

1. Arduino: An open-source electronics platform based on easy-to-use hardware and software, used to create interactive projects.

2. Bluetooth Module: Bluetooth module, a wireless communication module used for transmitting data over

the network.

3. Android Studio: A powerful integrated development environment (IDE) specifically designed for Android app development, offering advanced features and tools for building high-quality mobile applications.

4. Access Control: The process of selectively restricting access to a resource or physical space.

5. Authentication: The process of verifying the identity of a user, device, or entity.

6. Encryption: The process of encoding information in such a way that only authorized parties can access it.

7. Role-based Access Control (RBAC): A method of restricting system access to authorized users based on their roles within an organization.

8. GDPR: General Data Protection Regulation, a regulation in EU law on data protection and privacy for all individuals within the European Union and the European Economic Area.

9. User Interface (UI): The point of human-computer interaction and communication in a device, software, or application.

10. API (Application Programming Interface): A set of rules and protocols for building and interacting with software applications.

11. Real-time: Refers to systems that process data as it arrives, allowing for immediate responses.

12. Fail-safe: A mechanism designed to prevent or mitigate the consequences of system failure or error.

13. Internationalization: The process of designing a software application to adapt to different languages, cultures, and locales.

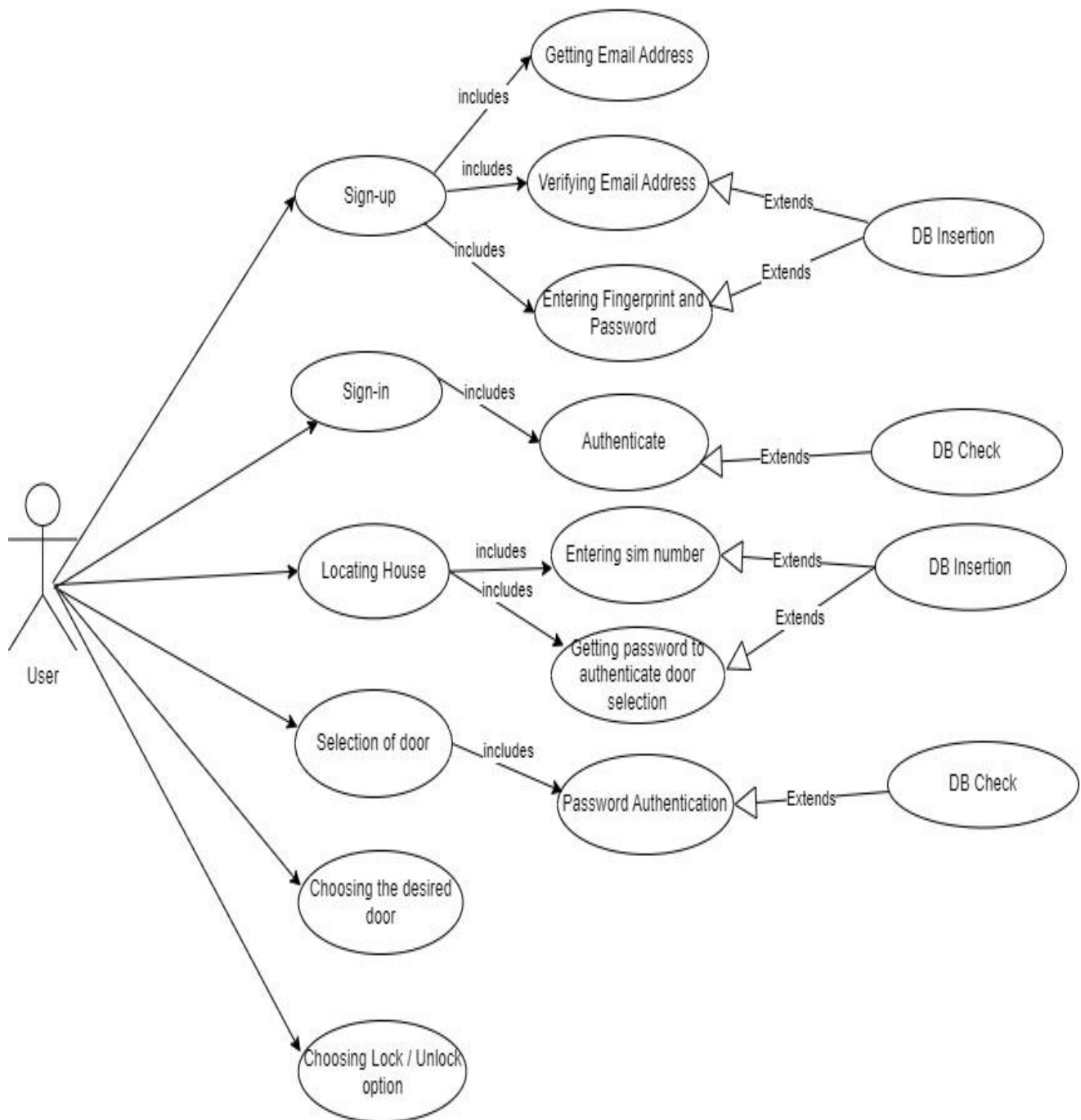
14. Localization: The process of adapting a software application for a specific region or language, including translation and cultural customization.

15. Scalability: The ability of a system, network, or process to handle growing amounts of work or adapt to accommodate increased demand.

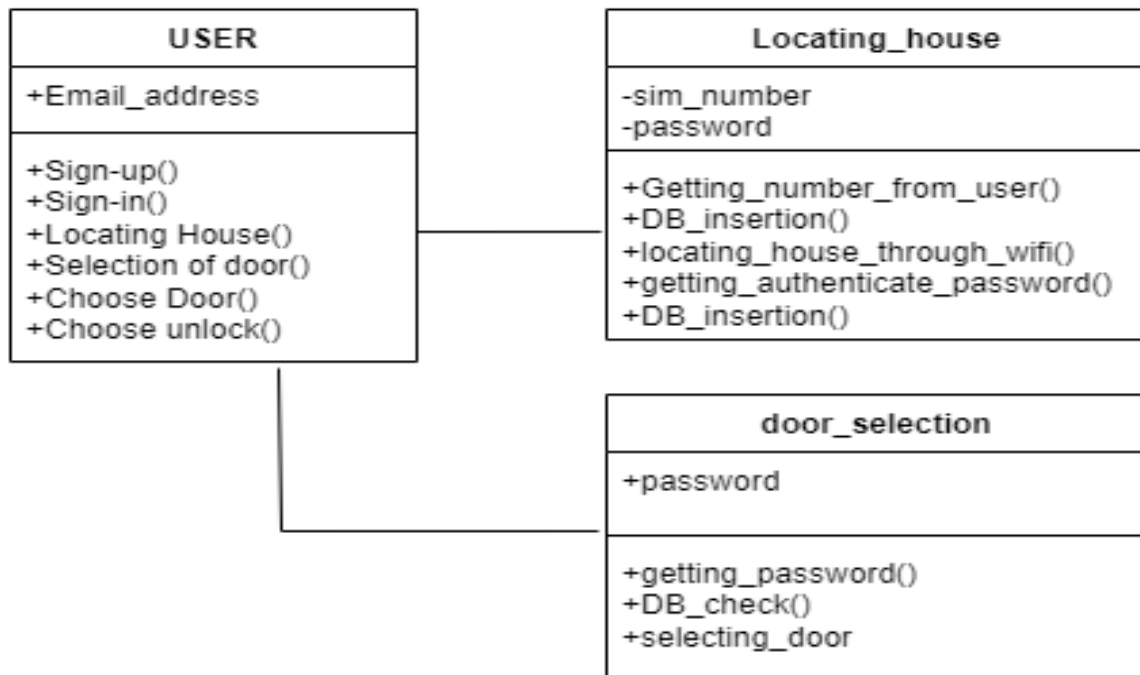
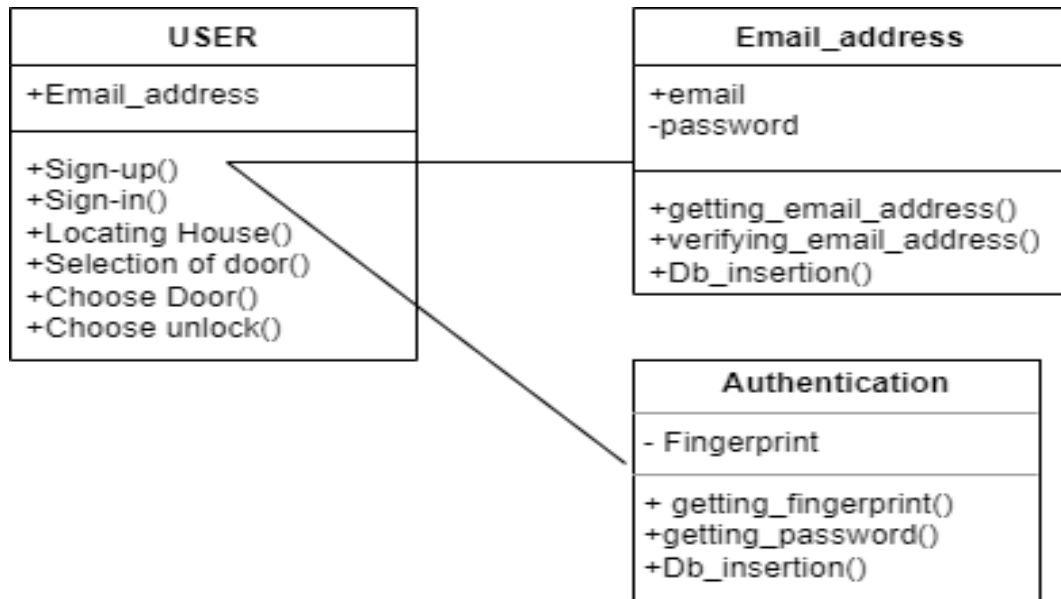
16. Legal Compliance: Adherence to laws, regulations, and standards relevant to the development and deployment of the system.

8. Analysis Models

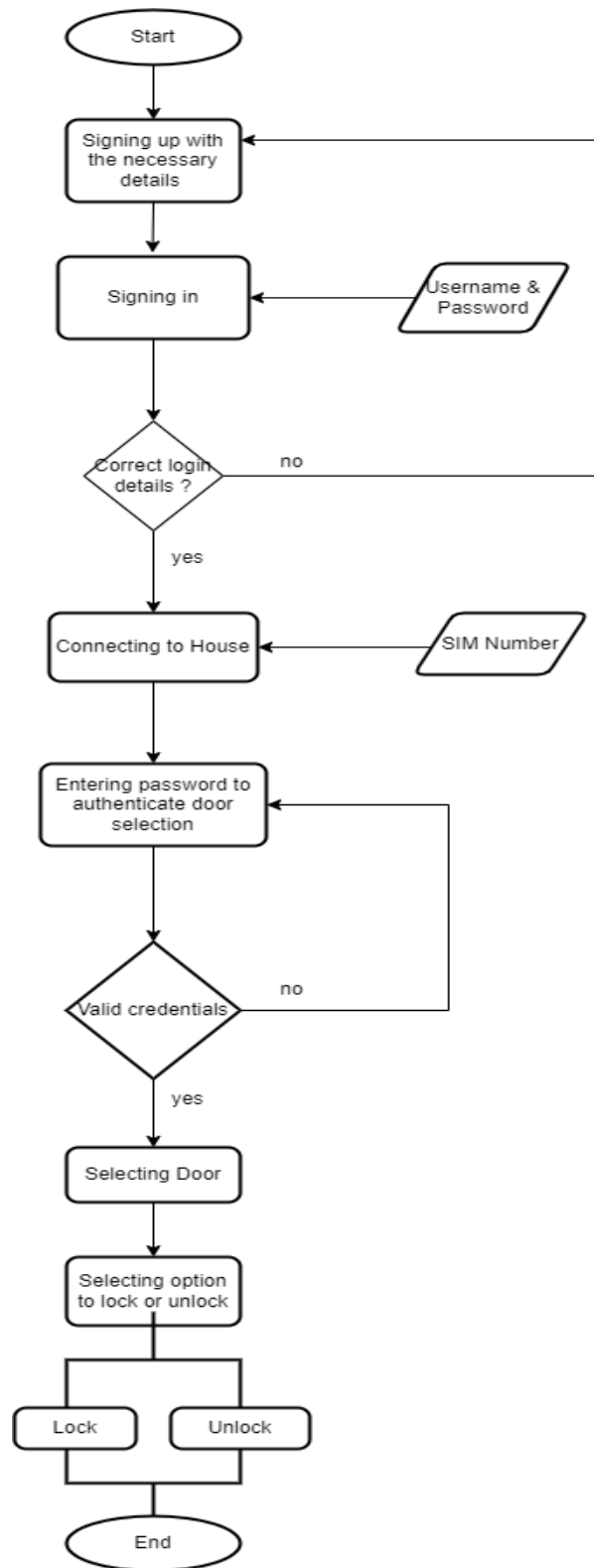
Use case diagram:



Class Diagram:



Activity Diagram:



9. Complexity Analysis

COCOMO MODEL

Code Size (KLOC): 2KLOC (for simplicity since it's a small app)

Organic: Effort Multiplier (E) = 2.4, Size Exponent (B) = 1.05

Semi-detached: Effort Multiplier (E) = 3.0, Size Exponent (B) = 1.12

Embedded: Effort Multiplier (E) = 3.6, Size Exponent (B) = 1.20

1. Organic:

$$\text{Effort} = 2.4 * (2)^{1.05} \approx 6.64 \text{ PM (Person-Month)}$$

$$\text{Development Time (Tdev)} = 2.5 * (\text{Effort})^{0.38} \approx 6.74 \text{ Months}$$

2. Semi-detached:

$$\text{Effort} = 3.0 * (2)^{1.12} \approx 8.13 \text{ PM}$$

$$\text{Development Time (Tdev)} = 2.5 * (\text{Effort})^{0.35} \approx 7.17 \text{ Months}$$

3. Embedded:

$$\text{Effort} = 3.6 * (2)^{1.20} \approx 10.37 \text{ PM}$$

$$\text{Development Time (Tdev)} = 2.5 * (\text{Effort})^{0.32} \approx 7.90 \text{ Months}$$

FUNCTIONAL POINT ANALYSIS

External Inputs (EI): 3 (assuming input for motor control)

External Outputs (EO): 2 (one for each button)

External Inquiries (EQ): 1 (for status inquiry)

Internal Logical Files (ILF): 1

External Interface Files (EIF): 1

$$FP = (EI * 4) + (EO * 5) + (EQ * 4) + (ILF * 7) + (EIF * 5)$$

$$FP = (3 * 4) + (2 * 5) + (1 * 4) + (1 * 7) + (1 * 5)$$

$$FP = (12) + (10) + (4) + (7) + (5)$$

$$FP = 38$$

PERT CHART

Activity	Description	Predecessors	Duration	Optimistic Time	Pessimistic Time	Most Likely Time	Mean	Variance
A	Procure Components	None	2	1	3	2	2	0.1667
B	Develop App	A	4	3	5	4	4	0.3333
C	Integrate Hardware & Software	A, B	3	2	4	3	3.3333	0.25
D	Test & Debug System	C	2	1	3	2	2.3333	0.1667
E	Deploy & Document	D	1	0.5	1.5	1	1	0.25

DESCRIPTION:

1. Procure Components-Bluetooth module, motors, Arduino, motor driver IC
2. Develop App-Login, door selection, password, open/close controls
3. Integrate Hardware & Software-Connect app to Arduino, motors
4. Test & Debug System-Functionality, security
5. Deploy & Document-Installation instructions, user guide

- Duration (Weeks Estimated)
- Optimistic Time (Weeks)
- Pessimistic Time (Weeks)
- Most Likely Time (Weeks)

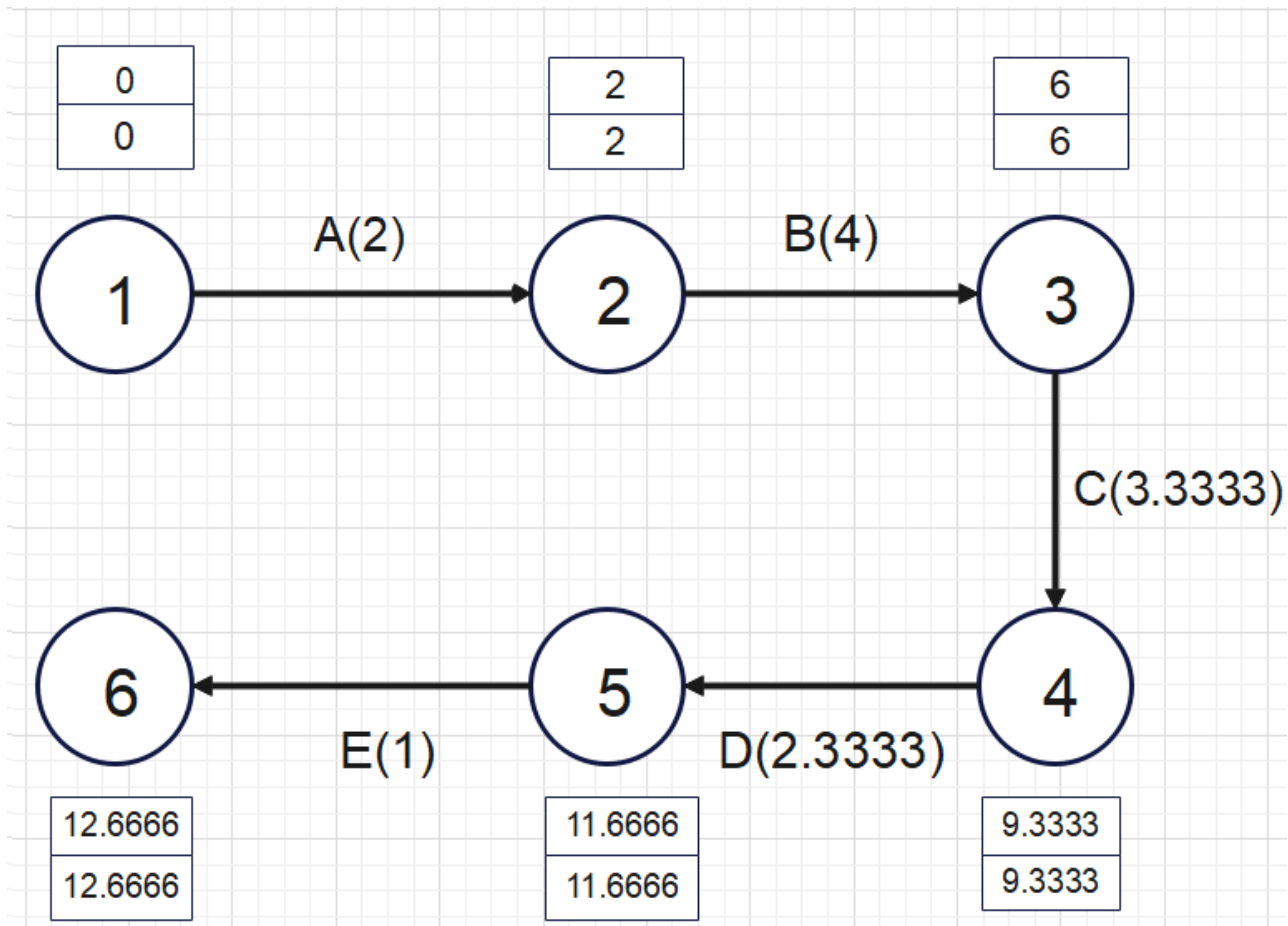
Critical Path

The critical path is the longest path through the PERT chart, determining the minimum project completion time. In this case, the critical path is A -> B -> C -> D -> E, with a total expected time of 12.67 weeks.

Mean and Variance Calculation

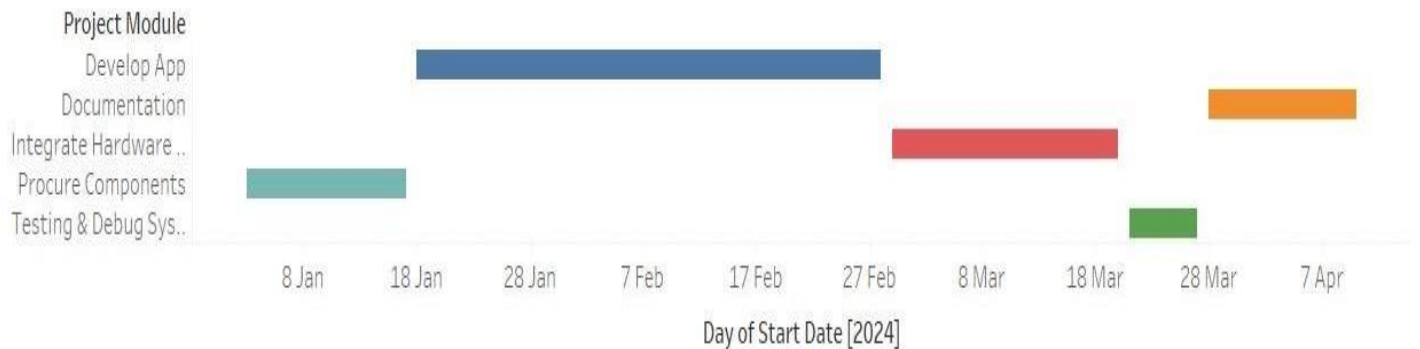
Mean (Expected Project Duration): $\Sigma(TE) = 2 + 4 + 3.3333 + 2.3333 + 1 = 12.67$ weeks (as shown in the table)

Variance (Project Duration Variability): $\Sigma(VE) = 0.1667 + 0.3333 + 0.2500 + 0.1667 + 0.2500 = 1.1667$ weeks²



GANTT CHART:

	A	B	C
1	Project Module	Start Date	End Date
2	Procure Components	2024-01-03	2024-01-17
3	Develop App	2024-01-18	2024-02-28
4	Integrate Hardware & Software	2024-02-29	2024-03-20
5	Testing & Debug System	2024-03-21	2024-03-27
6	Documentation	2024-03-28	2024-04-10



10. Blackbox Testing

- To test the login functionality by entering valid credentials and ensuring that the user is successfully authenticated if not the user cannot login into the app and verify that appropriate error messages are displayed for invalid credentials.

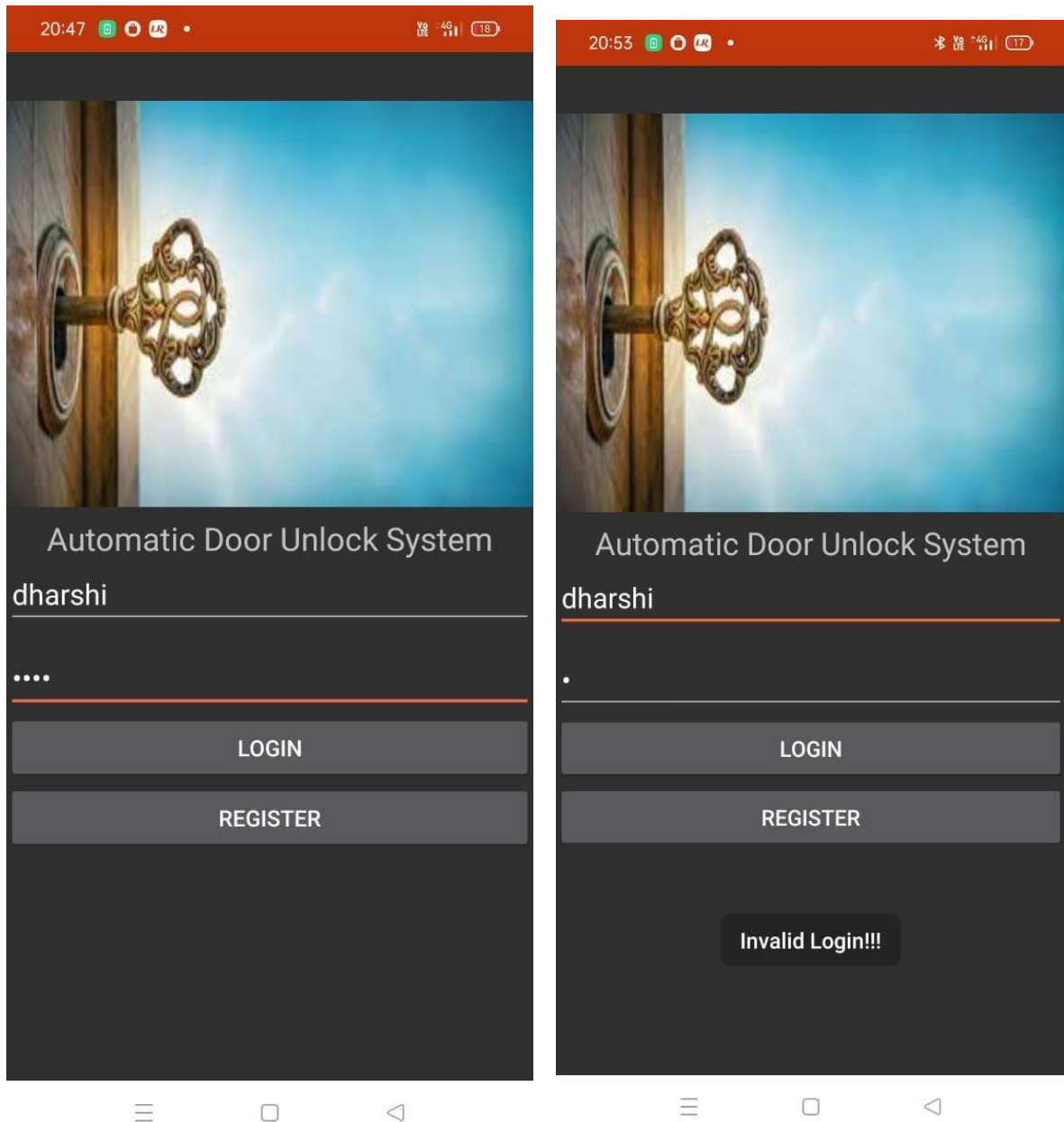


Fig 1: Login Credentials validating

- To test the registration process by providing valid input data and verifying that a new user account is created successfully.

20:47 4G 18

Register User

Dharshi

....

dharshi

2/1580 Vellammal nagar

8610373247

maha@gmail.com

REGISTER BACK

Sucessfully Registered!!

;

Fig 2: User Registration Successful

- To test the Bluetooth connectivity by ensuring that the app can detect and pair with the Bluetooth module controlling the motor and verifying that the app handles scenarios such as Bluetooth being disabled on the device or no available devices to pair with.

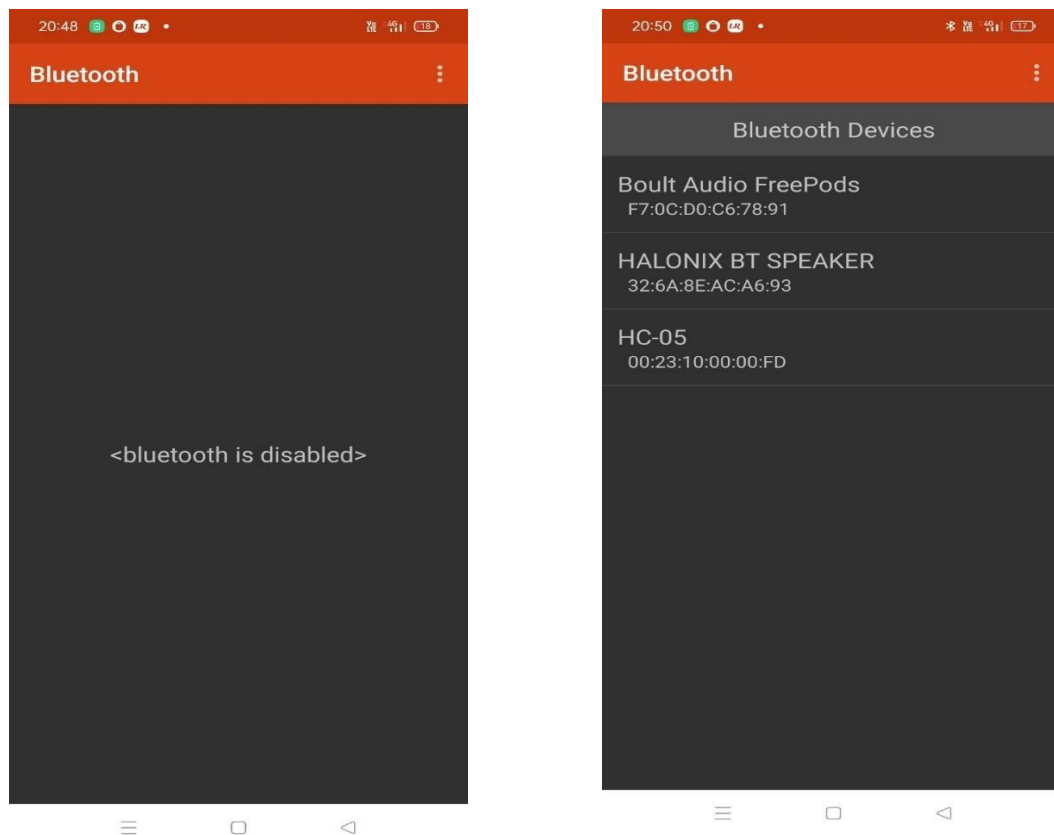


Fig 3: Bluetooth Device Connectivity

- To test the door selection functionality by ensuring that users can choose the door they want to unlock and verifying that only authenticated users can access the door unlocking functionality and testing different authentication scenarios, such as correct credentials, incorrect credentials, and unauthorized access attempts, and ensure that the system responds appropriately in each case.

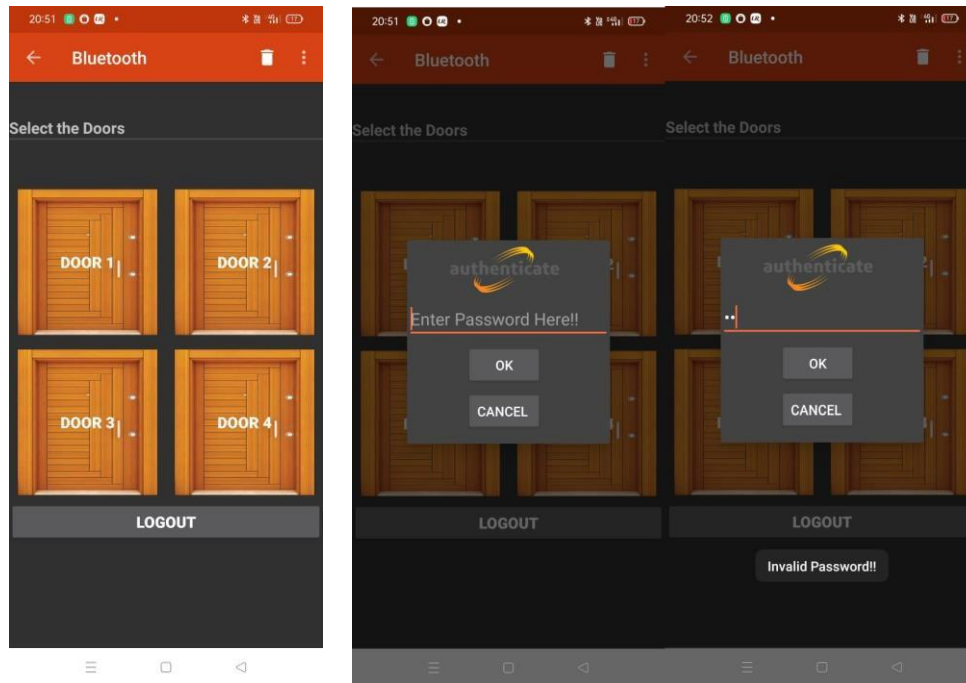


Fig 4: Authentication Of user successful

To test the motor control functionality by sending commands from the app to the motor and verifying that the motor responds accordingly and verifying that the motor only unlocks the door when a valid command is received from the app and testing error handling scenarios, such as the motor not responding or malfunctioning, and ensure that the app handles these situations gracefully.

To test the integration between different components of the system, such as the app, Bluetooth module, and motor control logic and verifying that data is transmitted correctly between the app and the motor, and that actions performed in the app result in the desired outcomes on the motor. To test the security of the system by attempting to bypass authentication or send unauthorized commands to the motor and verifying that the system properly protects against unauthorized access and that sensitive is handled securely.

Blackbox Test Cases

Test Cases	Expected Output	Failed/Passed
Login- Valid Credential	Login into app	Passed
Login- Invalid Credential	Login Failed	Passed
Bluetooth Connectivity	Connecting to available device	Passed
Door selection with valid credential	Motor rotates	Passed
Door selection with invalid credential	Motor does not rotates	Passed
First time door selection	Motor rotates Anticlockwise	Passed
Second time door selection	Motor rotates clockwise	Passed

11.OUTPUTS

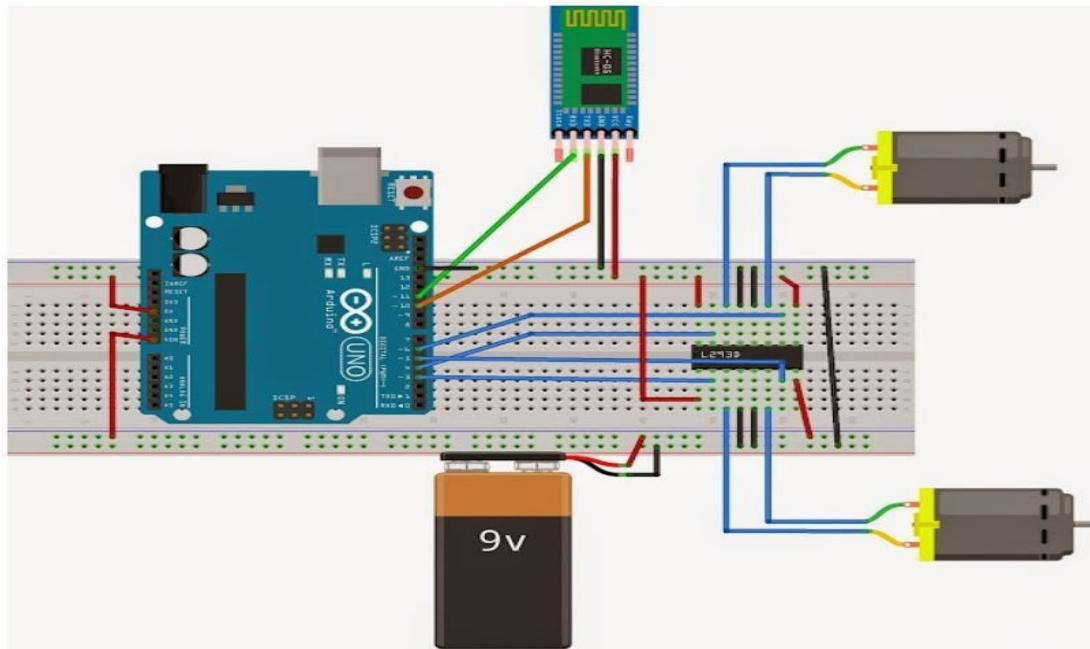


Fig 1: Hardware Setup Connection Diagram

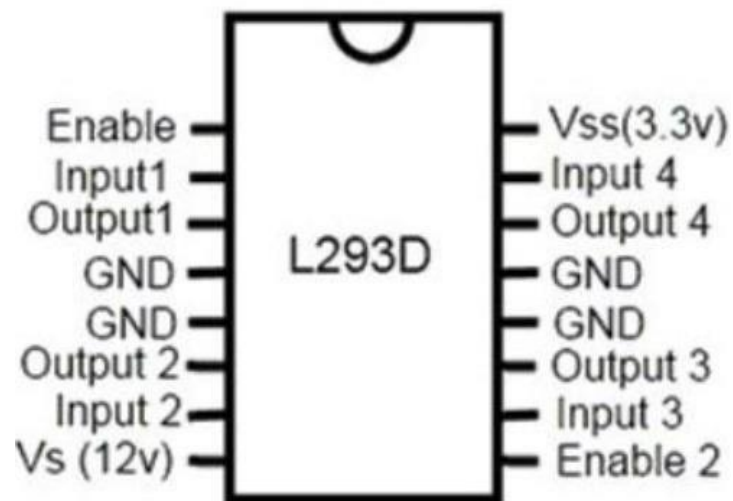


Fig 2: Pin diagram of motor driver used

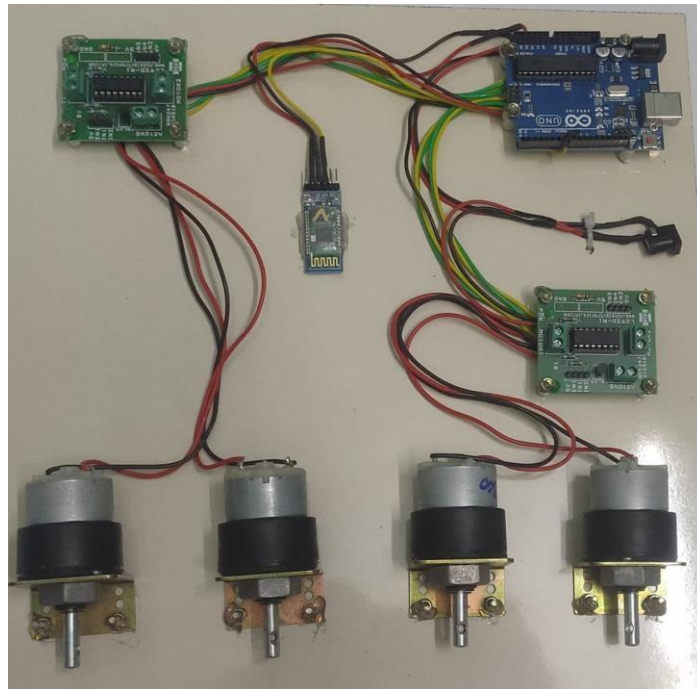


Fig 3: Hardware Setup

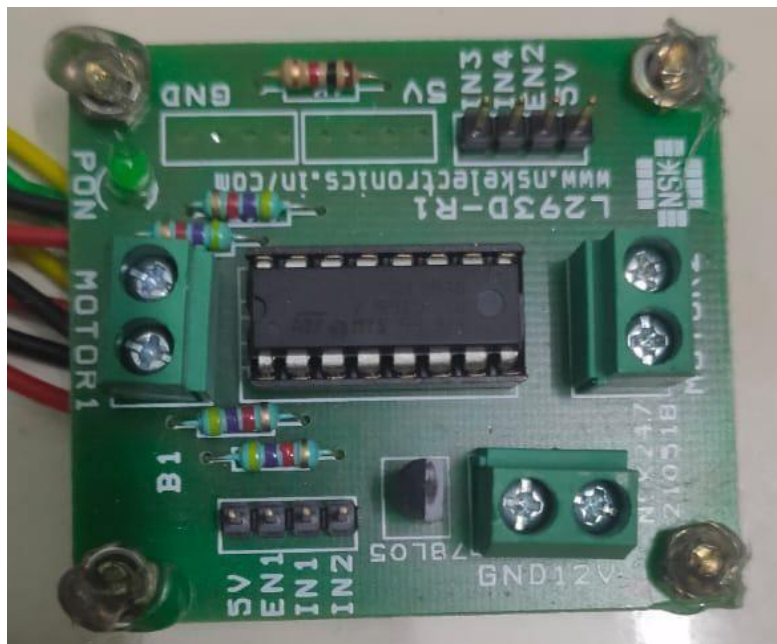


Fig 4: L293D Motor Driver

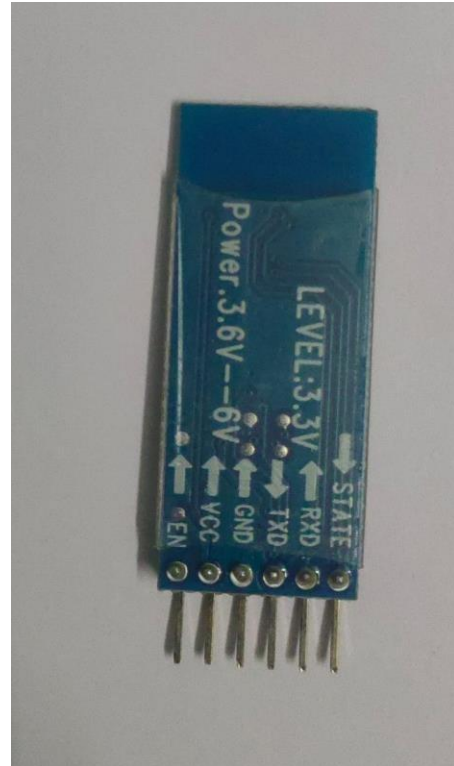
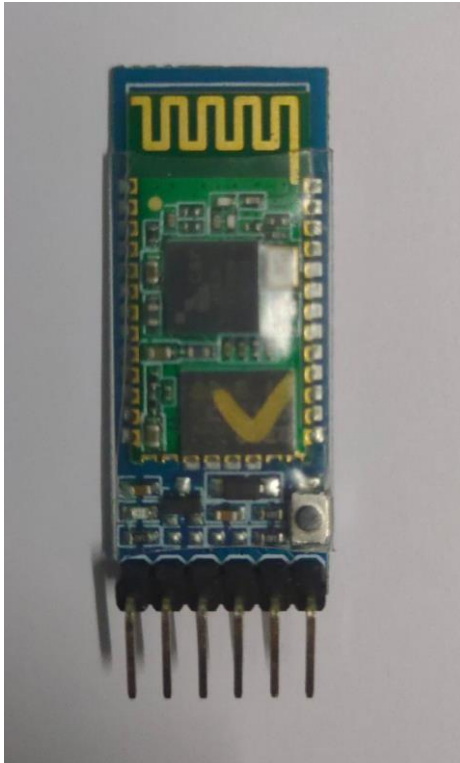


Fig 5: Bluetooth Module



Fig 6: Gear Motor

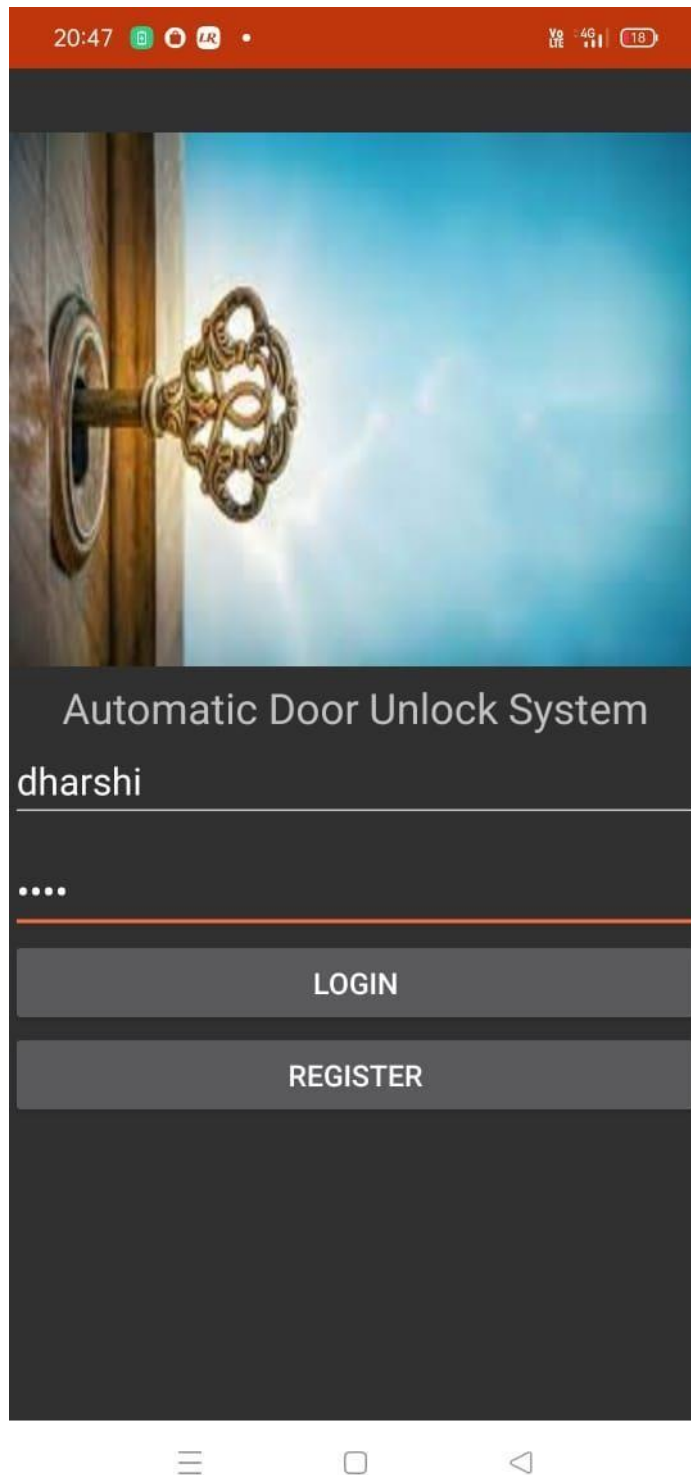


Fig 7: Login Screen

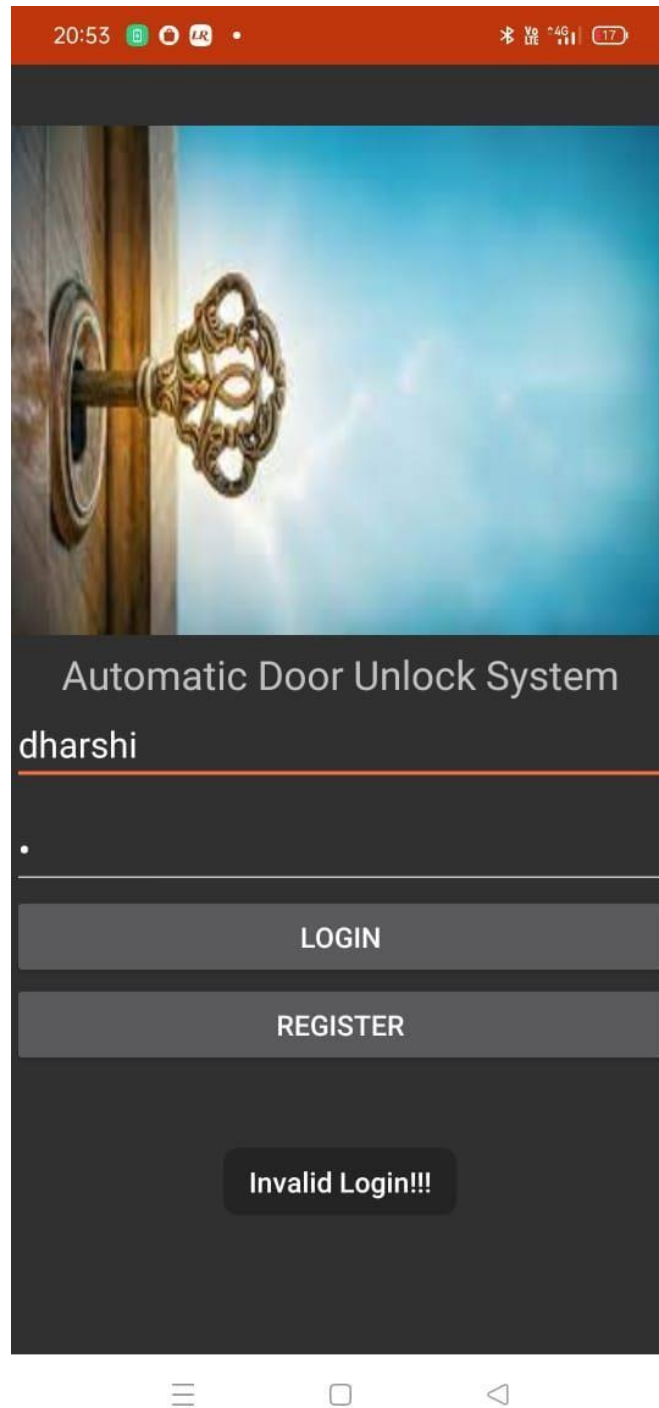
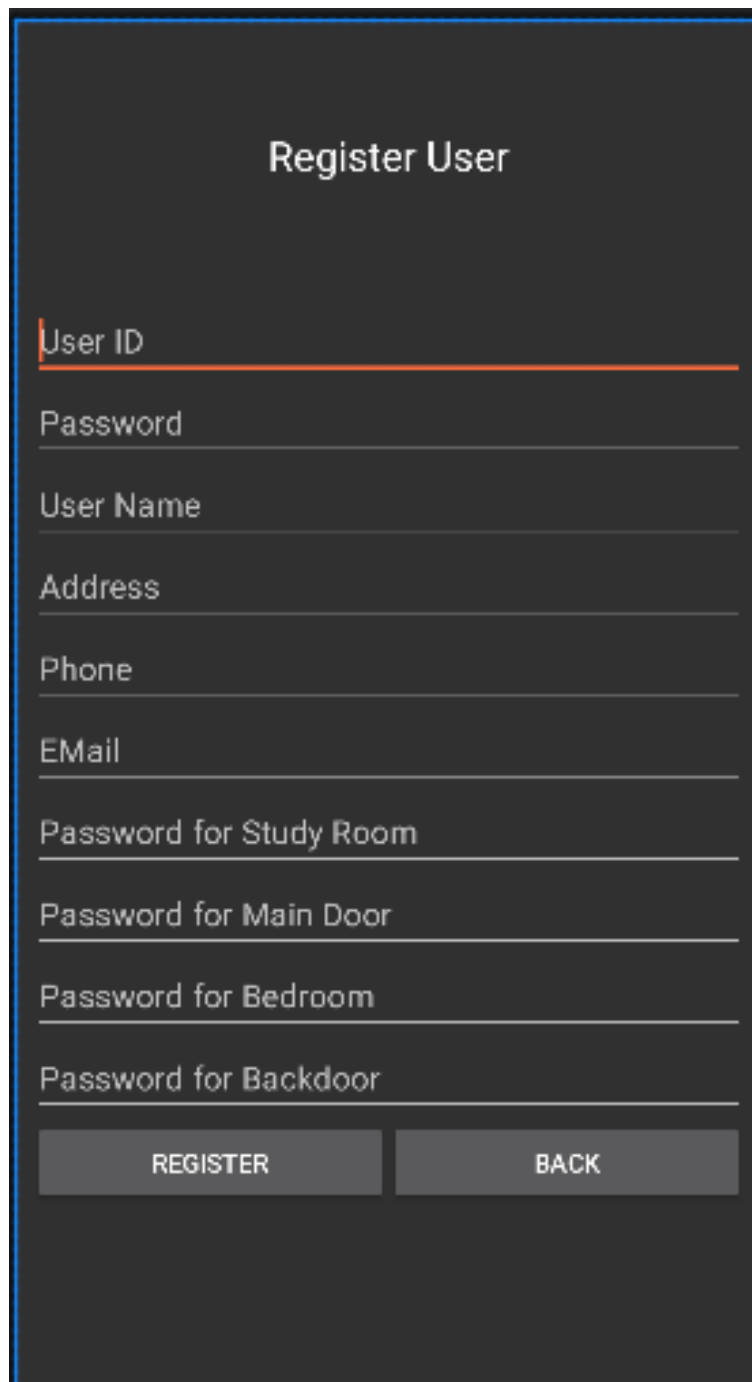


Fig 8: Invalid Login

A dark-themed user registration form titled "Register User". The form contains several input fields with labels: "User ID", "Password", "User Name", "Address", "Phone", "EMail", "Password for Study Room", "Password for Main Door", "Password for Bedroom", and "Password for Backdoor". The "User ID" field is highlighted with an orange underline. At the bottom, there are two buttons: "REGISTER" and "BACK".

Register User

User ID

Password

User Name

Address

Phone

EMail

Password for Study Room

Password for Main Door

Password for Bedroom

Password for Backdoor

REGISTER BACK

Fig 9: User Registration Page

The image shows a mobile application interface for user registration. At the top, there is a status bar with the time 20:47, signal strength, and battery level. The main title is "Register User". Below the title, there are several input fields: a name field containing "Dharshi", a password field with four dots, a username field containing "dharshi", an address field containing "2/1580 Vellammal nagar", a phone number field containing "8610373247", and an email field containing "maha@gmail.com". Below these fields are two buttons: "REGISTER" and "BACK". At the bottom, there is a dark grey button with the text "Sucessfully Registered!!". The bottom of the screen shows the standard Android navigation bar with three icons: a hamburger menu, a home button, and a back button.

20:47 VoLTE 4G 18

Register User

Dharshi

....

dharshi

2/1580 Vellammal nagar

8610373247

maha@gmail.com

REGISTER BACK

Sucessfully Registered!!

Fig 10: Successful Registration

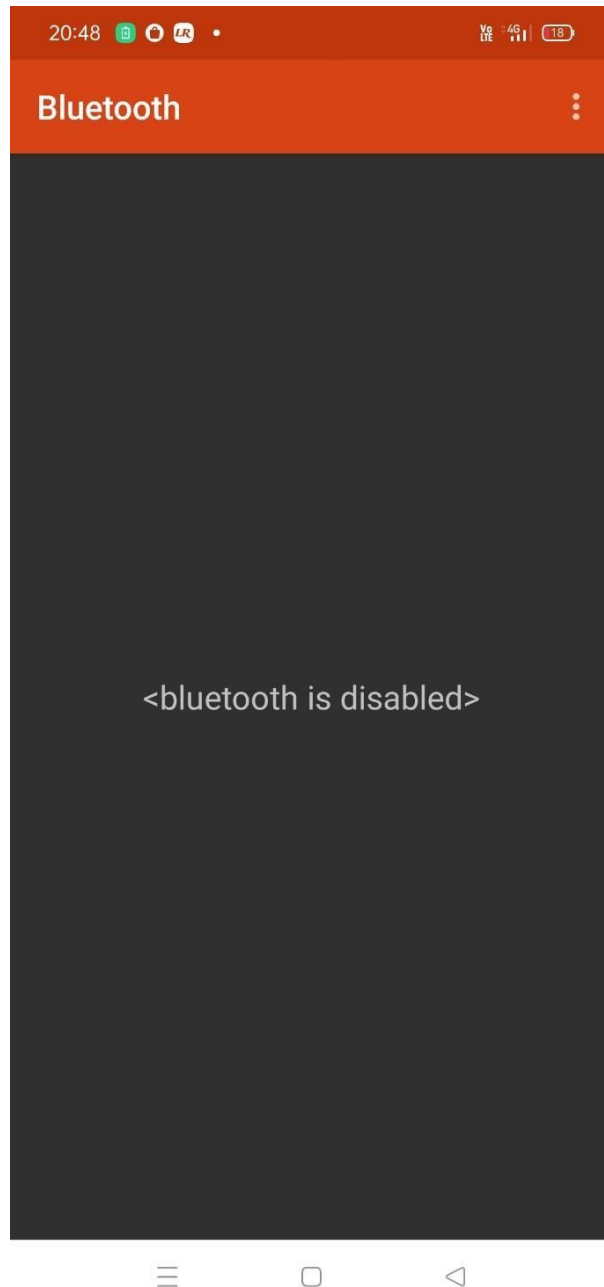


Fig 11: At initial before Bluetooth Connection

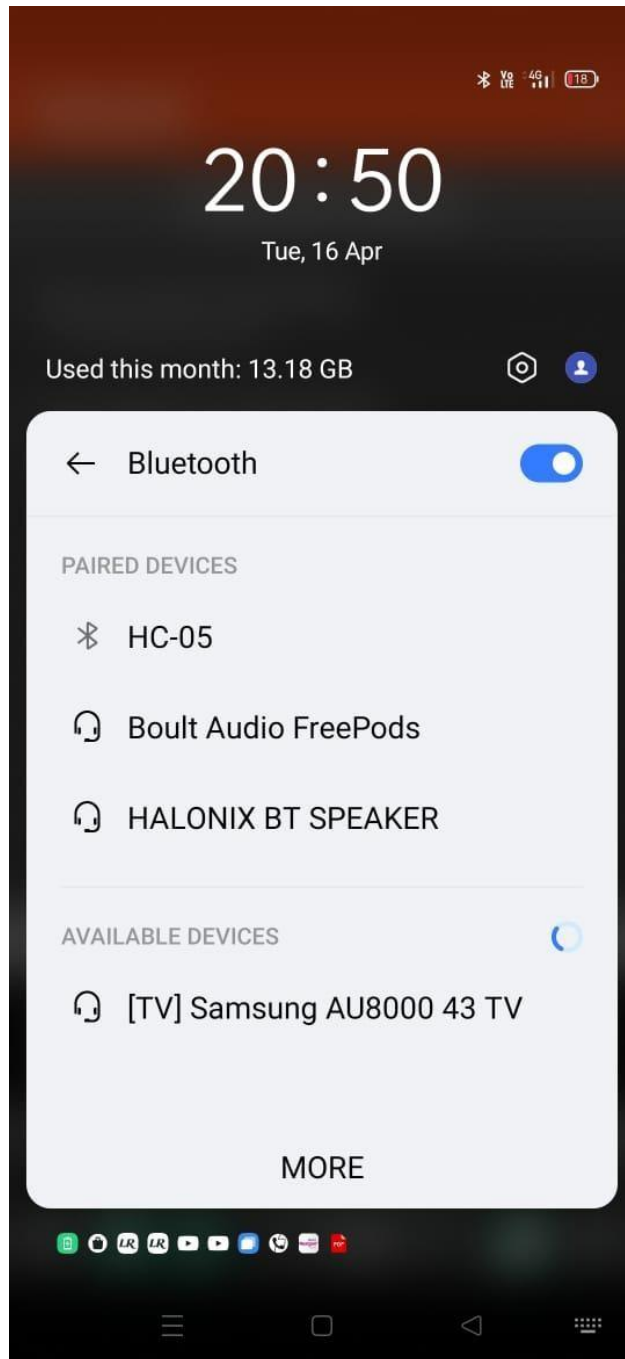


Fig 12.1: Connecting to Bluetooth

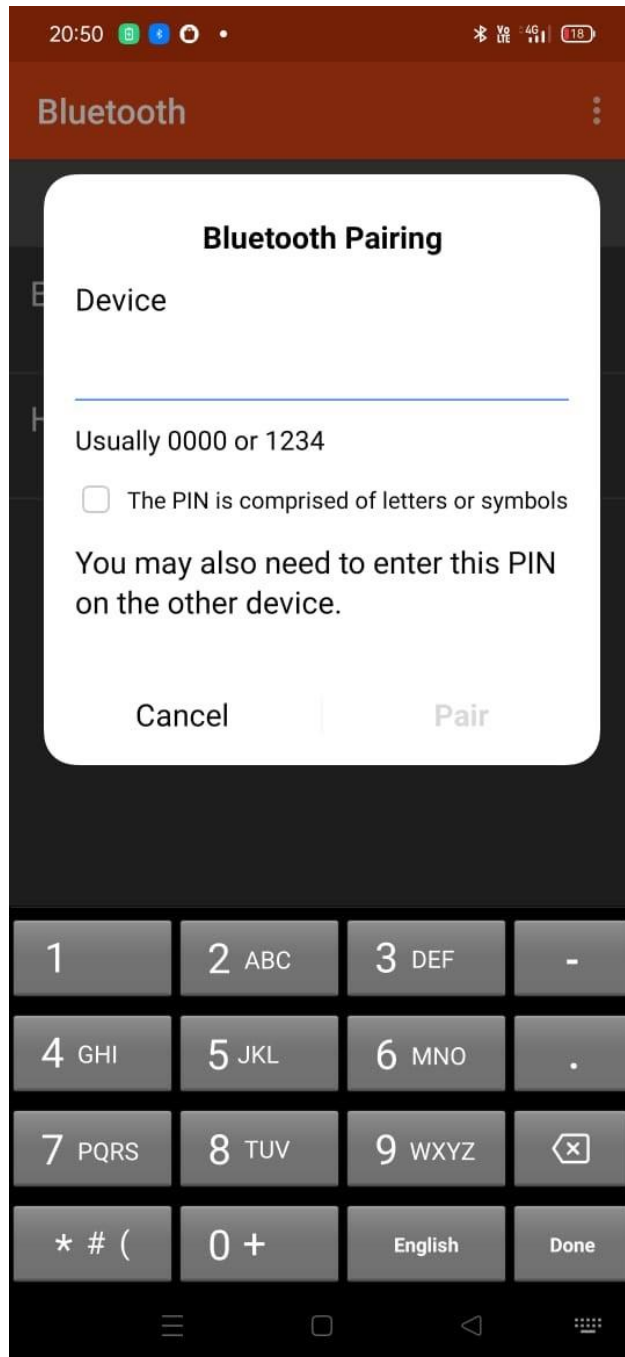


Fig 12.2: Connecting to Bluetooth

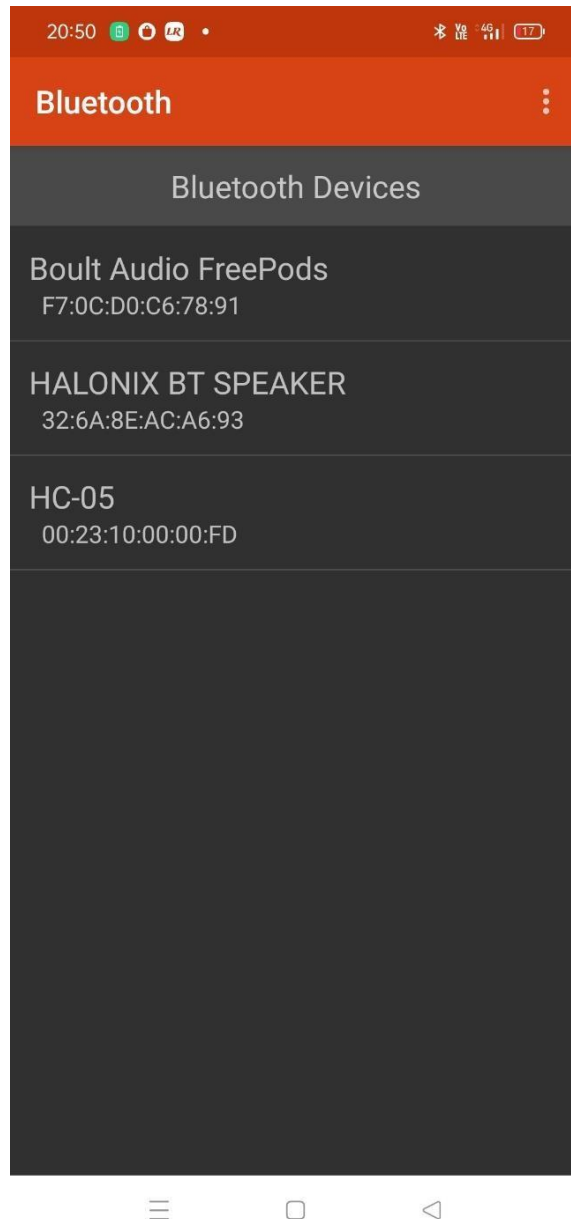


Fig 13: After Connecting to Bluetooth



Fig 14: Screen to select doors

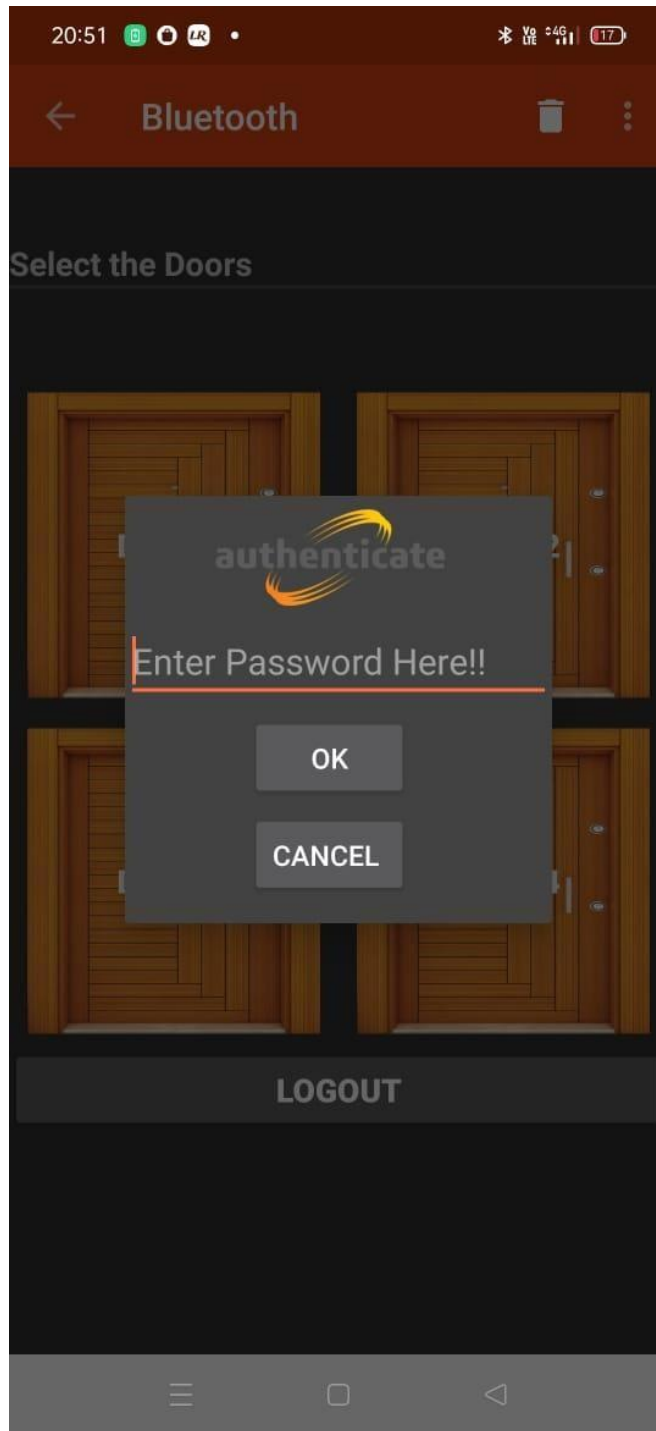


Fig 15: Authenticating user to unlock door

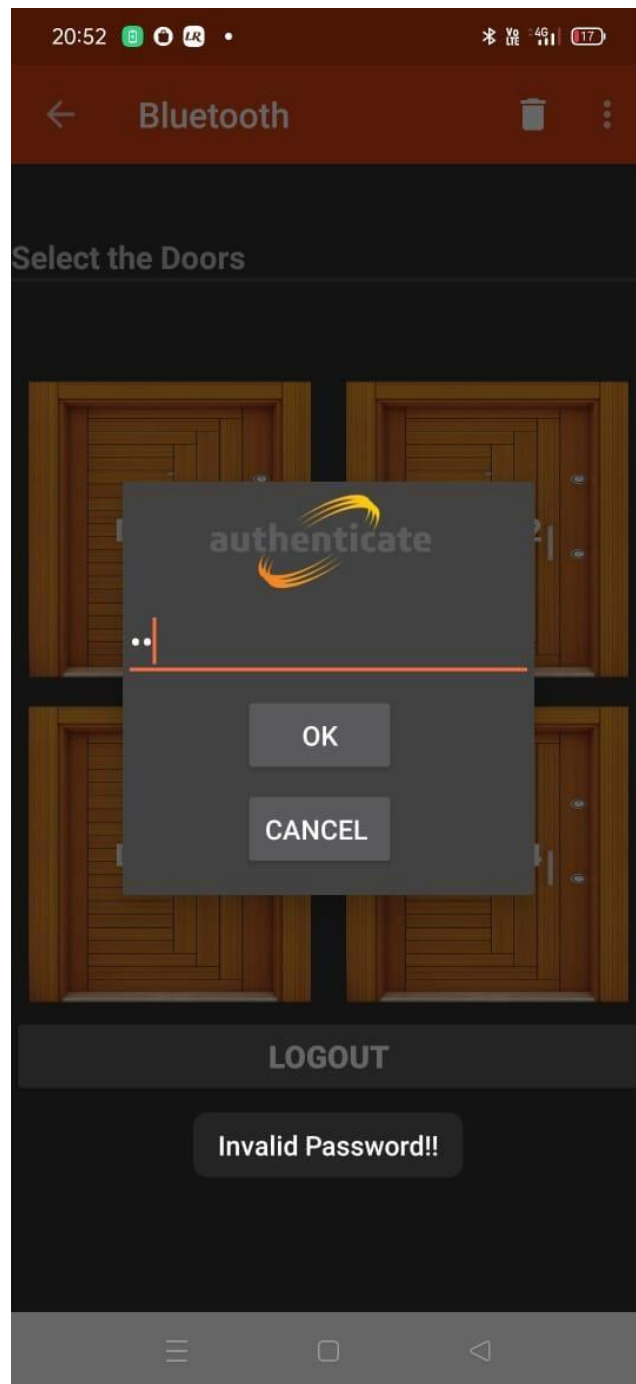


Fig 16: Invalid Password

Appendix : Code

Arduino Implementation

```
#include<SoftwareSerial.h>

// Define Bluetooth module pins
#defineBT_RX8
#defineBT_TX9

// Define motor control pins

#defineMOTOR1_IN1 A0
#defineMOTOR1_IN2 A1

#defineMOTOR2_IN1 A2
#defineMOTOR2_IN2 A3

#defineMOTOR3_IN12
#defineMOTOR3_IN23

#defineMOTOR4_IN14
#defineMOTOR4_IN25

SoftwareSerialbluetooth(BT_RX, BT_TX);

voidsetup(){
  // Set motor control pins as output

  pinMode(MOTOR1_IN1, OUTPUT);
  pinMode(MOTOR1_IN2, OUTPUT);

  pinMode(MOTOR2_IN1, OUTPUT);
  pinMode(MOTOR2_IN2, OUTPUT);

  pinMode(MOTOR3_IN1, OUTPUT);
  pinMode(MOTOR3_IN2, OUTPUT);

  pinMode(MOTOR4_IN1, OUTPUT);
  pinMode(MOTOR4_IN2, OUTPUT);

  digitalWrite(MOTOR1_IN1, LOW);
  digitalWrite(MOTOR1_IN2, LOW);

  digitalWrite(MOTOR2_IN1, LOW);
  digitalWrite(MOTOR2_IN2, LOW);

  digitalWrite(MOTOR3_IN1, LOW);
  digitalWrite(MOTOR3_IN2, LOW);

  digitalWrite(MOTOR4_IN1, LOW);
  digitalWrite(MOTOR4_IN2, LOW);
```

```

// Set Bluetooth baud rate
bluetooth.begin(9600);
}

void loop(){
  if(bluetooth.available() > 0){
    char command = bluetooth.read();
    // Motor 1 control
    if(command == '1'){
      digitalWrite(MOTOR1_IN1, HIGH);
      digitalWrite(MOTOR1_IN2, LOW); delay(30000);
      digitalWrite(MOTOR1_IN1, LOW);
      digitalWrite(MOTOR1_IN2, LOW);

    }
    elseif(command == '2'){
      digitalWrite(MOTOR1_IN1, LOW);
      digitalWrite(MOTOR1_IN2, HIGH); delay(30000);
      digitalWrite(MOTOR1_IN1, LOW);
      digitalWrite(MOTOR1_IN2, LOW);
    }
    else{

    }

    // Motor 2 control
    if(command == '3'){

      digitalWrite(MOTOR2_IN1, HIGH);
      digitalWrite(MOTOR2_IN2, LOW); delay(30000);
      digitalWrite(MOTOR2_IN1, LOW);
      digitalWrite(MOTOR2_IN2, LOW);
    }
    elseif(command == '4'){

      digitalWrite(MOTOR2_IN1, LOW);
      digitalWrite(MOTOR2_IN2, HIGH); delay(30000);
      digitalWrite(MOTOR2_IN1, LOW);
      digitalWrite(MOTOR2_IN2, LOW);
    }
    else{

    }

    // Motor 3 control
    if(command == '5'){

      digitalWrite(MOTOR3_IN1, HIGH);
      digitalWrite(MOTOR3_IN2, LOW); delay(30000);
      digitalWrite(MOTOR3_IN1, LOW);
      digitalWrite(MOTOR3_IN2, LOW);
    }
    elseif(command == '6'){

```

```

        digitalWrite(MOTOR3_IN1, LOW);
        digitalWrite(MOTOR3_IN2, HIGH);delay(30000);
        digitalWrite(MOTOR3_IN1, LOW);
        digitalWrite(MOTOR3_IN2, LOW);
    }
    else{

    }

    // Motor 4 control
    if(command == '7'){

        digitalWrite(MOTOR4_IN1, HIGH);
        digitalWrite(MOTOR4_IN2, LOW);delay(30000);
        digitalWrite(MOTOR4_IN1, LOW);
        digitalWrite(MOTOR4_IN2, LOW);
    }
    elseif(command == '8'){

        digitalWrite(MOTOR4_IN1, LOW);
        digitalWrite(MOTOR4_IN2, HIGH);delay(30000);
        digitalWrite(MOTOR4_IN1, LOW);
        digitalWrite(MOTOR4_IN2, LOW);
    }
    else{

    }
}
}
}

```

Android App Implementation

Login.java

```
package de.kai_morich.simple_bluetooth_terminal;

import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.os.StrictMode;
import android.annotation.SuppressLint;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Login extends AppCompatActivity {

    public Button b1,b3;
    public String login;
    public EditText e1,e2;

    SQLiteDatabase con;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);

        b3=(Button)findViewById(R.id.loginbutton);
        b1=(Button)findViewById(R.id.button1);
```

```

        e1=(EditText)findViewById(R.id.username);
        e2=(EditText)findViewById(R.id.password);

        con = this.openOrCreateDatabase("BluetoothMotor", MODE_PRIVATE,
null);

        String CREATE_reg_TABLE = "CREATE TABLE IF NOT EXISTS
user('userid' TEXT,'pwd' TEXT,'username' TEXT,'address' TEXT,'phone'
TEXT,'email' TEXT)";

        con.execSQL(CREATE_reg_TABLE);

        String CREATE_loggedin_TABLE = "CREATE TABLE IF NOT EXISTS
loggedin('userid' TEXT,'pwd' TEXT,'username' TEXT,'address' TEXT,'phone'
TEXT,'email' TEXT)";

        con.execSQL(CREATE_loggedin_TABLE);

        //Getting local database values

        Cursor c=con.rawQuery("select * from loggedin",null);

        c.moveToFirst();

        if(c.getCount()>0)
        {

            // Toast.makeText(getApplicationContext(),"Login
Success",Toast.LENGTH_SHORT).show();

            Intent i=new Intent(Login.this,MainActivity.class);

            i.putExtra("userid",c.getString(0));

            i.putExtra("pwd",c.getString(1));

            startActivity(i);

        }

        b1.setOnClickListener(new View.OnClickListener()

        {

            @Override

            public void onClick(View arg)

```



```

        {
startActivity(new Intent(Login.this,Register.class));
        }
    });
    b3.setOnClickListener(new View.OnClickListener()
    {
        @Override
        @SuppressWarnings("NewApi")
        public void onClick(View arg)
        {
            if (android.os.Build.VERSION.SDK_INT >= 9) {
StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
StrictMode.setThreadPolicy(policy);
            }

            //Getting local database values

            Cursor c=con.rawQuery("select * from user where
userid='"+ e1.getText().toString() +"' and pwd='"+
e2.getText().toString() +"'",null);

c.moveToFirst();

            if(c.getCount()>0)
            {
con.execSQL("INSERT INTO loggedin (userid,pwd) Values ('"+
e1.getText().toString() +"', '"+ e2.getText().toString() +"'");

                // Toast.makeText(getApplicationContext(),"Login
Success",Toast.LENGTH_SHORT).show();

```

```

                Intent i=new Intent(Login.this,MainActivity.class);
i.putExtra("userid",c.getString(0));
i.putExtra("pwd",c.getString(1));
startActivity(i);
        }
        else
        {
Toast.makeText(getApplicationContext(),"Invalid
Login!!!",Toast.LENGTH_SHORT).show();
        }
    }
    });
}
}

```

Register.java

```

package de.kai_morich.simple_bluetooth_terminal;

import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import androidx.appcompat.app.AppCompatActivity;

import android.annotation.SuppressLint;
import android.os.Bundle;
import android.os.StrictMode;

```

```

import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Register extends AppCompatActivity {
    private Button b1,b2;
    public String cal,s,res;
    public EditText e1,e2,e3,e4,e5,e6;
    SQLiteDatabase con;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.register);

        e1=(EditText) findViewById(R.id.editText11);
        e2=(EditText) findViewById(R.id.editText1);
        e3=(EditText) findViewById(R.id.editText2);
        e4=(EditText) findViewById(R.id.editText3);
        e5=(EditText) findViewById(R.id.editText4);
        e6=(EditText) findViewById(R.id.editText5);

        con = this.openOrCreateDatabase("BluetoothMotor", MODE_PRIVATE,
null);

        b1=(Button)findViewById(R.id.imageButton1);

```

```

        b2=(Button)findViewById(R.id.imageButton2);
        b1.setOnClickListener(new View.OnClickListener() {
            @Override
            @SuppressWarnings("NewApi")
            public void onClick(View v) {
                if (android.os.Build.VERSION.SDK_INT >= 9) {
                    StrictMode.ThreadPolicy policy = new
                    StrictMode.ThreadPolicy.Builder().permitAll().build();
                    StrictMode.setThreadPolicy(policy);
                }

                con.execSQL("INSERT INTO user (userid,pwd,username,address,phone,email)
                Values ('"+ e1.getText().toString() + "','"+ e2.getText().toString()
                + "','"+ e3.getText().toString() + "','"+ e4.getText().toString() + "','"+
                e5.getText().toString() + "','"+ e6.getText().toString() + "')");

                Toast.makeText(getApplicationContext(),"Sucessfully Registered!!",
                Toast.LENGTH_SHORT).show();
            }

        });

        b2.setOnClickListener(new View.OnClickListener() {
            @Override
            @SuppressWarnings("NewApi")
            public void onClick(View v) {
                finish();
            }
        });

```

```
        });  
  
    }  
}
```

TerminalFragment.java

```
package de.kai_morich.simple_bluetooth_terminal;  
  
import static android.content.Context.MODE_PRIVATE;  
import static android.content.Intent getIntent;  
import static  
    android.database.sqlite.SQLiteDatabase.openOrCreateDatabase;  
  
import android.app.Activity;  
import android.app.AlertDialog;  
import android.app.Dialog;  
import android.bluetooth.BluetoothAdapter;  
import android.bluetooth.BluetoothDevice;  
import android.content.ComponentName;  
import android.content.Context;  
import android.content.Intent;  
import android.content.ServiceConnection;  
import android.database.sqlite.SQLiteDatabase;  
import android.graphics.Color;  
import android.os.Build;
```

```
import android.os.Bundle;
import android.os.IBinder;
import android.os.StrictMode;
import android.text.Editable;
import android.text.Spannable;
import android.text.SpannableStringBuilder;
import android.text.method.ScrollingMovementMethod;
import android.text.style.ForegroundColorSpan;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import androidx.activity.result.contract.ActivityResultContracts;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.annotation.RequiresApi;
import androidx.fragment.app.Fragment;
```

```

import java.util.ArrayDeque;

public class TerminalFragment extends Fragment implements
ServiceConnection, SerialListener {

    SQLiteDatabase con;

    private enum Connected { False, Pending, True }

    TextView tv1,tv2;

    Boolean firstClicked = false;
    Boolean secondClicked = false;
    Boolean thirdClicked = false;
    Boolean fourthClicked = false;
    private String deviceAddress;
    private SerialService service;

    private Button btn1,btn2,btn3,btn4,btn5;

    private TextUtil.HexWatcherhexWatcher;

    private Connected connected = Connected.False;
    private booleaninitialStart = true;
    private booleanhexEnabled = false;
    private booleanpendingNewline = false;
    private String newline = TextUtil.newline_crlf;

    /*

```

```

        * Lifecycle
        */

        @Override

        public void onCreate(@Nullable Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setHasOptionsMenu(true);
setRetainInstance(true);
deviceAddress = getArguments().getString("device");

        con = getActivity().openOrCreateDatabase("BluetoothMotor",
MODE_PRIVATE, null);
    }

    @Override

    public void onDestroy() {

        if (connected != Connected.False)
disconnect();

getActivity().stopService(new Intent(getActivity(),
SerialService.class));
super.onDestroy();
    }

    @Override

    public void onStart() {
super.onStart();

if(service != null)
service.attach(this);

```



```

        else

getActivity().startService(new Intent(getActivity(),
SerialService.class)); // prevents service destroy on unbind from
recreated activity caused by orientation change

    }

    @Override

    public void onStop() {

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
if(service != null && !getActivity().isChangingConfigurations())
service.detach();

        }

super.onStop();

    }


    @SuppressWarnings("deprecation") // onAttach(context) was added with
API 23. onAttach(activity) works for all API versions

    @Override

    public void onAttach(@NonNull Activity activity) {

super.onAttach(activity);

getActivity().bindService(new Intent(getActivity(), SerialService.class),
this, Context.BIND_AUTO_CREATE);

    }


    @Override

    public void onDetach() {

```

```
        try { getActivity().unbindService(this); } catch(Exception
ignored) {}
```

```
super.onDetach();
```

```
}
```

```
@Override
```

```
public void onResume() {
```

```
super.onResume();
```

```
if(initialStart&& service != null) {
```

```
initialStart = false;
```

```
getActivity().runOnUiThread(this::connect);
```

```
}
```

```
}
```

```
@Override
```

```
public void onServiceConnected(ComponentName name, IBinder binder) {
```

```
    service = ((SerialService.SerialBinder) binder).getService();
```

```
service.attach(this);
```

```
if(initialStart&&isResumed()) {
```

```
initialStart = false;
```

```
getActivity().runOnUiThread(this::connect);
```

```
}
```

```
}
```

```
@Override
```

```

public void onServiceDisconnected(ComponentName name) {
    service = null;
}

/*
 * UI
 */
@Override
public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {

    View view = inflater.inflate(R.layout.fragment_terminal,
container, false);

    // TextView performance decreases with number of spans

    btn1 = view.findViewById(R.id.button1);
    btn2 = view.findViewById(R.id.button2);
    btn3= view.findViewById(R.id.button3);
    btn4 = view.findViewById(R.id.button4);
    btn5 = view.findViewById(R.id.button5);
    tv1 = view.findViewById(R.id.textView6);
    tv2 = view.findViewById(R.id.textView7);
    Intent ij = getActivity().getIntent();
    tv1.setText(ij.getStringExtra("userid"));

```

```

        tv2.setText(ij.getStringExtra("pwd"));

        btn1.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View view) {

                // custom dialog

                final Dialog dialog = new Dialog(getContext());

dialog.setContentView(R.layout.custom);

dialog.setTitle("Authenticate User...");


                // set the custom dialog components - text, image and
button

                final TextView text = (TextView)
dialog.findViewById(R.id.editText1);

                //text.setText("Android custom dialog example!");

                ImageView image = (ImageView) dialog.findViewById(R.id.image);

                // image.setImageResource(R.drawable.logo);


                Button cancelButton = (Button)
dialog.findViewById(R.id.cancelButton);

                cancelButton.setOnClickListener(new View.OnClickListener() {

                    @Override

                    public void onClick(View v) {

dialog.dismiss();

                    }

                });
            }
        });

```

```

        Button dialogButton = (Button)
dialog.findViewById(R.id.dialogButtonOK);

        // if button is clicked, close the custom dialog
dialogButton.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                if
(text.getText().toString().equals(tv2.getText().toString()))

                {

                    if (firstClicked == false) {

send("1");

firstClicked = true;

Toast.makeText(getContext(),
getResources().getString(R.string.slot11).toString(),
Toast.LENGTH_SHORT).show();

                        } else {

send("2");

firstClicked = false;

Toast.makeText(getContext(),
getResources().getString(R.string.slot12).toString(),
Toast.LENGTH_SHORT).show();

                                }

dialog.dismiss();

                    }

                    else

                    {

Toast.makeText(getContext(),"Invalid Password!!",
Toast.LENGTH_SHORT).show();

```

```

        }

    }

});

dialog.show();

}

});

btn2.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View view) {

        // custom dialog

        final Dialog dialog = new Dialog(getContext());

dialog.setContentView(R.layout.custom);

dialog.setTitle("Authenticate User...");


        // set the custom dialog components - text, image and
button

        final TextView text = (TextView)
dialog.findViewById(R.id.editText1);

        //text.setText("Android custom dialog example!");

ImageView image = (ImageView) dialog.findViewById(R.id.image);

        //  image.setImageResource(R.drawable.logo);


        Button cancelButton = (Button)
dialog.findViewById(R.id.cancelButton);

cancelButton.setOnClickListener(new View.OnClickListener() {

```

```

        @Override
        public void onClick(View v) {
dialog.dismiss();
        }
    });

    Button dialogButton = (Button)
dialog.findViewById(R.id.dialogButtonOK);

    // if button is clicked, close the custom dialog
dialogButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if
(text.getText().toString().equals(tv2.getText().toString()))
            {
                if (secondClicked == false) {

send("3");

secondClicked = true;

Toast.makeText(getApplicationContext(),
getResources().getString(R.string.slot21).toString(),
Toast.LENGTH_SHORT).show();

                } else {

send("4");

secondClicked = false;

Toast.makeText(getApplicationContext(),
getResources().getString(R.string.slot22).toString(),
Toast.LENGTH_SHORT).show();

                }
            }
        }
    });

```

```

dialog.dismiss();

        }

        else

        {

Toast.makeText(getApplicationContext(),"Invalid Password!!",
Toast.LENGTH_SHORT).show();

        }

    }

});

dialog.show();

    }

});

    btn3.setOnClickListener(new View.OnClickListener() {

        @Override

        public void onClick(View view) {

            // custom dialog

            final Dialog dialog = new Dialog(getApplicationContext());

dialog.setContentView(R.layout.custom);

dialog.setTitle("Authenticate User...");


            // set the custom dialog components - text, image and
button

            final TextView text = (TextView)
dialog.findViewById(R.id.editText1);

```



```

        //text.setText("Android custom dialog example!");
ImageView image = (ImageView) dialog.findViewById(R.id.image);
        // image.setImageResource(R.drawable.logo);

        Button cancelButton = (Button)
dialog.findViewById(R.id.cancelButton);
cancelButton.setOnClickListener(new View.OnClickListener() {

        @Override

        public void onClick(View v) {
dialog.dismiss();

        }

});

        Button dialogButton = (Button)
dialog.findViewById(R.id.dialogButtonOK);

        // if button is clicked, close the custom dialog
dialogButton.setOnClickListener(new View.OnClickListener() {

        @Override

        public void onClick(View v) {

                if
(text.getText().toString().equals(tv2.getText().toString()))

                {

                        if (thirdClicked == false) {

send("5");
thirdClicked = true;

```

```

Toast.makeText(getApplicationContext(),
getResources().getString(R.string.slot31).toString(),
Toast.LENGTH_SHORT).show();

                                } else {

send("6");

thirdClicked = false;

Toast.makeText(getApplicationContext(),
getResources().getString(R.string.slot32).toString(),
Toast.LENGTH_SHORT).show();

                                }

dialog.dismiss();

                                }

                                else

                                {

Toast.makeText(getApplicationContext(),"Invalid Password!!",
Toast.LENGTH_SHORT).show();

                                }

                                }

                                });

dialog.show();

                                }

                                });

btn4.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View view) {

        // custom dialog

```

```

        final Dialog dialog = new Dialog(getContext());
dialog.setContentView(R.layout.custom);
dialog.setTitle("Authenticate User...");

        // set the custom dialog components - text, image and
button

        final TextView text = (TextView)
dialog.findViewById(R.id.editText1);

        //text.setText("Android custom dialog example!");
ImageView image = (ImageView) dialog.findViewById(R.id.image);
        // image.setImageResource(R.drawable.logo);

        Button cancelButton = (Button)
dialog.findViewById(R.id.cancelButton);
cancelButton.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

dialog.dismiss();

            }

        });

        Button dialogButton = (Button)
dialog.findViewById(R.id.dialogButtonOK);

        // if button is clicked, close the custom dialog
dialogButton.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

```

```

            if
(text.getText().toString().equals(tv2.getText().toString()))
        {
            if (fourthClicked == false) {

send("7");

fourthClicked = true;

Toast.makeText(getApplicationContext(),
getResources().getString(R.string.slot41).toString(),
Toast.LENGTH_SHORT).show();

            } else {

send("8");

fourthClicked = false;

Toast.makeText(getApplicationContext(),
getResources().getString(R.string.slot42).toString(),
Toast.LENGTH_SHORT).show();

            }

dialog.dismiss();

        }

        else

        {

Toast.makeText(getApplicationContext(),"Invalid Password!!",
Toast.LENGTH_SHORT).show();

        }

    }

});

dialog.show();

}

```

```

    });

    btn5.setOnClickListener(new View.OnClickListener() {

        @Override

        public void onClick(View view) {

con.execSQL("delete from loggedin;");

Toast.makeText(getApplicationContext(),"Logout Successful
!!!",Toast.LENGTH_SHORT).show();

                Intent i = new Intent(getApplicationContext(), Login.class);
startActivity(i);

        }

    });


    return view;

}

@Override

    public void onCreateOptionsMenu(@NonNull Menu menu, MenuInflater
inflater) {

inflater.inflate(R.menu.menu_terminal, menu);

menu.findItem(R.id.hex).setChecked(hexEnabled);

    }


    private void send(String str) {

```

```

if(connected != Connected.True) {
    Toast.makeText(getActivity(), "not connected",
    Toast.LENGTH_SHORT).show();

        return;
    }
    try {
        String msg;
byte[] data;
        if(hexEnabled) {
            StringBuilder sb = new StringBuilder();
            TextUtil.toHexString(sb, TextUtil.fromHexString(str));
            TextUtil.toHexString(sb, newline.getBytes());

            msg = sb.toString();
            data = TextUtil.fromHexString(msg);
        } else {
            msg = str;
            data = (str + newline).getBytes();
        }

        SpannableStringBuilder spn = new SpannableStringBuilder(msg + '\n');
        spn.setSpan(new
        ForegroundColorSpan(getResources().getColor(R.color.colorSendText)), 0,
        spn.length(), Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);

        service.write(data);

        } catch (Exception e) {
onSerialIoError(e);

        }
    }

```

```

    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.clear) {
            return true;
        } else if (id == R.id.newline) {
            String[] newlineNames =
                getResources().getStringArray(R.array.newline_names);
            String[] newlineValues =
                getResources().getStringArray(R.array.newline_values);
            int pos =
                java.util.Arrays.asList(newlineValues).indexOf(newline);
            AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
            builder.setTitle("Newline");
            builder.setSingleChoiceItems(newlineNames, pos, (dialog, item1) -> {
                newline = newlineValues[item1];
            });
            dialog.dismiss();
            builder.create().show();
            return true;
        } else if (id == R.id.hex) {
            hexEnabled = !hexEnabled;
            hexWatcher.enable(hexEnabled);
        }
    }

```

```

item.setChecked(hexEnabled);

        return true;
    } else {
        return super.onOptionsItemSelected(item);
    }
}

/*
 * Serial + UI
 */
private void connect() {
    try {
BluetoothAdapterbluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
BluetoothDevice device = bluetoothAdapter.getRemoteDevice(deviceAddress);
status("connecting...");

        connected = Connected.Pending;

SerialSocket socket = new
SerialSocket(getActivity().getApplicationContext(), device);
service.connect(socket);

        } catch (Exception e) {
onSerialConnectError(e);

        }
}

private void disconnect() {

```



```

        connected = Connected.False;
service.disconnect();
    }

    private void status(String str) {
SpannableStringBuilderspn = new SpannableStringBuilder(str + '\n');
spn.setSpan(new
ForegroundColorSpan(getResources().getColor(R.color.colorStatusText)), 0,
spn.length(), Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);

    }

    /*
     * SerialListener
     */
    @Override
    public void onSerialConnect() {
        status("connected");
        connected = Connected.True;
    }

    @Override
    public void onSerialConnectError(Exception e) {
status("connection failed: " + e.getMessage());
disconnect();
    }

```

```

    }

    @RequiresApi(api = Build.VERSION_CODES.GINGERBREAD)
    @Override
    public void onSerialRead(byte[] data) {
ArrayDeque<byte[]>datas = new ArrayDeque<>();
datas.add(data);

    }

    @Override
    public void onSerialRead(ArrayDeque<byte[]>datas) {

    }

    @Override
    public void onSerialIoError(Exception e) {
status("connection lost: " + e.getMessage());
disconnect();
    }

}

```

BluetoothUtil.java

```

package de.kai_morich.simple_bluetooth_terminal;

import android.Manifest;
import android.annotation.SuppressLint;
import android.app.AlertDialog;
import android.bluetooth.BluetoothDevice;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Build;

import androidx.activity.result.ActivityResultLauncher;
import androidx.fragment.app.Fragment;

public class BluetoothUtil {

    interface PermissionGrantedCallback {
        void call();
    }

    /**
     * sort by name, then address. sort named devices first
     */
    @SuppressWarnings("MissingPermission")

```

```

        static int compareTo(BluetoothDevice a, BluetoothDevice b) {
booleanaValid = a.getName()!=null && !a.getName().isEmpty();
booleanbValid = b.getName()!=null && !b.getName().isEmpty();
if(aValid&&bValid) {
            int ret = a.getName().compareTo(b.getName());
            if (ret != 0) return ret;
            return a.getAddress().compareTo(b.getAddress());
        }
        if(aValid) return -1;
        if(bValid) return +1;
        return a.getAddress().compareTo(b.getAddress());
    }

    /**
     * Android 12 permission handling
     */
    private static void showRationaleDialog(Fragment fragment,
DialogInterface.OnClickListener listener) {
        final AlertDialog.Builder builder = new
AlertDialog.Builder(fragment.getActivity());
        builder.setTitle(fragment.getString(R.string.bluetooth_permission_title))
        ;
        builder.setMessage(fragment.getString(R.string.bluetooth_permission_grant
));
        builder.setNegativeButton("Cancel", null);
        builder.setPositiveButton("Continue", listener);
        builder.show();
    }

```

```

    }

    private static void showSettingsDialog(Fragment fragment) {

        String s =
fragment.getResources().getString(fragment.getResources().getIdentifier("@android:string/permgrouplab_nearby_devices", null, null));

        final AlertDialog.Builder builder = new
AlertDialog.Builder(fragment.getActivity());

builder.setTitle(fragment.getString(R.string.bluetooth_permission_title))
;

builder.setMessage(String.format(fragment.getString(R.string.bluetooth_pe
rmission_denied), s));

builder.setNegativeButton("Cancel", null);

builder.setPositiveButton("Settings", (dialog, which) ->

fragment.startActivity(new
Intent(android.provider.Settings.ACTION_APPLICATION_DETAILS_SETTINGS,
Uri.parse("package:" + BuildConfig.APPLICATION_ID))));

builder.show();

    }

    static boolean hasPermissions(Fragment fragment,
ActivityResultLauncher<String> requestPermissionLauncher) {

if(Build.VERSION.SDK_INT < Build.VERSION_CODES.S)

        return true;

boolean missingPermissions =
fragment.getActivity().checkSelfPermission(Manifest.permission.BLUETOOTH_
CONNECT) != PackageManager.PERMISSION_GRANTED;

boolean showRationale =
fragment.shouldShowRequestPermissionRationale(Manifest.permission.BLUETOOTH_
CONNECT);

```

```

        if(missingPermissions) {
            if (showRationale) {
showRationaleDialog(fragment, (dialog, which) ->

requestPermissionLauncher.launch(Manifest.permission.BLUETOOTH_CONNECT));

            } else {

requestPermissionLauncher.launch(Manifest.permission.BLUETOOTH_CONNECT);

            }

            return false;
        } else {
            return true;
        }
    }
}

```

```

    static void onPermissionsResult(Fragment fragment, boolean granted,
PermissionGrantedCallbackcb) {
if(Build.VERSION.SDK_INT<Build.VERSION_CODES.S)

        return;

booleanshowRationale =
fragment.shouldShowRequestPermissionRationale(Manifest.permission.BLUETOOTH_CONNECT);

        if (granted) {
cb.call();

        } else if (showRationale) {
showRationaleDialog(fragment, (dialog, which) ->cb.call());

```

```
        } else {  
showSettingsDialog(fragment);  
        }  
    }  
  
}
```

12.CONCLUSION

Designing an automatic door/unlock system with Bluetooth connectivity using Arduino and an Android app represents a significant advancement in modern convenience and security. By seamlessly integrating hardware and software, this system offers users enhanced control over access to their spaces while prioritizing ease of use. The utilization of Arduino as the microcontroller ensures robust performance and flexibility in managing various door mechanisms. Meanwhile, the Android app serves as the intuitive interface through which users can remotely operate the system, granting or restricting access as needed. This system's Bluetooth connectivity adds an extra layer of security by enabling authorized access solely through authorized devices, effectively preventing unauthorized entry. Moreover, its compatibility with smartphones ensures widespread accessibility without the need for additional hardware. In conclusion, the automatic door/unlock system with Bluetooth connectivity, powered by Arduino and an Android app, not only streamlines access control but also enhances security, making it a valuable addition to residential, commercial, and industrial environments. Its seamless integration, user-friendly interface, and enhanced security features position it as a reliable solution for modern access management needs.

13.REFERENCES

- <https://www.arduino.cc/>
- <https://www.arduino.cc/en/Reference/Bluetooth>
- <https://www.arduino.cc/en/Reference/BluetoothCommunication>
- <https://developer.android.com/docs>
- <https://developer.android.com/guide/topics/connectivity/bluetooth>
- <https://www.instructables.com/Arduino-Bluetooth-Door-Lock/>
- <https://learn.adafruit.com/introduction-to-bluetooth-low-energy>
- <https://stackoverflow.com/questions/tagged/arduino+android+bluetooth>