Contents **Page number**

1. Requirement Analysis & Problem Definition

1.1 Objective

To develop a centralized database for smart home appliances that stores sensor data, manages device status, and facilitates rule-based automation using a structured and secure backend system.

1.2 Problem Statement

As smart devices become integral to homes, the unstructured nature of data collection and the absence of efficient control systems result in fragmented automation. Manual handling and lack of centralized control affect efficiency and user experience.

1.3 Proposed Solution

Designing a MySQL-based relational database that supports CRUD operations on smart devices, tracks environmental data, and enables rule-based automation through well-defined schema, indexing, and optimized queries.

1.4 Stakeholders

- Homeowners
- Database Administrators
- Developers
- Security Analysts

1.5 Functional Requirements

- Register and manage devices and rooms
- Log temperature and humidity data
- Execute user-defined automation rules
- Enable role-based user access and control

1.6 Non-Functional Requirements

- High availability and scalability
- Secure login and encryption
- Fast and responsive query performance
- Modular and maintainable schema design

2. System Architecture & Functional Overview

2.1 Architecture Components

- IoT Devices: Sensors and actuators like lights, fans, alarms
- Backend API: REST API (Python Flask) to handle requests
- Database: MySQL for structured data storage
- Frontend (Optional): UI for visualization and control

2.2 Workflow

1. Sensors collect data (e.g., temperature, humidity)
2. Devices send data to the backend via API
3. Backend inserts or updates database records
4. User can monitor/control via UI or app
5. Rules trigger automated actions

2.3 Functional Modules

- User Management
- Device Registration and Monitoring
- Environmental Logging
- Automation Rules

2.4 Logical Flow

User/App → Flask API → MySQL Database ← IoT Devices

3. ER Diagram & Schema Design

3.1 Entities

Users
user_id, username, password, email, created_at

Rooms
room_id, room_name, floor_number

Devices
device_id, device_name, device_type, status, room_id, last_updated

Environment_Log
log_id, device_id, temperature, humidity, timestamp

Rules
rule_id, device_id, trigger_condition, action, created_at

3.2 Relationships

- One user can manage multiple rooms
- One room contains multiple devices
- Devices log multiple environment records
- Each device can have one or more automation rules

3.3 ER Diagram

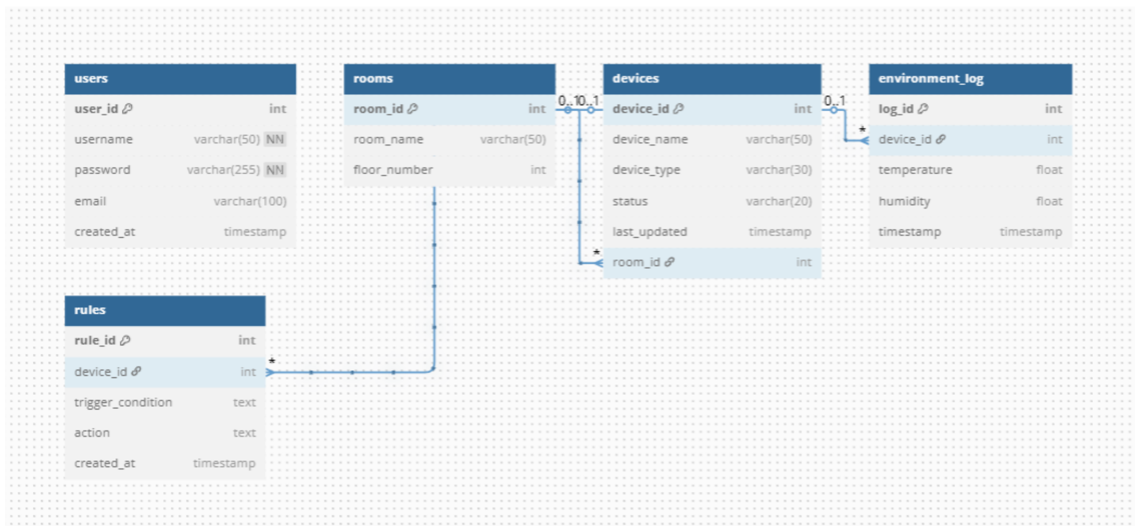

Figure 1: ER Diagram

3.4 Schema Design

```
CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  email VARCHAR(100),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE rooms (
  room_id INT AUTO_INCREMENT PRIMARY KEY,
  room_name VARCHAR(50),
  floor_number INT
);

CREATE TABLE devices (
  device_id INT AUTO_INCREMENT PRIMARY KEY,
  device_name VARCHAR(50),
  device_type VARCHAR(30),
  status VARCHAR(20),
  room_id INT,
  last_updated TIMESTAMP,
  FOREIGN KEY (room_id) REFERENCES rooms(room_id)
);

CREATE TABLE environment_log (
  log_id INT AUTO_INCREMENT PRIMARY KEY,
  device_id INT,
  temperature FLOAT,
  humidity FLOAT,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (device_id) REFERENCES devices(device_id)
);

CREATE TABLE rules (
  rule_id INT AUTO_INCREMENT PRIMARY KEY,
  device_id INT,
  trigger_condition TEXT,
  action TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (device_id) REFERENCES devices(device_id)
);
```

4. Normalization & SQL Query Formulation

4.1 Normalization

1NF: Atomic columns and unique rows
2NF: No partial dependencies
3NF: No transitive dependencies

4.2 Example Queries

```
-- Add a new room
```

```
INSERT INTO rooms (room_name, floor_number) VALUES ('Living Room', 1);

-- Add a device
INSERT INTO devices (device_name, device_type, status, room_id, last_updated)
VALUES ('Smart AC', 'Cooling', 'OFF', 1, NOW());

-- Log temperature and humidity
INSERT INTO environment_log (device_id, temperature, humidity)
VALUES (1, 27.3, 45.2);

-- Get latest readings for all devices
SELECT * FROM environment_log ORDER BY timestamp DESC;

-- Get devices in a room
SELECT d.device_name FROM devices d
JOIN rooms r ON d.room_id = r.room_id
WHERE r.room_name = 'Bedroom';

-- Add a rule
INSERT INTO rules (device_id, trigger_condition, action)
VALUES (1, 'temperature > 30', 'Turn ON Smart AC');
```

5. Database Implementation & Indexing
5.1 Implementation
Step 1: Create database using SQL scripts
Step 2: Populate data manually or via API

```
CREATE DATABASE smart_home;
USE smart_home;
```

5.2 Indexing

- Primary and foreign keys for relationships
- Index on timestamp for fast log queries
- Composite index on device_id, date for environment_log

5.3 Backup Strategy

- Scheduled backups using mysqldump
- Redundant local/cloud storage

6. Testing & Performance Evaluation

6.1 Testing Strategy

- **Unit Testing:** Device registration, sensor data logging
- **Integration Testing:** Logs correctly link to devices
- **System Testing:** End-to-end workflow validation
- **UAT:** Verified user experience with mock frontend

6.2 Performance Metrics

- **Insert Rate:** ~60 logs/sec with minimal latency
- **Read Latency:** < 150 ms for indexed queries
- **Stress Test:** Simulated 1000+ entries with stable performance

6.3 Load Simulation

Used Apache JMeter to simulate 50 devices sending data every second for 5 minutes

7. Technologies Used

Tool	**Purpose**
MySql	Relational databse
Python flask	Rest API for DB interaction
Postman	API testing
Draw.io	ER diagram creation
GitHub	Version control and collaboration
VS Code	Development ide
MySql workbench	Schema design and query testing

8. Security, Usability & Scalability Considerations

8.1 Security

- Passwords hashed using bcrypt
- SQL Injection protection via parameterized queries

- Role-based access control (admin vs user)

8.2 Usability

- Simple schema supports extensibility
- REST API with clear documentation
- Potential for UI integration with minimal effort

8.3 Scalability

- Modular schema supports adding new device types
- Can be migrated to cloud DB like AWS RDS or Google Cloud SQL
- Future support for distributed logs using Kafka/MongoDB

9. Conclusion & Future Scope

9.1 Conclusion

The IoT-based Smart Home Appliance Database successfully captures and organizes data from smart devices. It provides a solid backend framework for real-time monitoring, rule-based automation, and efficient device management.

9.2 Future Enhancements

- Integration with MQTT for real-time streaming
- Mobile app for instant control
- AI/ML model for energy optimization
- Third-party API integration (e.g., Alexa, Google Home)
- Analytics dashboard for usage patterns

10. References & Appendix

References

- GitHub Repo: https://github.com/tarek-debug/Smart_Home_Appliance_Database
- MySQL Documentation: https://dev.mysql.com/doc/
- Flask Documentation: https://flask.palletsprojects.com/

- DBMS Course Textbook and Lectures

Group members (52-5):

Aditya bhardwaj(2023A6R052)
Hardik Uttam (2023A6R053)
Rohit raina (2023A6R054)