Mini project report on

Smart Home Automation Management System

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE22CS351A – DBMS Project

Submitted by:

Jova Varghese	**PES2UG22CS239**
Janvi Munshi	**PES2UG22CS231**

Under the guidance of
Prof. Nivedita Kasturi
Assistant Professor
PES University

AUG - DEC 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Smart Home Automation Management System

is a bonafide work carried out by

Jova Varghese	**PES2UG22CS239**
Janvi Munshi	**PES2UG22CS231**

In partial fulfilment for the completion of fifth semester DBMS Project (UE22CS351A) in the Program of Study -Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2024 – DEC. 2024. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

Signature
Prof. Nivedita Kasturi
Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Smart Home Automation Management System** has been carried out by us under the guidance of **Prof. Nivedita Kasturi, Assistant Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2024.

Jova Varghese **PES2UG22CS239**

Janvi Munshi **PES2UG22CS231**

ABSTRACT

The **Smart Home Automation Management System** is a database-driven project designed to facilitate the seamless control and monitoring of smart devices within a home environment. Developed by Jova Varghese and Janvi Munshi, the system enables users to automate device actions, track maintenance schedules, and log device usage durations. With a Python-Tkinter interface for ease of use, the system relies on MySQL for robust data storage and Git for version control, making it a secure, real-time solution. User roles and permissions provide administrators with full device control and automation management, while regular users can view devices and logs. This project enhances home automation by enabling users to manage and monitor devices and create and update automation rules for device operations.

Key functionalities include user authentication, device management, maintenance logging, and reporting features. Through the use of structured tables and foreign key constraints, the database effectively organizes information related to users, devices, automations, logs, and maintenance. Triggers are implemented to automatically log device usage, and SQL functions support secure user authentication. Additionally, nested and aggregate queries are used to check data integrity and generate usage summaries, offering users a comprehensive view of device activities and maintenance needs. By leveraging Python and MySQL, the project combines real-time control with reliable data storage, making it a practical tool for efficient smart home management.

TABLE OF CONTENTS

I. INTRODUCTION

The Smart Home Automation Management System is a software application designed to enhance the convenience, control, and efficiency of managing smart devices within a home environment. With the rise of IoT (Internet of Things) technologies, homes have become increasingly equipped with connected devices—such as lights, thermostats, cameras, and appliances—that can be controlled remotely. While individual devices often come with their own interfaces or apps, integrating them into a unified system can improve the user experience by providing centralized control, real-time monitoring, and automation.

The primary goal of this project is to create an intuitive, functional, and secure platform where users can not only control their smart devices but also automate routines, track usage data, and manage device maintenance. This application enables users to define automation rules, such as scheduling device operations, setting conditions for device states, and adjusting settings based on time or user preferences. Additionally, the system records device usage statistics and maintains logs for analyzing trends and optimizing power consumption, contributing to a smarter, more efficient home ecosystem.

To support these features, the application uses a relational database to store and manage data. The system's backend, built in Python, connects with MySQL to handle CRUD (Create, Read, Update, Delete) operations for data related to devices, users, automation schedules, and maintenance records. Python's Tkinter library provides a graphical user interface, making it easy for users to interact with the system. Key functionalities include user authentication, automated control and scheduling of devices, device usage logging, and maintenance tracking.

This system has practical applications in real-world smart home setups, where the complexity and number of devices necessitate a robust solution for automation and monitoring. By utilizing IoT protocols such as MQTT or HTTP for device communication, this system is also designed to be adaptable to a range of smart devices and configurations. Furthermore, the use of MySQL as the backend database ensures data integrity and scalability, while Git and GitHub aid in version control, promoting collaborative development and code management.

Overall, the Smart Home Automation Management System combines the latest in smart home technology with efficient database management, enabling users to maintain control over their home environment in a structured, convenient, and user-friendly manner. This project not only demonstrates effective use of software development techniques but also highlights the potential of IoT and automation in enhancing everyday life.

II. PROBLEM DEFINITION WITH USER REQUIREMENT SPECIFICATIONS

1. Core Functionality

- **User Authentication**
  - Secure login system with unique IDs
  - Role-based access (Admin: full control, Regular User: view-only)
  - User management and permission controls
- **Device & Automation Management**
  - Remote device control (on/off capabilities)
  - Automation scheduling and rule creation
  - Real-time device status monitoring
  - Usage tracking and statistics
  - Automated maintenance scheduling (yearly)
- **Monitoring & Reporting**
  - Device usage statistics and analysis
  - Maintenance log tracking
  - Automated report generation
  - Historical data access

2. Technical Implementation

- **Architecture**
  - Frontend: Python Tkinter GUI
  - Backend: Python 3.x
  - Database: MySQL (5 main tables)
  - IoT Integration: MQTT/HTTP protocols
- **Database Structure**
  - Users: credentials, roles, permissions
  - Devices: status, details, ownership
  - Automation: schedules, rules
  - Logs: usage history, events
  - Maintenance: service records, schedules

3. System Requirements & Constraints

- **Hardware Needs**
  - Server: 4GB RAM, 2 CPU cores, 100GB storage
  - Client: Python-compatible device
  - IoT devices: Protocol compatibility required
- **Limitations**
  - Optimized for home/small business use
  - Requires stable internet connection
  - Limited to IoT-compatible devices
  - May need scaling for larger deployments

III. LIST OF SOFTWARES/TOOLS/PROGRAMMING LANGUAGES USED

Python

Python serves as the primary programming language for the backend, managing device control, database connections, and automation functionalities. Its versatility and extensive library support make it ideal for handling various aspects of the system, from logic processing to data manipulation.

Tkinter

Tkinter is utilized to create the graphical user interface (GUI) for the Smart Home Automation Management System. It provides users with an interactive and intuitive interface to control devices, set schedules, and monitor maintenance logs easily.

MySQL

MySQL is the relational database management system (RDBMS) used to store and manage all data, including user information, device settings, automation rules, and maintenance logs. Its structured data model and SQL-based queries make it efficient for handling large datasets securely and reliably.

MySQL Workbench

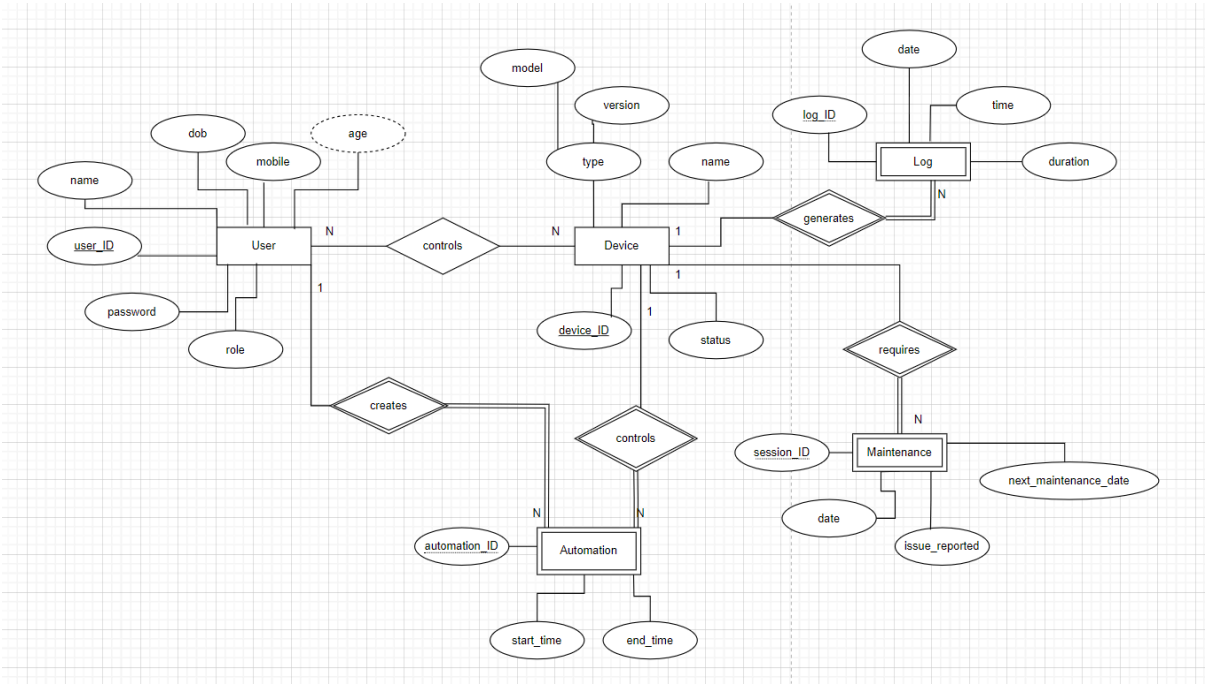
MySQL Workbench is used to design, visualize, and manage the MySQL database, allowing the team to perform database administration tasks, write complex queries, and ensure efficient data structuring and integrity throughout the project.

Git and GitHub

Git is the version control system, and GitHub is the platform used for collaborative development, enabling team members to track code changes, manage versions, and coordinate efficiently. This setup ensures organized and error-free project development with full history and rollback capabilities.

IV. ER MODEL

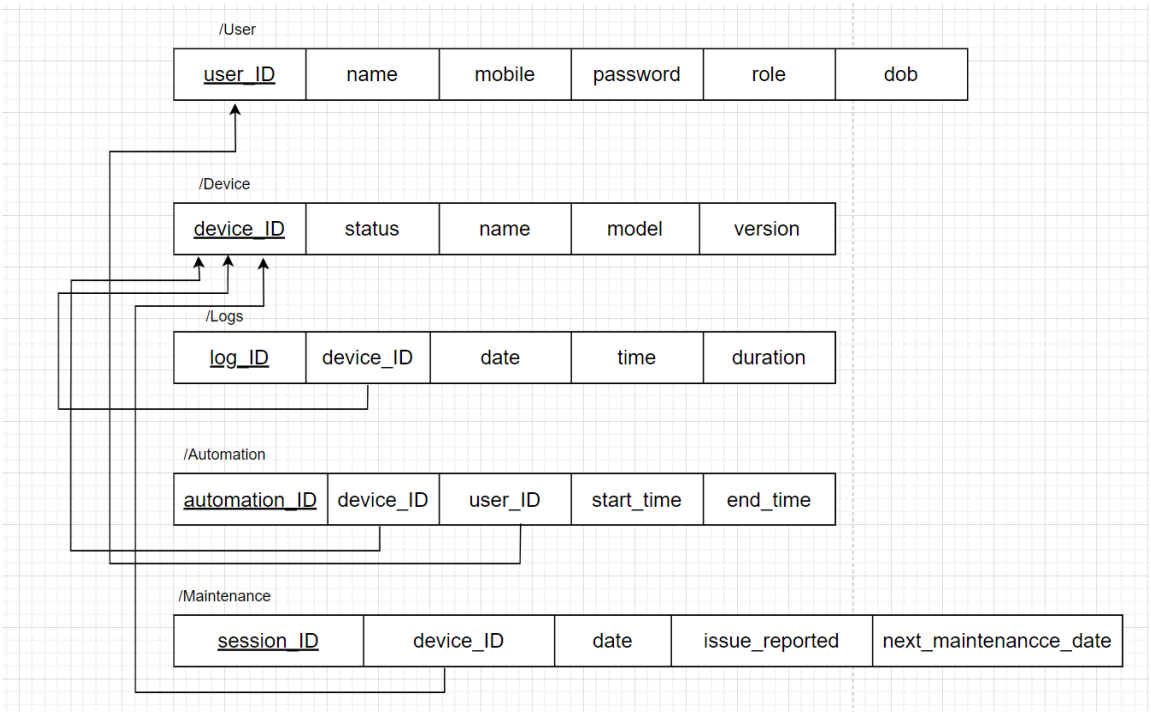
ER Diagram for Smart Home Automation Management System:



The ERD depicts a comprehensive device management and automation system centered around five main entities: User, Device, Automation, Log, and Maintenance. Users, identified by unique user_IDs and characterized by personal details like name and role, have the ability to control multiple devices and create various automations. Devices, tracked through device_IDs and attributes like model and status, serve as the central entity connecting to all other components in the system.

This interconnected system enables devices to generate activity logs, undergo maintenance sessions, and participate in automation routines. The relationship structure reveals a focus on device control and monitoring, where users can create automated processes that involve multiple devices, while maintaining detailed records of device activities through logs and scheduled maintenance. This design suggests a system built for managing smart devices or industrial equipment where tracking usage, automating operations, and maintaining equipment are crucial functionalities.

V. ER TO RELATIONAL MAPPING



This relational database schema outlines a device management and automation system with five primary tables: User, Device, Logs, Automation, and Maintenance.

K**ey Relationships in the Database:**

1. **User-Device Connection**
    - One user can control multiple devices
    - Each device has a single controlling user
    - Implemented through user_ID as foreign key in Device table
2. **Device-Log Association**
    - Each device generates multiple log entries
    - Every log entry belongs to one specific device
    - Tracked via device_ID in Logs table
3. **Device-Automation Link**
    - Devices can participate in multiple automations
    - Each automation can involve multiple devices
    - Connected through device_ID in Automation table
4. **Device-Maintenance Tracking**
    - Single device can have multiple maintenance records
    - Each maintenance record belongs to one device
    - Linked using device_ID in Maintenance table

VI. DDL STATEMENTS

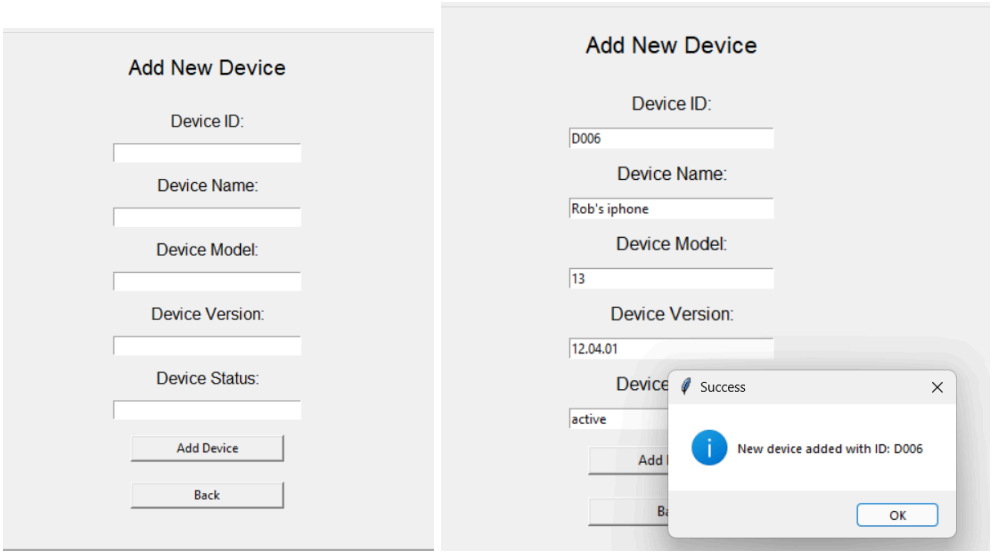
```
CREATE DATABASE smart_home;
USE smart_home;

CREATE TABLE User (
    user_ID VARCHAR(10) PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    mobile VARCHAR(15),
    password VARCHAR(100) NOT NULL,
    role VARCHAR(20),
    dob DATE
);

CREATE TABLE Device (
    device_ID VARCHAR(10) PRIMARY KEY,
    status VARCHAR(20),
    name VARCHAR(50),
    model VARCHAR(50),
    version VARCHAR(20)
);

CREATE TABLE Logs (
    log_ID INT AUTO_INCREMENT PRIMARY KEY,
    device_ID VARCHAR(10),
    date DATE,
    time TIME,
    duration INT,
    FOREIGN KEY (device_ID) REFERENCES Device(device_ID)
);

CREATE TABLE Automation (
    automation_ID VARCHAR(10) PRIMARY KEY,
    device_ID VARCHAR(10),
    user_ID VARCHAR(10),
    start_time TIME,
    end_time TIME,
    FOREIGN KEY (device_ID) REFERENCES Device(device_ID),
    FOREIGN KEY (user_ID) REFERENCES User(user_ID)
);
```

```
CREATE TABLE Maintenance (
    session_ID VARCHAR(10) PRIMARY KEY,
    device_ID VARCHAR(10),
    date DATE,
    issue_reported TEXT,
    next_maintenance_date DATE,
    FOREIGN KEY (device_ID) REFERENCES Device(device_ID)
);
```

These DDL commands will create the necessary tables (Device, User, Automation, Logs, Maintenance) in our MySQL database. Each table is designed to store relevant data for the management of smart home automation, such as device details, user authentication, automation schedules, device usage logs, and maintenance records. The FOREIGN KEY constraints ensure data integrity and maintain the relationships between the tables.

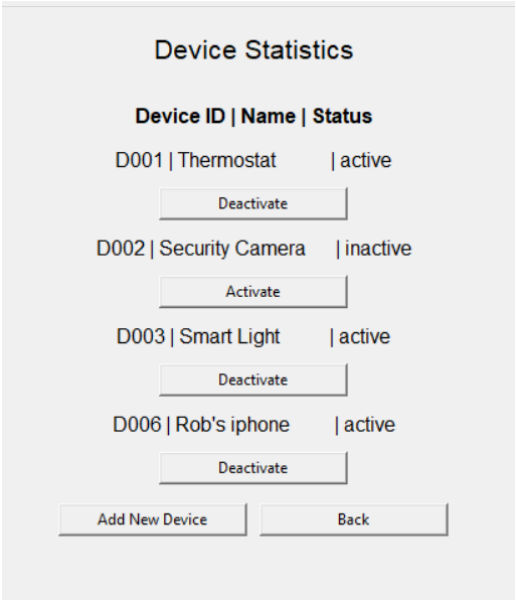
VII. DML STATEMENTS (CRUD OPERATION SCREENSHOTS)

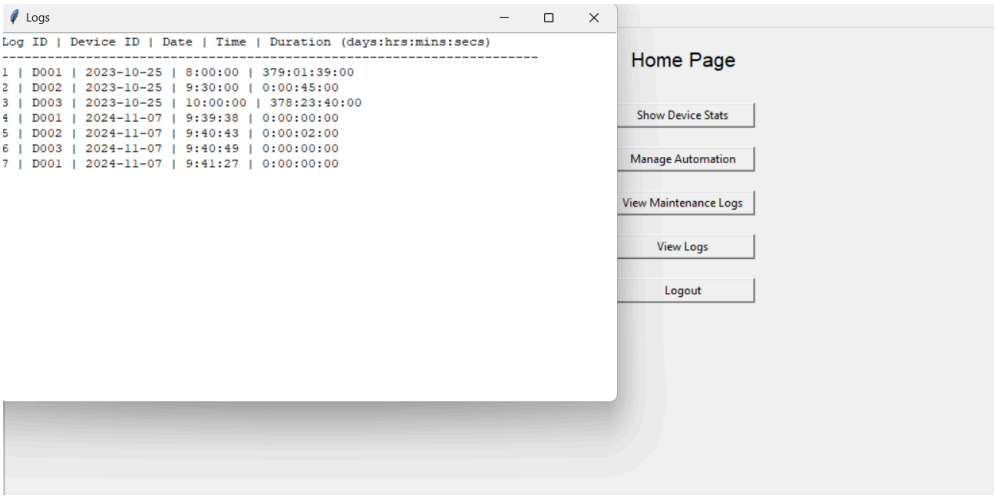
1. CREATE:



Adding a new device to the home system

2. READ:

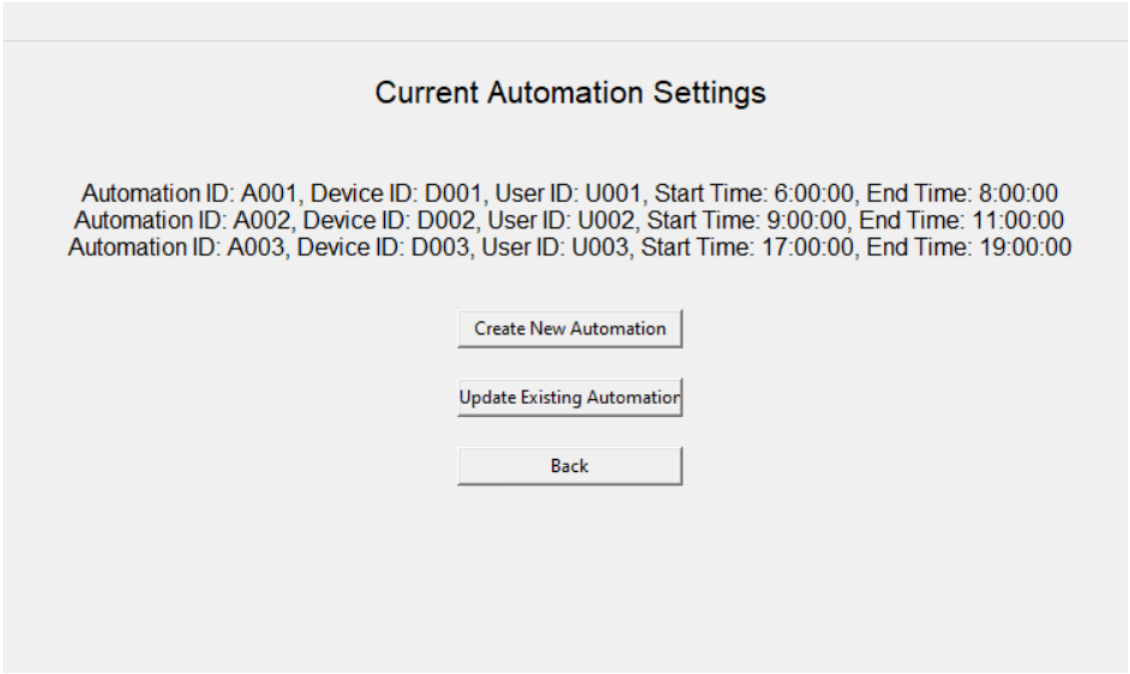


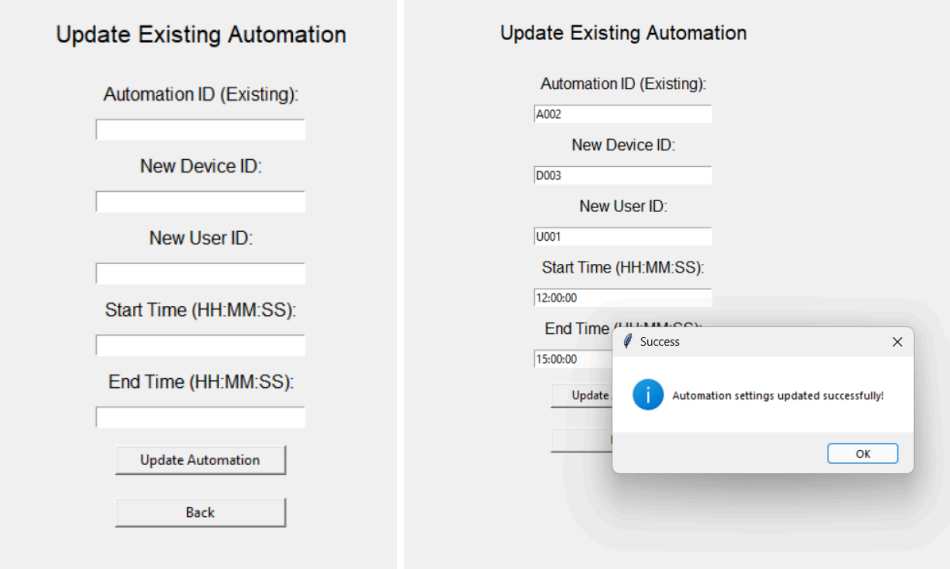


Here, the list of devices and all the logs with respect to all devices are read and displayed in a well-formatted manner.

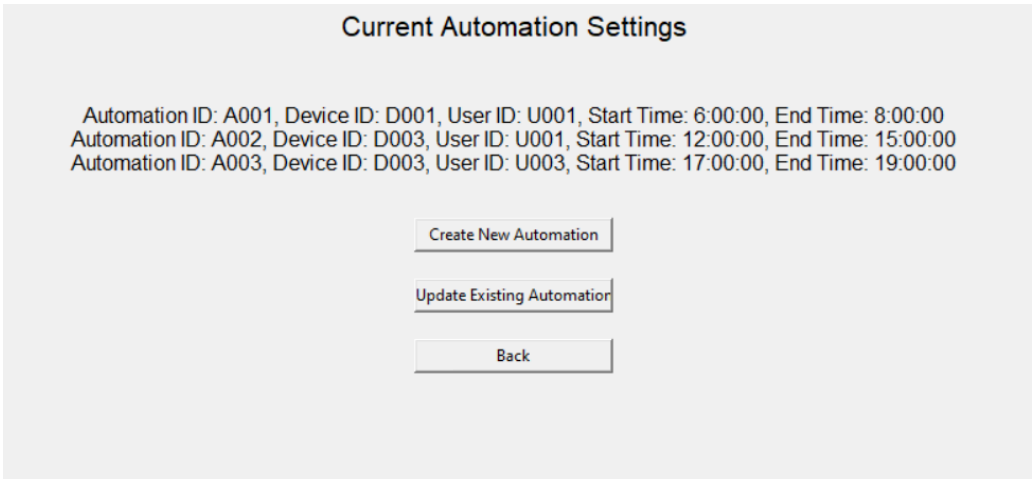
3. **UPDATE:**

Before update:



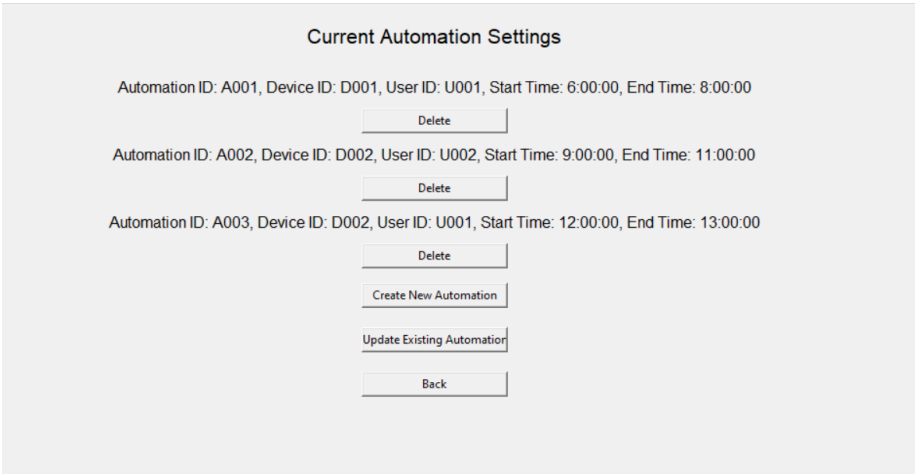


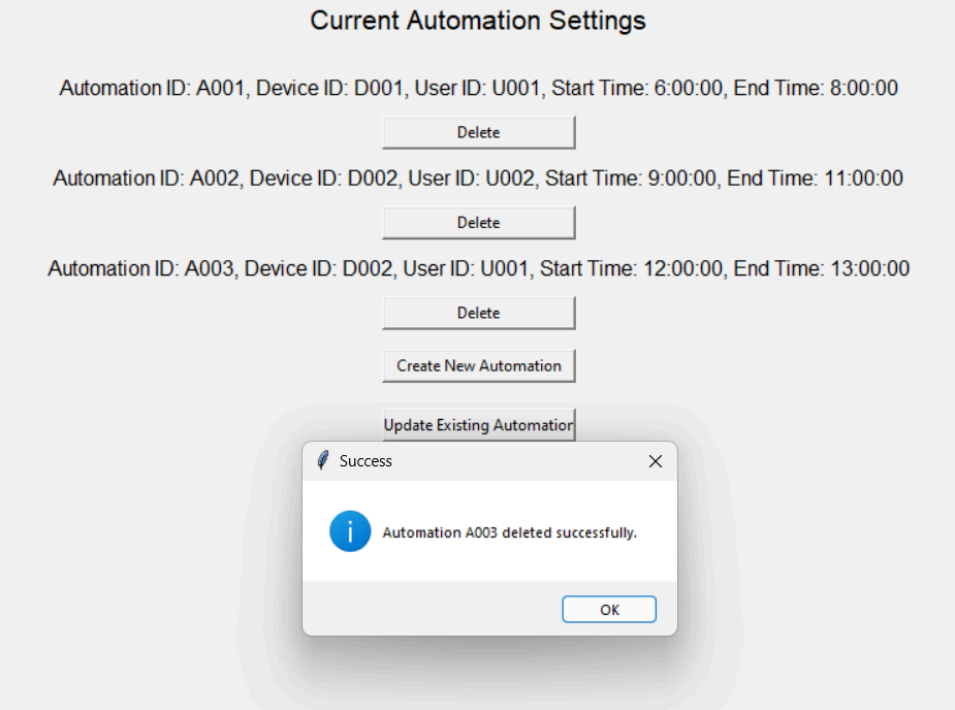
Updated Automation:



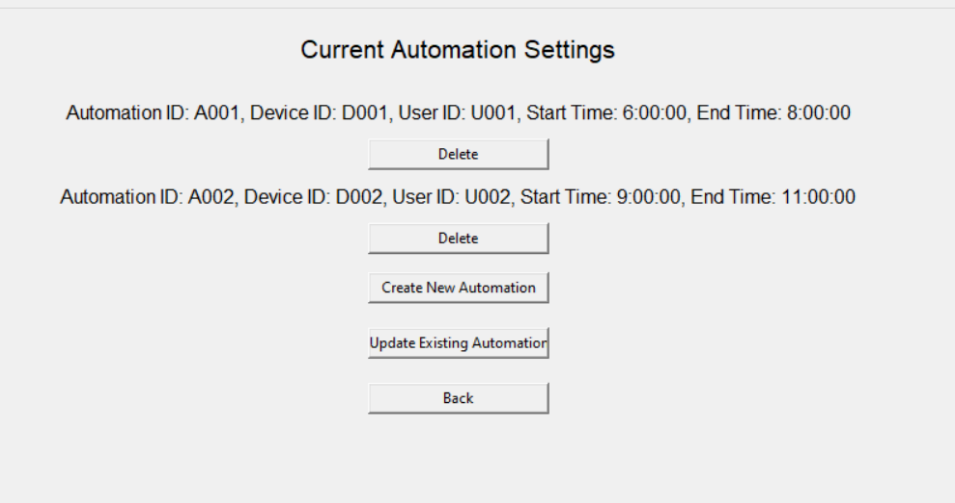
4. **DELETE:**

Before delete:





After delete:



VIII. QUERIES (JOIN QUERY, AGGREGATE FUNCTION QUERIES AND NESTED QUERY)

NESTED QUERY:

Automation Insertion with Validity Check (Query to insert a new automation entry after checking device and user validity)

```
cursor.execute("""
                    INSERT INTO Automation (automation_ID, device_ID, user_ID, start_time,
end_time)
                    SELECT %s, %s, %s, %s, %s
                    WHERE EXISTS (
                       SELECT 1 FROM Device WHERE device_ID = %s
                    ) AND EXISTS (
                       SELECT 1 FROM User WHERE user_ID = %s
                    )
                """, (new_automation_id, new_device_id, new_user_id, start_time, end_time,
new_device_id, new_user_id))
```

Explanation:

This query inserts a new record into the Automation table with the specified automation_ID, device_ID, user_ID, start_time, and end_time. It only performs the insertion if the given device_ID exists in the Device table and the user_ID exists in the User table. If either condition fails, the insert operation is not performed, ensuring data integrity and valid relationships between the tables.

JOIN AND AGGREGATE QUERIES:

Device Log Duration Summary (Query to retrieve total device usage duration from log entries)

```
cursor.execute("""
                    SELECT
                       l.log_id,
                       d.device_id,
                       l.date,
                       l.time,
                       SUM(l.duration) AS total_duration
                    FROM
                       Logs l
```

```
            JOIN
                Device d ON l.device_id = d.device_ID
            GROUP BY
                l.log_id, d.device_id, l.date, l.time
        """)
        logs = cursor.fetchall()
```

Explanation:

This query retrieves a summary of logs from the Logs table, **joining it** with the Device table to fetch device-related information. It selects the log_id, device_id, date, time, and the total duration (SUM(l.duration)) for each log entry. The results are **grouped by** log_id, device_id, date, and time, providing the total duration for each device's logs on a specific date and time.

IX. STORED PROCEDURE, FUNCTIONS AND TRIGGERS

TRIGGERS:

1. **Device Deactivation Duration Update Trigger (Trigger to update the device usage duration when it is deactivated)**
2. **Device Activation Log Insert Trigger (Trigger to insert a new log when a device is activated)**

```
CREATE TRIGGER after_device_deactivate
AFTER UPDATE ON Device
FOR EACH ROW
BEGIN
   DECLARE total_seconds INT DEFAULT 0;
   DECLARE total_minutes INT DEFAULT 0;
   DECLARE last_log_time DATETIME;

   IF NEW.status IN ('inactive', 'off') AND OLD.status IN ('active', 'on') THEN
      SELECT CONCAT(date, ' ', time) INTO last_log_time
      FROM Logs
      WHERE device_ID = OLD.device_ID
      ORDER BY log_ID DESC LIMIT 1;

      SET total_seconds = TIMESTAMPDIFF(SECOND, last_log_time, NOW());
      SET total_minutes = FLOOR(total_seconds / 60);

      UPDATE Logs
      SET duration = total_minutes
      WHERE device_ID = OLD.device_ID
      ORDER BY log_ID DESC LIMIT 1;
   END IF;
END //

CREATE TRIGGER before_device_activate
BEFORE UPDATE ON Device
FOR EACH ROW
BEGIN
   IF NEW.status IN ('active', 'on') AND OLD.status NOT IN ('active', 'on') THEN
      -- Insert a log entry with current timestamp for the duration
      INSERT INTO Logs (device_ID, date, time, duration)
      VALUES (NEW.device_ID, CURDATE(), CURTIME(), 0);
   END IF;
END //
```

Explanation:
after_device_deactivate:

- This trigger is activated after a device's status is updated to "inactive" or "off." It calculates the duration for which the device was active by finding the last log entry and determining the time difference between the last log and the current time. It then updates the duration field in the most recent log entry.

before_device_activate:

- This trigger is activated before a device's status is updated to "active" or "on." It inserts a new log entry with the current timestamp and sets the duration to 0, marking the beginning of the device's active status.

FUNCTION:
User Authentication Function (Function to authenticate users based on user ID and password)

```
CREATE FUNCTION authenticate_user(user_id VARCHAR(50), user_password
VARCHAR(50))
RETURNS VARCHAR(50)
DETERMINISTIC
BEGIN
   DECLARE user_role VARCHAR(50);

   SELECT role INTO user_role
   FROM User
   WHERE user_ID = user_id AND password = user_password;

   RETURN user_role;  -- Returns the user role or NULL if not found
END //

DELIMITER ;
```

Explanation:
authenticate_user:

- This function accepts a user_id and user_password as inputs. It checks the User table to find a matching record where both the user ID and password match the provided values. If a match is found, it retrieves and returns the user's role. If no match is found, it returns NULL, indicating authentication failure.

X. CODE SNIPPETS FOR INVOKING THE PROCEDURES/FUNCTIONS/TRIGGER:

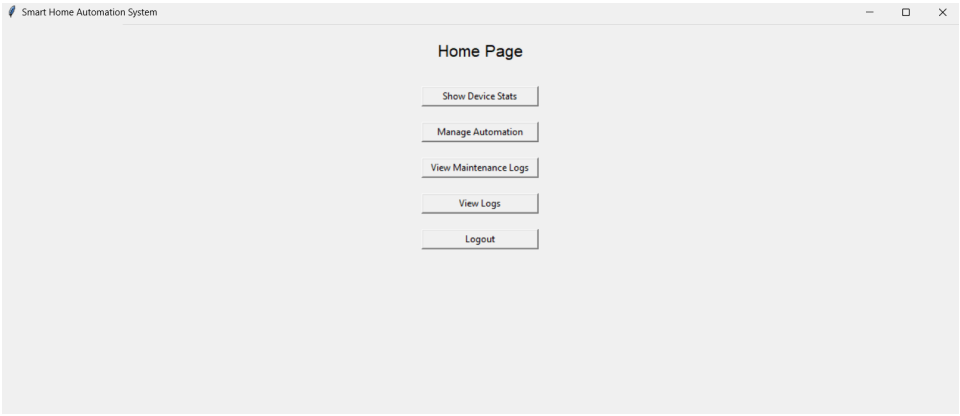
```
def authenticate_user(self):
    """Authenticate user based on ID and password."""
    user_id = self.user_id.get()
    password = self.user_password.get()

    connection = self.create_connection()
    if connection:
        with connection.cursor() as cursor:
            # Call the SQL function to authenticate user
            cursor.execute("SELECT authenticate_user(%s, %s)", (user_id, password))
            result = cursor.fetchone()

            if result and result[0]:  # Check if a role is returned
                self.user_role = result[0]  # Store the user role for later use
                self.show_common_options()
            else:
                messagebox.showerror("Login Error", "Invalid User ID or Password.")
```

This part of the function **invokes the authenticate_user SQL function** to authenticate the user based on their provided **user ID** and **password**. It executes the SQL function with the user's credentials using a cursor to interact with the database. If the function returns a valid user role, it stores the role and displays common options for the authenticated user. If the credentials are invalid, an error message is shown.

XI. FRONT END DEVELOPMENT (FUNCTIONALITIES/FEATURES OF THE APPLICATION)

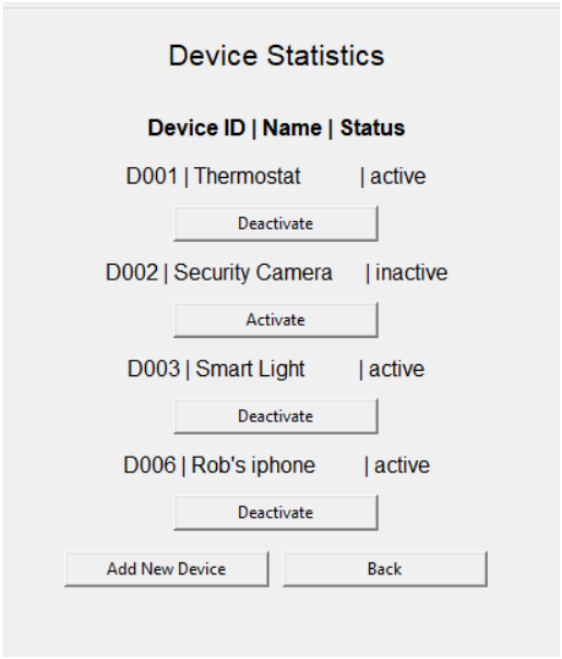
The frontend of the Smart Home Automation Management System is built using Tkinter, Python's standard GUI library, to provide users with a simple and interactive interface. Through Tkinter, various widgets like buttons, dropdowns, tables, and input fields are designed to allow users to interact seamlessly with the system's features.

Home page:

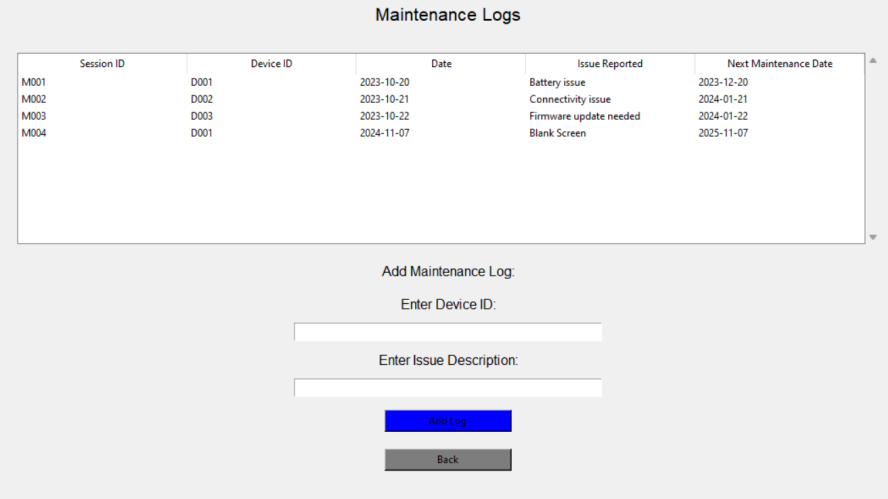


This interface enables users to:

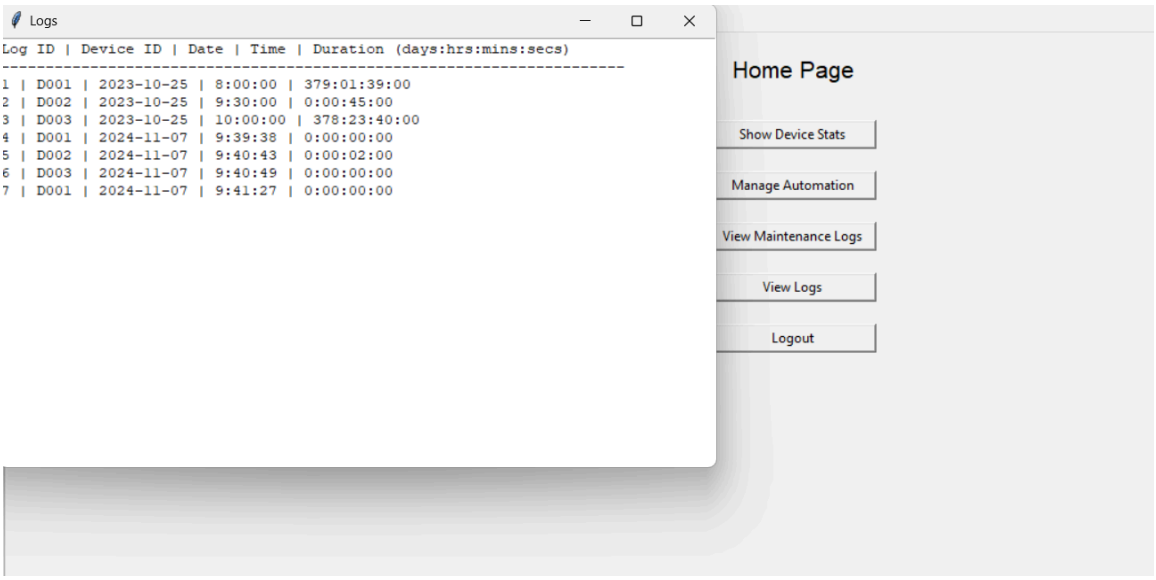
Control Devices: Users can switch devices on or off, set automation schedules, and customize settings for different devices. The interface allows intuitive access to each device's controls in a user-friendly layout.



2. **View and Add Maintenance Logs:** A dedicated table view displays maintenance logs for each device, including details like session ID, device ID, reported issues, and upcoming maintenance dates. Users can also add new logs directly through the frontend by entering required details.



3. **Monitor Device status logs:** Users can access real-time statistics and statuses of devices, including active/inactive states, usage patterns, and performance logs. The frontend makes it easy for users to keep track of device health and operational efficiency.



Overall, Tkinter helps create a minimalistic yet functional interface that aligns with the system's core goal: providing a smooth and efficient experience in managing smart home devices. The layout and design emphasize ease of navigation, allowing users to interact with various functionalities without needing technical expertise.

XII. GITHUB REPO LINK AND QR

https://github.com/Janvi-M/Smart-Home-Automation-System-Using-MySQL-and-Python

XIII. REFERENCES / BIBLIOGRAPHY

- Schneider, M., & Williams, K. (2023).
  Building User-Friendly GUIs with Python and Tkinter. Packt Publishing.
- Tkinter Documentation
  Python Software Foundation. "Tkinter – Python Interface to Tcl/Tk."
  URL: https://docs.python.org/3/library/tkinter.html
- IEEE Guide for Smart Home Automation Systems
  IEEE Standards Association. "IEEE P2413 Standard for Smart Home Automation Systems."
  URL: https://ieeexplore.ieee.org