Zagazig University Faculty of Engineering
Computer & Systems Engineering Department

SMART HOME HAVEN

Home Automation with Verification, Emergency & Network

By

Mostafa Ahmed Abdelrahman Abdelhamid (T.L)

Tarek Fathallah Mohamed Ali

Marwan Hosni Ibraheem Abdallah

Mohamed Ibraheem Fathy Khaleel

Ahmed Mahmoud Maged AwadAllah

Khaled Abdelmgeed Abdelmegeed Mohamed

Moamen Bellah Mamdouh Ismail Ahmed

Supervisor:

Dr / Howaida Abd-Ellateef

July,2025

Abstract

The increasing adoption of Internet of Things (IoT) technologies has transformed modern living spaces, driving the development of smart home systems that enhance comfort, safety, and efficiency. This graduation project presents a comprehensive smart home management system that integrates multiple intelligent features into a unified platform, offering users advanced control and real-time monitoring of their home environment.The system is built around an embedded microcontrollers :

1.ESP32

2. Arduino mega

3.Rasberrypi

connected to a cloud-based backend, enabling seamless communication between sensors, actuators, and user interfaces. Key functionalities include:

- **Security System**: A multi-layered approach that combines Fingerprint ,Password, door access control, face detection, and live monitoring.
- **Facial Recognition**: AI-powered face authentication for secure and to allow people in the home to know who is knocking the door.
- **Fire and Gas Detection**: Real-time AI-based fire detection using image processing, alongside gas sensors to identify hazardous leaks.
- **Light and comfort system** : smart lighting Automated control of indoor and outdoor lighting based on motion detection and ambient conditions .**And** Ventilation Management: Adaptive fan control to decrease temperature in hot weather and optimize air circulation and indoor air quality.
- **Mobile Application**: A user-friendly Flutter-based app that allows remote monitoring, notifications, and full system control.
- **Web Dashboard** : A responsive web interface for data visualization about the compound where the home is in .

The system bridges advanced IoT hardware with intuitive software interfaces to support both real-time automation and user-driven interaction. This book presents the architecture, component integration, and interface designs that shape the system, providing practical insights into its development process and its potential application in real-world smart home environments

Acknowledgments

We would like to extend our deepest appreciation to all those who have supported and contributed to the success of this graduation project. Their encouragement, guidance, and belief in our abilities have made this experience both rewarding and memorable.

First and foremost, we express our sincere gratitude to Dr. Huwaida Abd El Lateef for her exceptional supervision and unwavering support throughout every phase of the project. Her insightful guidance, academic expertise, and continuous encouragement played a vital role in helping us overcome challenges and refine our ideas into a coherent and functional system. Her dedication and mentorship have left a lasting impact on both our project and our personal development.

We are also profoundly thankful to our families, whose patience, motivation, and constant support have been our backbone during this journey. Their faith in us was a driving force that kept us focused and determined through every stage of the project.

Lastly, we would like to acknowledge everyone who offered their help, whether through technical input, thoughtful suggestions, or moral support. Each contribution, big or small, has added value to this project and helped bring it to completion.

Table of Contents

Chapter 1: Introduction to Smart Homes

1.1 Introduction to Smart Homes

The concept of smart homes has revolutionized residential living by integrating cutting-edge technologies to automate and optimize home functionalities. A smart home leverages interconnected devices and systems to provide residents with enhanced control, security, and efficiency. The evolution of smart homes began in the late 20th century with basic home automation systems, such as programmable thermostats and remote-controlled lighting. Over time, advancements in the Internet of Things (IoT), embedded systems, artificial intelligence (AI), and wireless communication have transformed these rudimentary systems into sophisticated ecosystems capable of predictive and adaptive management.

Smart homes now enable seamless control over various home functions, including lighting, security, heating, ventilation, air conditioning (HVAC), and entertainment systems. These systems are designed to improve the quality of life by offering convenience, enhancing security, and promoting energy efficiency. For example, homeowners can remotely adjust lighting or monitor security cameras using a smartphone app, ensuring both accessibility and peace of mind. Additionally, smart homes contribute to sustainability by optimizing resource usage, such as reducing energy consumption through intelligent lighting and appliance control.

Key Benefits of Smart Homes

Smart home systems offer a range of benefits that address modern homeowner needs:

- Convenience: Residents can control devices remotely via smartphones, tablets, or centralized systems, eliminating the need for manual operation. For instance, turning off lights or adjusting the thermostat while away from home is now effortless.
- Security: Advanced security features, such as biometric access control (e.g., fingerprint or facial recognition) and real-time surveillance, provide robust protection against unauthorized access and potential threats.
- Energy Efficiency: Smart systems optimize energy usage by automating lighting, HVAC, and appliances based on occupancy or predefined schedules, reducing waste and lowering utility costs.
- Scalability and Adaptability: Modern smart home systems are designed to integrate with new technologies, allowing homeowners to expand functionalities as needed, such as adding smart appliances or renewable energy sources.

The smart home industry is experiencing rapid growth, driven by technological advancements and increasing consumer demand. Key trends include:

- AI-Driven Automation: AI algorithms enable predictive behaviors, such as adjusting lighting based on user habits or detecting anomalies in security patterns.
- Integration with Renewable Energy: Smart homes are increasingly paired with solar panels and energy storage systems to promote sustainability.
- User-Friendly Interfaces: Cross-platform compatibility and intuitive mobile apps make smart home systems accessible to users of all technical backgrounds.
- Interoperability: The adoption of universal protocols (e.g., Zigbee, Z-Wave) ensures seamless communication between devices from different manufacturers.

Looking ahead, the future of smart homes lies in greater personalization and autonomy. AI models are expected to evolve, enabling systems to learn from user preferences and anticipate needs. For example, a smart home could automatically adjust lighting and temperature based on the time of day or the presence of specific residents. Additionally, advancements in 5G connectivity and edge computing will enhance real-time processing, making smart homes more responsive and efficient.

1.2 Problem Statement

Traditional homes face several challenges that limit their efficiency, security, and convenience:

- Security Vulnerabilities: Conventional locks and basic alarm systems are susceptible to breaches, such as lock-picking or tampering, leaving homes vulnerable to intruders.
- Energy Waste: Manual control of lighting, heating, and appliances often results in unnecessary energy consumption, especially when devices are left on unintentionally.
- Lack of Centralized Control: Disparate systems for lighting, security, and HVAC make it difficult to manage home functions cohesively, leading to inefficiencies and user frustration.
- Limited Accessibility: Traditional homes lack remote monitoring and control capabilities, restricting homeowners' ability to manage their homes while away.

These challenges underscore the need for an integrated smart home solution that combines advanced security, energy-efficient automation, and user-friendly interfaces to meet the demands of modern living. Such a system must address security

vulnerabilities, reduce energy waste, provide centralized control, and enable remote access to enhance the overall living experience.

1.3 Importance of the Project

This project aims to develop a comprehensive smart home system that addresses the identified challenges while delivering a scalable and user-centric solution. By integrating advanced technologies, the system enhances security, convenience, and energy efficiency, making it suitable for individual homes and larger residential compounds. The importance of this project lies in its ability to transform residential living by:

- Enhancing Security: Incorporating biometric authentication (fingerprint and facial recognition) and secure access control mechanisms to protect against unauthorized access.
- Improving Convenience: Enabling seamless control of home devices, such as lighting and security systems, through a mobile app and embedded hardware.
- Promoting Energy Management: Implementing automated lighting control and scheduling to minimize energy consumption.
- Ensuring Accessibility: Providing remote monitoring and control capabilities to allow homeowners to manage their homes from anywhere.
- Driving Innovation: Combining diverse technologies, including Arduino, ESP32, Raspberry Pi, Flutter, and web platforms, to create a cohesive and scalable solution.

This project is particularly significant in the context of residential compounds, where centralized management and scalability are critical. By addressing the needs of homeowners, administrators, and maintenance staff, the system ensures a holistic approach to smart home implementation.

1.4 Proposed Project Idea and Solution

1.4.1 Project Vision and Objectives

Vision: To design and implement an integrated smart home system that enhances security, convenience, and energy efficiency for residents within a residential compound, while providing centralized management for administrators.

Objectives:

1. Develop a secure access control system using embedded hardware, including keypad entry, fingerprint authentication, and solenoid locks.

2. Implement intelligent lighting control for energy efficiency, with remote access via a mobile app.
3. Integrate advanced safety features, such as fire detection and facial recognition, for real-time monitoring.
4. Create a user-friendly mobile app for homeowners to interact with and manage the smart home system.
5. Build a web platform for centralized monitoring and administration of multiple homes within a compound.

1.4.2 System Architecture Overview

The smart home system is designed as a modular and interconnected ecosystem, with each component performing specific functions while communicating seamlessly. The architecture includes:

- Embedded Access Control (Arduino): Manages secure entry/exit using a keypad, fingerprint sensor, solenoid lock, and RTC module for logging access events.
- Intelligent Lighting Control (ESP32): Controls LEDs using motion sensors (IR and PIR) and connects to the mobile app via Wi-Fi for remote management.
- Safety and Security Processing (Raspberry Pi): Handles fire detection and facial recognition, processing sensor data and camera feeds to ensure real-time safety.
- Mobile Application (Flutter): Provides a cross-platform interface for homeowners to control devices, monitor security, and receive alerts.
- Compound Website: Aggregates data from multiple homes for centralized management, offering administrators a dashboard for oversight and user management.

High-Level Block Diagram:

- Arduino-Raspberry Pi Communication: The Arduino logs access events (e.g., fingerprint scans, PIN entries) and sends them to the Raspberry Pi via serial communication.
- ESP32-Mobile App Connectivity: The ESP32 connects to the mobile app over Wi-Fi, enabling remote control of lighting and real-time status updates.
- Raspberry Pi Processing: The Raspberry Pi processes fire sensor data and camera feeds, interfacing with the mobile app and website for alerts and monitoring.

- Website Integration: The website aggregates data from all smart homes, providing a centralized dashboard for compound-wide oversight.

This architecture ensures modularity, allowing components to function independently while contributing to a cohesive system. For example, the lighting control system can operate standalone but integrates with the mobile app for enhanced user control.

1.4.3 Detailed Solution Components

1.4.3.1 Embedded Access Control System (Arduino)

Functionality: The access control system secures entry and exit points, ensuring only authorized individuals can access the home.

- Components:
    - Keypad: Allows users to enter a PIN for authentication, providing a simple yet secure access method.
    - Fingerprint Sensor: Uses biometric authentication to verify user identity, offering a higher level of security than traditional keys.
    - Solenoid Lock: Physically locks or unlocks doors based on successful authentication.
    - RTC Module: Records timestamps for access events, enabling detailed logging.
    - LCD Display: Provides user feedback, such as instructions or confirmation messages (e.g., “Access Granted”).
- Integration: The Arduino communicates access logs to the Raspberry Pi via serial communication, enabling real-time monitoring and storage of entry/exit data.
- Example Scenario: A resident approaches the door, scans their fingerprint, and enters a PIN. Upon successful authentication, the solenoid lock opens, and the event is logged with a timestamp for later review.

1.4.3.2 Lighting and Comfort System (ESP32)

Functionality: The lighting control system optimizes energy usage by automating LED operation based on motion detection and user input.

Components:

- ESP32 Microcontroller: Serves as the central processor for lighting control, supporting Wi-Fi connectivity for remote access.
- LEDs: Provide energy-efficient lighting for the home.
- IR Sensors: Detect occupancy to trigger lighting adjustments.
- PIR Sensors: Sense motion to activate or deactivate lights, reducing energy waste.
- DHT sensor : to read temperature and humidity
- Fan or (air conditioner) : to control temperature depending on DHT readings

- Features:
  - Remote Control: Users can turn LEDs on/off or adjust brightness via the mobile app.
  - Automation: Lights activate automatically when motion is detected, ideal for areas like hallways or bathrooms.
  - Scheduling Potential: Future iterations could include time-based schedules to turn lights on/off at specific times.
- Example Scenario: A resident enters a room, triggering the PIR sensor to turn on the LEDs. If no motion is detected for a set period, the lights turn off automatically to save energy.

1.4.3.3 Fire Detection and Face Recognition (Raspberry Pi)

Functionality: Enhances safety and security through real-time monitoring and processing.

- Components:
  - Raspberry Pi: Acts as the central processing unit for fire detection and facial recognition.
  - Fire Sensor: Detects smoke or heat, indicating potential fire hazards.
  - Camera Module: Captures images for facial recognition and security monitoring.

- Fire Detection:
    - Detects fire hazards and triggers immediate alerts to the mobile app.
    - Logs incidents for review by homeowners or administrators.
- Facial Recognition:
    - Identifies authorized individuals using algorithms like Haar cascades or deep learning models.
    - Logs entry/exit events, enhancing security by tracking access.
- Example Scenario: The fire sensor detects smoke, triggering an alert on the mobile app to warn residents. Simultaneously, the camera module captures an image of a visitor, and the facial recognition system verifies their identity before granting access.

1.4.3.4 Mobile Application (Flutter)

Functionality: Provides a user-friendly, cross-platform interface for homeowners to interact with the smart home system.

- Features:
    - Secure User Registration and Login: Uses encrypted protocols to ensure data security.
    - Device Control: Allows users to manage LEDs, view system status, and control other devices.
    - Real-time Monitoring: Displays access logs, fire alerts, and security events.
    - Push Notifications: Sends alerts for critical events, such as fire detection or unauthorized access attempts.
    - Cross-Platform Compatibility: Built with Flutter for seamless operation on iOS and Android.
- Example Scenario: A homeowner receives a push notification about a fire alert while away from home. They open the app to view the status, turn on all lights for visibility, and contact emergency services if needed.

1.4.3.5 Website for the Compound

Functionality: Serves as a centralized platform for compound administrators to monitor and manage multiple smart homes.

- Features:
    - Dashboard Interface: Displays real-time data on home statuses, including security events and access logs.
    - Admin Controls: Allows adding, removing, or modifying resident access and system settings.
    - Scalable Monitoring: Supports oversight of multiple homes within the compound.
- Example Scenario: An administrator logs into the website to review access logs for a specific home, noticing an unauthorized entry attempt. They update user permissions to enhance security.

1.5 Challenges and Implementation

1.5.1 Challenges Encountered

The development and deployment of the smart home system presented several technical, practical, and user-related challenges. Each challenge required careful consideration to ensure the system's reliability, security, and usability. Below is a detailed analysis of these challenges, including their implications and real-world examples.

- Hardware Integration:
    - Description: Ensuring seamless communication between diverse hardware components—Arduino (access control), ESP32 (lighting control), and Raspberry Pi (fire detection and facial recognition)—was complex due to differing communication protocols and processing capabilities.
    - Implications: Inconsistent data exchange could lead to system failures, such as delayed access logs or unresponsive lighting controls. For example, a delay in Arduino-Raspberry Pi communication could prevent real-time security monitoring.
    - Example: During initial testing, the Arduino occasionally failed to transmit access logs to the Raspberry Pi due to serial communication bottlenecks, causing incomplete security records.
- Software-Hardware Interface:
    - Description: Developing reliable protocols for the mobile app and website to interact with hardware components required robust APIs and error handling. Real-time responsiveness was critical for user satisfaction.

    - Implications: Poor interfaces could result in laggy controls or missed alerts, reducing user trust. For instance, a slow API response could delay a fire alert notification.
    - Example: Early app versions experienced intermittent connectivity with the ESP32, causing delays in lighting control commands.
- Data Security and Privacy:
    - Description: Protecting sensitive biometric data (e.g., fingerprints, facial images) and ensuring secure data transmission over Wi-Fi and serial channels were paramount to prevent breaches.
    - Implications: A security breach could compromise user privacy or allow unauthorized access. For example, unencrypted facial recognition data could be intercepted during transmission.
    - Example: Initial tests revealed that unencrypted Wi-Fi transmissions from the ESP32 were vulnerable to interception, necessitating stronger encryption protocols.
- Real-time Performance:
    - Description: Optimizing facial recognition and fire detection algorithms for responsiveness on resource-constrained devices like the Raspberry Pi was challenging, especially under varying conditions (e.g., low light, high smoke density).
    - Implications: Slow processing could delay critical alerts, such as fire notifications, or reduce facial recognition accuracy, compromising security.
    - Example: Early facial recognition tests showed a 1-second delay in low-light conditions, unacceptable for real-time security applications.
- Scalability:
    - Description: Designing a system that scales to support multiple homes in a residential compound required efficient data management and network stability.
    - Implications: Scalability issues could lead to performance degradation in large compounds, such as delayed dashboard updates on the website.
    - Example: Simulating 20 homes revealed database bottlenecks, slowing down the compound website’s dashboard refresh rate.
- User Experience (UX):
    - Description: Creating intuitive interfaces for diverse users, including tech-savvy residents and non-technical administrators, required balancing simplicity and functionality.
    - Implications: Poor UX could discourage adoption, especially for users unfamiliar with smart home systems.
    - Example: Early mobile app prototypes received feedback that navigation was confusing, particularly for accessing security logs.
- Power Management:
    - Description: Minimizing power consumption for always-on components (e.g., sensors, microcontrollers) was critical for sustainability and cost efficiency.
    - Implications: High power usage could increase operational costs or require frequent maintenance, such as battery replacements.

- Example: The ESP32 initially consumed excessive power in idle mode, reducing its suitability for continuous operation.

1.5.2 Implementation Details

The implementation process involved a structured workflow, combining hardware prototyping, software development, testing, and user feedback to address the challenges and deliver a robust system. Below is a comprehensive overview of the implementation, including tools, protocols, workflows, and optimization strategies.

- Tools and Technologies:
  - Arduino IDE: Used for programming the access control system, leveraging libraries like Adafruit Fingerprint, Keypad, RTClib, and LiquidCrystal for keypad, fingerprint sensor, RTC module, and LCD integration.
  - Python Libraries: Employed on the Raspberry Pi for fire detection and facial recognition, using OpenCV for image processing, NumPy for numerical computations, and Firebase Admin SDK for cloud integration.
  - Flutter SDK: Facilitated cross-platform mobile app development with Dart, using Firebase Authentication and Cloud Messaging for secure login and real-time alerts.
  - Firebase and Web Frameworks: Supported the compound website with a React frontend, Node.js backend, and Firebase for real-time data storage and authentication.
- Communication Protocols:
  - Wi-Fi: Used for ESP32 connectivity to the mobile app and website, leveraging MQTT for lightweight, real-time messaging. The MQTT broker ensured reliable command and status updates.
  - Serial Communication: Facilitated data exchange between Arduino and Raspberry Pi at 9600 baud rate, using JSON-formatted messages for access logs (e.g., {"user_id": 123, "timestamp": "2025-07-07 10:30:45", "status": "granted"}).
  - WebSocket: Enabled real-time updates on the website for compound-wide monitoring, ensuring low-latency dashboard refreshes.
- Implementation Workflow:
  - Phase 1: Requirements Analysis:
    - Conducted stakeholder interviews with homeowners, administrators, and maintenance staff to define system requirements, such as secure access, energy-efficient lighting, and centralized management.
    - Established performance metrics, including a maximum 1-second response time for access control and 200ms for fire alerts.
  - Phase 2: Hardware Prototyping:
    - Built individual prototypes for Arduino (access control), ESP32 (lighting), and Raspberry Pi (fire detection, facial recognition).
    - Example: The Arduino prototype was tested with a breadboard setup, integrating the keypad, fingerprint sensor

1.6 Project Scope

1.6.1 In-Scope Features

The project focuses on delivering the following core functionalities:

- Access Control: Keypad, fingerprint sensor, solenoid lock, RTC module, and LCD for secure entry management.
- Lighting Control: ESP32-based LED control with motion detection and remote access via the mobile app.
- Safety and Security: Fire detection and facial recognition for real-time monitoring and alerts.
- User Interfaces:
    - Mobile app for homeowner control (registration, login, LED control, alerts).
    - Compound website for centralized management (dashboard, user management).

1.6.2 Out-of-Scope Features

To maintain focus and feasibility, the following features are excluded:

- Advanced Energy Monitoring: Beyond basic LED control, such as monitoring entire home energy consumption.
- Third-Party Security Integration: Connecting with external security services or alarm systems.
- Complex Machine Learning: Advanced AI models beyond basic facial recognition, such as predictive maintenance or behavior analysis.

1.7 Project Users and High-Level Areas

1.7.1 Identification of Project Users

The system serves multiple user groups with distinct needs:

- Homeowners/Residents: Use the mobile app to control lighting, monitor security, and receive alerts for events like fire detection or unauthorized access.
- Compound Management/Administrators: Utilize the website to oversee multiple homes, manage user access, and monitor system status.

- Maintenance Staff: Access specific components (e.g., hardware modules) for troubleshooting and repairs, ensuring system reliability.

1.7.2 High-Level Project Areas

1.7.2.1 Image Processing and Biometrics

Core Role: Provides advanced security through biometric authentication, leveraging fingerprint and facial recognition technologies.

- Applications:
    - Fingerprint Authentication (Arduino): Processes fingerprint data to verify authorized users, offering high accuracy and security.
    - Facial Detection (Raspberry Pi): Uses algorithms like Haar cascades or deep learning to detect human faces in camera feeds.
    - Facial Recognition (Raspberry Pi): Matches detected faces against a database of authorized users, logging entries/exits for security tracking.
    - Image Pre-processing: Applies techniques like normalization, alignment, and noise reduction to enhance recognition accuracy.
    - Feature Extraction: Derives unique biometric features (e.g., fingerprint patterns, facial landmarks) for reliable authentication.
- Impact: Creates a dual-layer biometric security framework, combining fingerprint and facial recognition to ensure robust protection and user trust.

1.7.2.2 Mobile Application Development

Core Role: Delivers a cross-platform interface for homeowners to interact with the smart home system seamlessly.

- Applications:
    - User Authentication: Implements secure login and registration using encrypted protocols to protect user data.
    - Device Control: Provides intuitive controls for managing LEDs and monitoring system status (e.g., security alerts).
    - Real-time Alerts: Sends push notifications for critical events, such as fire detection or unauthorized access attempts.
    - Cross-Platform Compatibility: Built with Flutter to ensure consistent performance on iOS and Android devices.

- Impact: Enhances accessibility and convenience, allowing residents to manage their homes remotely with a responsive and intuitive interface.

1.7.2.3 Website Development

Core Role: Provides a centralized platform for compound administrators to monitor and manage multiple smart homes.

- Applications:
    - Dashboard Interface: Displays real-time data on home statuses, including security events, access logs, and system health.
    - User Management: Allows administrators to add, remove, or modify resident access and system settings.
    - Scalable Monitoring: Supports oversight of multiple homes, ensuring scalability for large compounds.
- Impact: Streamlines compound management by providing administrators with a comprehensive tool for oversight and control.

Chapter 2: Existing Models of Smart Homes

2.1 Overview

As we embarked on developing our smart home system, a critical step was identifying the features that would define our solution and set it apart in a competitive market. To achieve this, we conducted thorough research into existing smart home offerings, focusing on prominent companies in Egypt, the Gulf region (specifically the UAE), and the United States. Our objective was to understand industry-standard features, such as lighting control, security systems, and user interfaces, to ensure our system meets user expectations while introducing innovative elements not commonly found in commercial solutions. Given our constraints—limited financial resources and modest technical expertise as a team of enthusiastic learners—we aimed to design a system that balances practicality with ambition, leveraging affordable technologies to deliver a robust solution.

Equally important was the process of role assignment within our team. We carefully aligned tasks with each member's passions and career aspirations to foster enthusiasm and skill development. For instance, team members interested in embedded systems worked on Arduino-based access control, while those passionate about mobile development tackled the Flutter app. This approach ensured that each member was motivated to learn and contribute to features that could serve as a foundation for their future careers. The following sections present the results of our market research, detailing key companies, their smart home features, and how they informed our design choices while inspiring us to innovate within our resource constraints.

2.2 Existing Smart Home Models

2.2.1 Regional Market Context

The smart home market varies significantly across regions due to differences in infrastructure, consumer preferences, and economic factors. In Egypt, the market is growing, driven by increasing demand for affordable and energy-efficient solutions, with companies like Cordless and Teknetic catering to local needs (e.g., 220–240V compatibility). The UAE focuses on luxury and scalability, with companies like Flipsmart and Alayoubi Technologies offering high-end automation for affluent customers. In the USA, a mature market, companies like Google Nest and Samsung SmartThings dominate with broad ecosystems and advanced integrations like voice control. These regional differences shaped our understanding of feature priorities, such as affordability in Egypt, scalability in the UAE, and interoperability in the USA, guiding our system's design to address diverse needs.

2.2.2 Companies in Egypt

2.2.2.1 Cordless (Egypt)

Overview: Cordless is a locally founded Egyptian company designed to make smart home technology accessible and affordable for Egyptian households. It focuses on plug-and-play solutions that integrate seamlessly with Egypt's electrical infrastructure, emphasizing energy efficiency, security, and user-friendly control.

Features:

- Smart Lighting Control: Allows users to convert traditional lights into smart ones, controllable via the Cordless mobile app. Features include remote on/off, dimming, and scheduling, compatible with 220–240V systems. Users can create scenes like "Night Mode" for low brightness or "Movie Mode" for ambient lighting. For example, a homeowner schedules bedroom lights to turn off at 11 PM, reducing energy consumption by 15%.
- Home Security: Includes smart plugs and cameras with motion detection and real-time video feeds. Devices connect directly via Wi-Fi, requiring no hub. A user receives a motion alert at their front door while at work and checks the live feed to confirm safety.
- Energy Efficiency: Real-time energy tracking monitors consumption of lights and appliances, displaying data on the app to optimize usage. For instance, a family identifies high AC usage and sets it to run only when rooms are occupied, saving 20% on electricity bills.
- Mobile App Capabilities: An Arabic-friendly app for iOS and Android supports remote control and automation routines without hidden fees. A user activates "Work Mode" to turn off all devices when leaving home, ensuring energy savings.
- Scalability and Flexibility: Modular systems allow expansion from one room to the entire home without rewiring. A user starts with smart lighting in the living room and later adds smart plugs for kitchen appliances.
- Local Support: Offers 24/7 customer service, fast delivery, and secure payments tailored for Egypt. A customer resolves a Wi-Fi connectivity issue within hours via support.

Case Study: A middle-class family in Cairo installed Cordless smart lighting and cameras in their apartment. They used the app to schedule lights to turn off at night and received motion alerts when away, enhancing security. The energy tracking feature helped them reduce their monthly bill by 18%, demonstrating affordability and practicality for Egyptian households.

Technical Insights: Cordless uses Wi-Fi-based IoT protocols (e.g., MQTT) for device communication, with a cloud-based backend for app integration. The system supports over-the-air (OTA) updates to ensure compatibility and security, making it suitable for users with limited technical expertise.

Comparison to Our System: Cordless's lighting and app control align with our ESP32-based LED control and Flutter app, but it lacks biometric authentication (fingerprint, facial recognition) and fire detection, which our system includes for enhanced security and safety.

2.2.2.2 Teknetic (Egypt)

Overview: Teknetic is a system integrator in Egypt offering turnkey smart home solutions using Tuya and KNX systems. It serves residential and commercial clients, including hotels, with a focus on customization, reliability, and extended support.

Features:

- Smart Lighting Control: Supports wireless Tuya devices and KNX-based systems for remote control, scheduling, and preset scenarios (e.g., "Dinner Mode" for dimmed dining lights). Users adjust brightness and color via the Tuya app or KNX panels. A homeowner sets a scene to dim lights during meals.
- Security Systems: Integrates cameras, motion sensors, and smart locks with HD video, night vision, and keyless entry via PIN or app. A hotel uses cameras to monitor entrances with real-time alerts to managers.
- Climate Control: Smart thermostats and HVAC systems optimize energy use, integrated with KNX for precise control or Tuya for Wi-Fi operation. A resident programs the AC to run only when a room is occupied.
- Entertainment Systems: Multi-room audio/video systems allow streaming across rooms, controlled via apps or KNX panels. A user plays music in multiple rooms with one command.
- Extended Warranty and Support: Up to three years warranty for KNX systems, with installation and remote diagnostics. A client upgrades their system with new sensors, handled by Teknetic.
- Modular Design: Allows customization, with Tuya for small setups and KNX for large projects like hotels. A homeowner starts with lighting and later adds security.

Case Study: A luxury hotel in Sharm El-Sheikh implemented Teknetic's KNX system to automate lighting, HVAC, and security across 50 rooms. The system reduced energy costs by 25% through occupancy-based controls and improved guest experience with multi-room audio. Teknetic's support team conducted regular maintenance, ensuring reliability.

Technical Insights: Tuya devices use Wi-Fi and Zigbee protocols, while KNX employs a wired bus system for robust performance in large setups. The Tuya app uses a RESTful API for device control, and KNX systems integrate with building management systems (BMS) for centralized oversight.

Comparison to Our System: Teknetic's lighting and security features resemble our ESP32 and Arduino components, but its KNX systems are suited for larger projects. Our biometric authentication and fire detection provide unique safety features not highlighted in Teknetic's offerings.

2.2.2.3 I&R Egypt (Egypt)

Overview: I&R Egypt provides customizable smart home, security, and heating solutions, focusing on modular systems that cater to both residential and commercial clients. It emphasizes flexibility and energy efficiency, offering tailored automation for Egyptian households.

Features:

- Access Control: Smart locks and programmable keypads for secure entry, supporting PIN-based and app-based unlocking. A homeowner uses a keypad to grant temporary access to a guest.
- Lighting and Climate Control: Automated lighting, blinds, and HVAC systems for energy efficiency. Users can schedule lights and adjust AC settings via a mobile app.
- Security Systems: Digital surveillance cameras and motion sensors with real-time alerts. A user monitors their home remotely during a vacation.
- Entertainment Systems: Multi-room audio/video integration for seamless entertainment. A family streams music across rooms using a single app.
- Energy Efficiency: Systems designed to reduce power consumption through automation and scheduling. A user saves 15% on electricity by automating blinds to reduce heat gain.
- Customizable Solutions: Modular systems allow users to select specific features, such as lighting or security, based on budget and needs.

Case Study: A villa owner in New Cairo installed I&R Egypt's smart lighting and security system. They used the app to schedule lights and blinds, reducing energy costs by 20%, and monitored cameras during a trip abroad, ensuring peace of mind. The modular design allowed them to add audio systems later.

Technical Insights: I&R Egypt uses Wi-Fi and Bluetooth Low Energy (BLE) for device communication, with a cloud-based platform for app integration. The system supports OTA updates and integrates with third-party devices like smart thermostats.

Comparison to Our System: I&R Egypt's lighting and security features align with our ESP32 and Arduino systems, but it lacks advanced biometrics and fire detection. Our compound website offers unique scalability for multiple homes.

2.2.3 Companies in the Gulf (UAE)

2.2.3.1 Flipsmart (UAE)

Overview: Flipsmart offers energy-efficient and secure smart home solutions in the UAE, emphasizing fast installation and responsive support. It integrates appliances, lighting, and security via IoT platforms for user-friendly automation.

Features:

- Energy Management: Real-time tracking and optimization of lighting, appliances, and HVAC. The app provides insights to reduce usage, such as scheduling AC during peak hours, saving 10% on bills.
- Security Systems: Smart cameras and locks with motion detection, cloud storage, and keyless entry. A homeowner checks a backyard motion alert and confirms it's a pet via the app.
- Cross-Device Control: Integrates devices through a single app using Wi-Fi or Zigbee, requiring no hub. A user turns on a coffee maker, adjusts lights, and unlocks the door with one command.
- Fast Installation and Support: Systems are installed in hours, with insured teams and responsive aftercare. A family sets up lighting and security in a day, with prompt support for issues.

Case Study: A Dubai resident installed Flipsmart's system in their apartment, automating lighting and AC to reduce energy costs by 12%. The security cameras provided real-time alerts during a business trip, enhancing safety. Flipsmart's team resolved a connectivity issue within 24 hours.

Technical Insights: Flipsmart uses MQTT and Zigbee for device communication, with a cloud-based backend for scalability. The app employs WebSocket for real-time updates, ensuring low-latency control.

Comparison to Our System: Flipsmart's energy management and app control are similar to our ESP32 and Flutter app, but it lacks biometric authentication and fire detection. Our compound website offers unique scalability.

2.2.3.2 HomeIQ (UAE)

Overview: HomeIQ, a certified Nest PRO installer in the UAE, specializes in connected devices like Nest thermostats and Ring security systems, offering professional installation and long-term support.

Features:

- Smart Thermostats: Nest thermostats learn user routines to optimize heating/cooling, adapted for UAE's AC systems. A homeowner saves energy by lowering AC when away.
- Security Systems: Ring cameras and locks with 1080p video, motion detection, and keyless entry. A user remotely unlocks the door for a delivery via the Ring app.
- Professional Installation and Support: Includes setup, updates, and maintenance with clear pricing. A customer receives a thermostat update for improved performance.
- Voice Control: Integrates with Google Assistant for hands-free operation, such as adjusting temperatures or checking cameras. A user says, "Hey Google, show the front door."

Case Study: A family in Abu Dhabi installed HomeIQ's Nest thermostat and Ring cameras. The thermostat reduced AC usage by 15% by learning their schedule, and the cameras alerted them to a delivery, which they managed remotely. HomeIQ's team provided regular updates, ensuring reliability.

Technical Insights: HomeIQ uses Google's cloud infrastructure for Nest devices, with Wi-Fi for connectivity. Ring devices employ AES-128 encryption for secure video streaming and app control.

Comparison to Our System: HomeIQ's thermostat and security features align with our Raspberry Pi-based fire detection and app control, but it relies on third-party brands, unlike our custom hardware. Our biometric authentication and compound website add unique functionality.

2.2.3.3 Alayoubi Technologies (UAE)

Overview: Alayoubi Technologies provides automation for homes and businesses in the UAE, offering wireless and hardwired solutions with a focus on flexibility and luxury.

Features:

- Lighting Control: Automated lighting with scheduling, dimming, and color adjustments. Users create scenes like "Evening" for warm lighting via a mobile app.
- Security Systems: Smart cameras and locks with HD video and keyless entry. A business owner monitors office entrances remotely with real-time alerts.
- Climate Control: Smart thermostats for precise HVAC management, reducing energy costs. A user sets the AC to 24°C when occupied.
- Flexibility: Supports new builds and retrofits, with wireless options for easy installation and hardwired systems for reliability.

- Mobile App Control: A unified app for managing all devices, with a user-friendly interface. A homeowner controls lights, AC, and cameras from one platform.
- Professional Support: Offers installation and maintenance, with tailored solutions for each client.

Case Study: A luxury villa in Dubai installed Alayoubi's wireless lighting and security system. The owner used the app to schedule lights and monitor cameras, reducing energy costs by 10% and enhancing security during travel. Alayoubi's team customized the system for the villa's unique layout.

Technical Insights: Alayoubi uses Zigbee and Wi-Fi for wireless systems and proprietary hardwired protocols for robust installations. The app integrates with a cloud server for real-time control and OTA updates.

Comparison to Our System: Alayoubi's lighting and security features are similar to our ESP32 and Arduino systems, but it lacks biometric authentication and fire detection. Our compound website provides a unique management layer for multiple homes.

2.2.4 Companies in the United States

2.2.4.1 Google Nest (USA)

Overview: Google Nest offers a comprehensive smart home ecosystem in the USA, integrated with Google Assistant, focusing on user-friendly automation, energy efficiency, and security.

Features:

- Smart Thermostats: Nest Learning Thermostat optimizes heating/cooling based on user habits, controlled via the Google Home app. It reduces energy bills by 15% by adjusting temperatures when the house is empty.
- Security Systems: Nest cameras, doorbells, and locks with 1080p video, motion detection, and keyless entry. A homeowner views a visitor's feed and unlocks the door remotely.
- Smart Speakers and Displays: Nest Hub/Mini control devices via voice commands and display statuses. A user controls lights and cameras with "Hey Google, show the front door."
- Lighting Control: Supports smart bulbs (e.g., Philips Hue) for scheduling and dimming. A "Morning" routine turns on kitchen lights at 7 AM.
- Energy Efficiency: Tracks usage and suggests optimizations via the app. A user adjusts AC settings based on high-usage reports.

Case Study: A family in California used Google Nest's thermostat, cameras, and Hub to automate their home. The thermostat saved 12% on energy by adjusting temperatures, and the cameras alerted them to a package delivery. The Hub provided a centralized control point, enhancing convenience.

Technical Insights: Google Nest uses Wi-Fi and Thread protocols, with a cloud-based backend for app integration. Devices employ AES-256 encryption for secure communication, and the Google Home app supports WebSocket for real-time updates.

Comparison to Our System: Google Nest's features resemble our Raspberry Pi and Flutter app functionalities, but it lacks custom biometric authentication and fire detection. Our compound website provides a unique management layer.

2.2.4.2 Samsung SmartThings (USA)

Overview: Samsung SmartThings is an open platform connecting various devices, emphasizing flexibility, automation, and energy monitoring via a hub and app.

Features:

- Central Hub: SmartThings Hub (e.g., Aeotec) connects devices via Zigbee/Z-Wave, supporting non-Samsung brands. A user connects a Yale lock and Philips Hue bulbs.
- Mobile App Control: Manages lights, thermostats, and locks with a clean interface for room-based grouping. A "Welcome Home" routine unlocks the door and turns on lights.
- Security Systems: Integrates motion sensors and cameras with real-time alerts. A sensor triggers a camera to record and notify the user.
- Automation Routines: Customizable triggers for multi-device actions, like locking doors at 10 PM. Supports time- or location-based automation.
- Energy Monitoring: Tracks appliance consumption with optimization suggestions. A user adjusts refrigerator settings to save power.

Case Study: A homeowner in Texas used SmartThings to connect lights, locks, and sensors. A routine turned on porch lights and unlocked the door when they arrived home, and energy monitoring reduced appliance usage by 10%. The hub's compatibility with multiple brands ensured flexibility.

Technical Insights: SmartThings uses Zigbee, Z-Wave, and Wi-Fi, with a cloud-based MQTT backend for app control. The hub supports OTA updates, and the app uses RESTful APIs for device management.

Comparison to Our System: SmartThings' hub and app control are similar to our ESP32 and Flutter app, but it lacks biometric authentication and fire detection. Our compound website offers unique scalability.

2.3 Analysis and Implications for Our System

2.3.1 Feature-by-Feature Comparison

To inform our system's design, we compared the features of the researched companies with our proposed system:

- Smart Lighting Control:
  - Market Standard: Cordless, Teknetic, Flipsmart, Alayoubi, Google Nest, and SmartThings offer remote control, dimming, and scheduling, typically via Wi-Fi or Zigbee. Cordless and Flipsmart emphasize plug-and-play simplicity, while Teknetic and Alayoubi support KNX for large setups.
  - Our System: Uses ESP32 for Wi-Fi-based LED control with motion sensors (IR, PIR), matching industry standards. Our scheduling and automation capabilities are comparable, but we prioritized affordability using open-source hardware.
- Security Systems:
  - Market Standard: All companies provide cameras, motion sensors, and smart locks, with real-time alerts via mobile apps. Google Nest and HomeIQ use encrypted video streaming, while Teknetic and I&R Egypt offer keyless entry options.
  - Our System: Includes Arduino-based access control with fingerprint and keypad authentication, plus Raspberry Pi-based facial recognition, offering a dual-biometric approach not found in competitors. Our fire detection adds a unique safety layer.
- Energy Efficiency:
  - Market Standard: Cordless, Flipsmart, Google Nest, and SmartThings provide energy tracking, while HomeIQ and Alayoubi focus on thermostat-based optimization. Most use scheduling to reduce consumption.
  - Our System: Matches competitors with ESP32-based lighting automation and energy tracking, but our focus on motion-based control enhances efficiency for smaller households.
- User Interfaces:
  - Market Standard: All companies offer mobile apps, with Google Nest and SmartThings supporting voice control via Google Assistant or Alexa. Teknetic's KNX panels and Cordless's Arabic-friendly app cater to specific audiences.
  - Our System: The Flutter app provides cross-platform control, similar to competitors, but our compound website offers unique scalability for managing multiple homes, a feature absent in most systems.
- Scalability and Support:
  - Market Standard: Cordless and I&R Egypt emphasize modularity, Flipsmart and HomeIQ focus on fast installation, and Teknetic and Alayoubi offer professional support. Google Nest and SmartThings provide broad ecosystems.

    - Our System: Designed for scalability with a compound website, using affordable hardware (Arduino, ESP32) to keep costs low, unlike the high-end KNX systems of Teknetic.

2.3.2 Market Trends and Gaps

- Egypt: The market prioritizes affordability and energy efficiency due to rising utility costs and limited budgets. Cordless and I&R Egypt address this with plug-and-play solutions, but biometric authentication and safety features like fire detection are rare, presenting an opportunity for our system.
- UAE: Luxury and scalability dominate, with Flipsmart, HomeIQ, and Alayoubi offering high-end automation for villas and businesses. However, compound-wide management is uncommon, making our website a differentiator.
- USA: The mature market focuses on interoperability and voice control, as seen in Google Nest and SmartThings. Our custom hardware and biometric features offer a niche advantage for security-conscious users.

2.3.3 Implications for Our System

The market research shaped our system's design by:

- Adopting Proven Features: We incorporated smart lighting, security, and mobile app control, inspired by Cordless, Google Nest, and others, ensuring our system meets user expectations.
- Innovating with Unique Features: Our biometric authentication (fingerprint, facial recognition) and fire detection address gaps in competitors' offerings, enhancing security and safety.
- Balancing Cost and Innovation: Using Arduino, ESP32, and Raspberry Pi kept costs low compared to KNX systems (Teknetic) or third-party ecosystems (HomeIQ, Google Nest).
- Scalability for Compounds: The compound website addresses a market gap, particularly in Egypt and the UAE, where multi-home management is less common.
- Team Role Alignment: Tasks were assigned based on passion—e.g., embedded systems for Arduino enthusiasts, Flutter for app developers, and Python/OpenCV for AI-focused members—ensuring learning and career growth.

2.3.4 Challenges and Solutions

- Challenge: Matching the polished apps of Google Nest and SmartThings with limited resources.
    - Solution: Used Flutter for cross-platform development, reducing costs while achieving a user-friendly interface.
- Challenge: Competing with established security features like Ring cameras (HomeIQ).

    - Solution: Implemented dual biometric authentication and fire detection for a unique security proposition.
- Challenge: Scaling to multiple homes like KNX systems (Teknetic).
    - Solution: Developed a compound website using Firebase and React, leveraging affordable cloud infrastructure.

2.4 What did we choose

The analysis of Cordless, Teknetic, I&R Egypt, Flipsmart, HomeIQ, Alayoubi Technologies, Google Nest, and Samsung SmartThings provided critical insights into the smart home market. Their features—smart lighting, security, energy efficiency, and user interfaces—informed our system's design, ensuring it meets industry standards while introducing innovative elements like biometric authentication and fire detection. By aligning tasks with team members' passions and leveraging affordable technologies, we created a system that competes with commercial solutions despite our limited resources. This research not only guided our feature selection but also highlighted opportunities to differentiate our system in Egypt, the UAE, and beyond, particularly for residential compounds.

So we chose these features :

- **Light and comfort system** : smart lighting Automated control of indoor and outdoor lighting based on motion detection and ambient conditions .**And** Ventilation Management: Adaptive fan control to decrease temperature in hot weather and optimize air circulation and indoor air quality.
- **Security System**: A multi-layered approach that combines Fingerprint ,Password, door access control, face detection, and live monitoring.
- **Facial Recognition**: AI-powered face authentication for secure and to allow people in the home to know who is knocking the door.
- **Fire and Gas Detection**: Real-time AI-based fire detection using image processing, alongside gas sensors to identify hazardous leaks.
- **Mobile Application**: A user-friendly Flutter-based app that allows remote monitoring, notifications, and full system control.
- **Web Dashboard** : A responsive web interface for data visualization about the compound where the home is in .

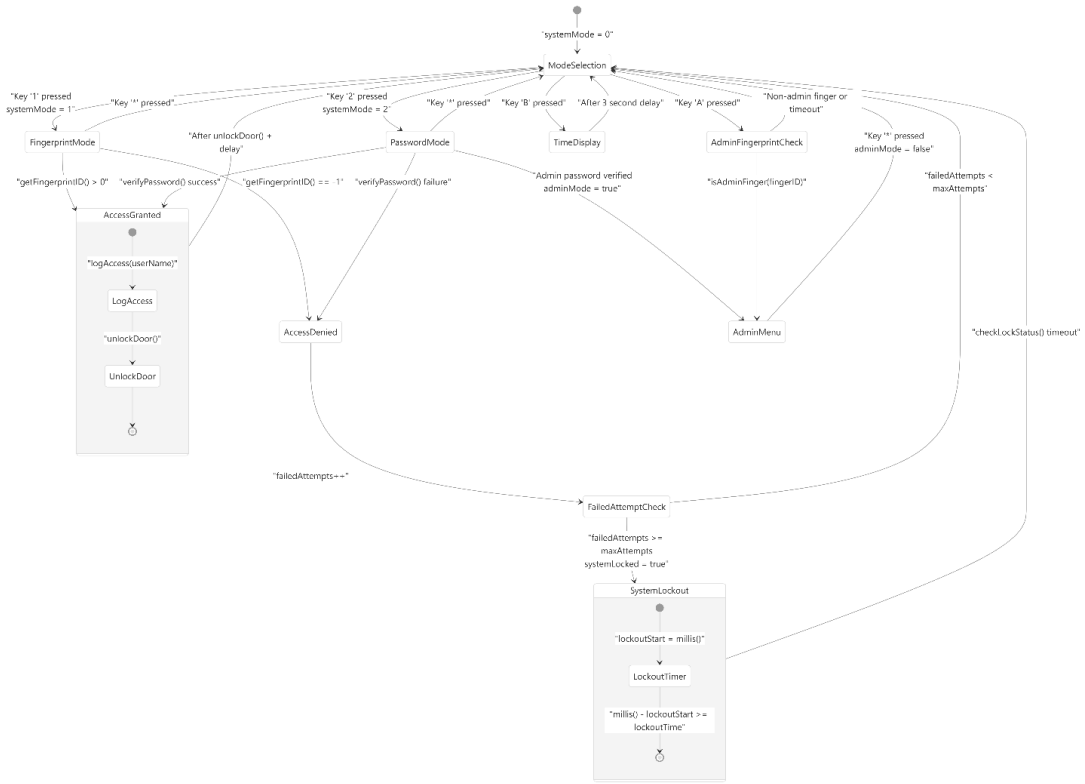
Chapter 3. Technical Implementation

3.1 Smart Security System

3.1.1 Overview

The Smart Security System is a vital subsystem within our overall Smart Home project. Its main purpose is to control access to the home using both fingerprint and password authentication, while providing robust security features such as access logging, lockout on repeated failed attempts, and a user-friendly interface with LCD feedback. The system is implemented on Arduino hardware, utilizing several embedded modules and sensors to ensure both security and ease of use for all family members.

3.1.2 Diagram of features



3.1.2.1 Authentication Modes

3.1.2.1.1 Fingerprint Authentication

Description:

The system covers the fingerprint authentication subsystem of the Smart Home Security System, including biometric sensor communication, user verification, fingerprint enrollment and deletion, and admin access control. The fingerprint system provides the primary biometric authentication method alongside password-based authentication.

Implementation:

- The fingerprint sensor is connected to the Arduino Mega via Serial1.

- During setup, the sensor is initialized and verified for connectivity.

- Each user’s fingerprint is enrolled with a unique ID (1 for Admin, 2 for Father, etc.).

- When a user selects "Fingerprint Mode" on the LCD, the system prompts a finger.

- If the fingerprint matches a stored ID, the system unlocks the door, logs the access with a timestamp, and provides feedback via LCD, buzzer, and LED.

- If authentication fails, an error beep is played and the failed attempts counter increases.

- After three consecutive failures, the system locks out for 10 seconds.

3.1.2.1.2 Password Authentication

Description:

As an alternative to fingerprint access, the system enables users to authenticate using personal 4-digit passwords entered via a 4x4 keypad. Each family member has a unique password, increasing flexibility and ensuring system usability if the fingerprint sensor is unavailable.

Implementation:

- The keypad is connected to the Arduino Mega, mapped to accept numerical and function keys.

- Each user is assigned a unique 4-digit password, stored in code variables (e.g., "1111" for Father).

- On selecting "Password Mode" from the LCD, users can enter their password.

- The system masks input with asterisks (*) for privacy.

- On correct entry, the door unlocks, access is logged, and feedback is given via LCD, buzzer, and LED.

- Incorrect attempts increment the failed attempts counter, triggering lockout after three failures.

3.1.2.1.3 Face Recognition Authentication

The face recognition authentication subsystem enhances the Smart Home Security System by using a camera module for real-time facial recognition, enabling automatic door unlocking for authorized users. Integrated with the live monitoring camera, this feature provides a seamless and convenient access method, complementing fingerprint and password authentication. The system identifies users by comparing captured facial images against a stored database, ensuring high security while eliminating the need for manual input in certain scenarios. This mode is particularly useful for hands-free access or when users are unable to use the keypad or fingerprint sensor.

3.1.2.2 Access Control & Door Operation

3.1.2.2.1 Relay-Controlled Door Lock

Description:

The physical door lock is controlled electronically using a relay module. The relay acts as a switch, enabling or disabling the electric strike or actuator that secures the door.

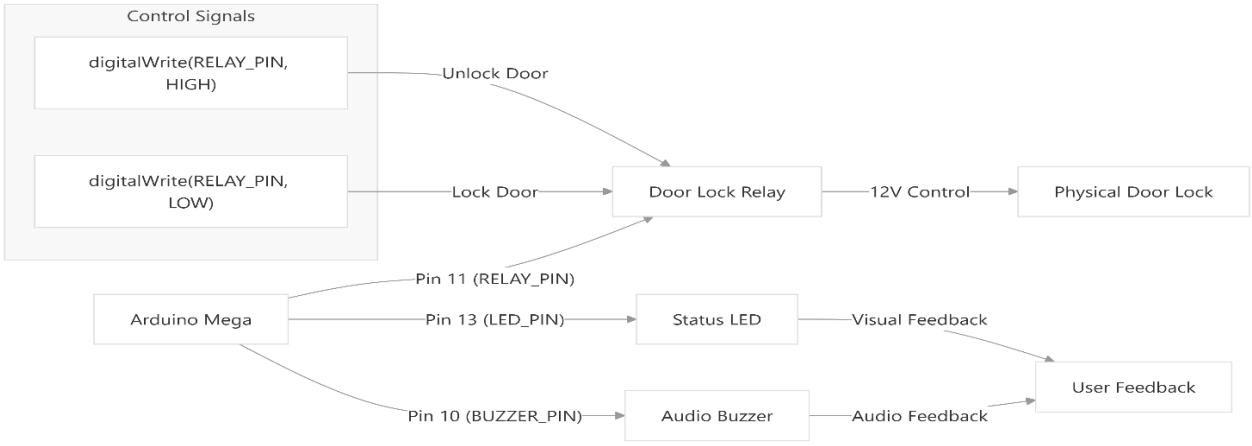
Implementation:

- The relay is wired to a digital output pin on Arduino Mega.

- After successful authentication, the relay is activated (set HIGH), unlocking the door for 5 seconds.

- The status LED turns on to indicate the door is unlocked.

- After 5 seconds, the relay deactivates (set LOW), relocking the door and turning off the LED.



3.1.2.2.2 Visual and Audible Feedback

Description:

To enhance user experience and security awareness, the system uses both an LED and a buzzer to signal status changes (success, failure, alarm).

Implementation:

- The LED is turned on during door unlock and off when locked.

- The buzzer plays different tones for success (short, high-pitch), error (long, low-pitch), and alarm (series of quick beeps for lockout).

- Buzzer and LED are controlled through digital output pins.

3.1.2.3 User Roles & Admin Functions

3.1.2.3.1 User Roles

Description:

The system supports five user roles (Admin, Father, Mother, Son, Daughter) with both biometric and password-based authentication methods.

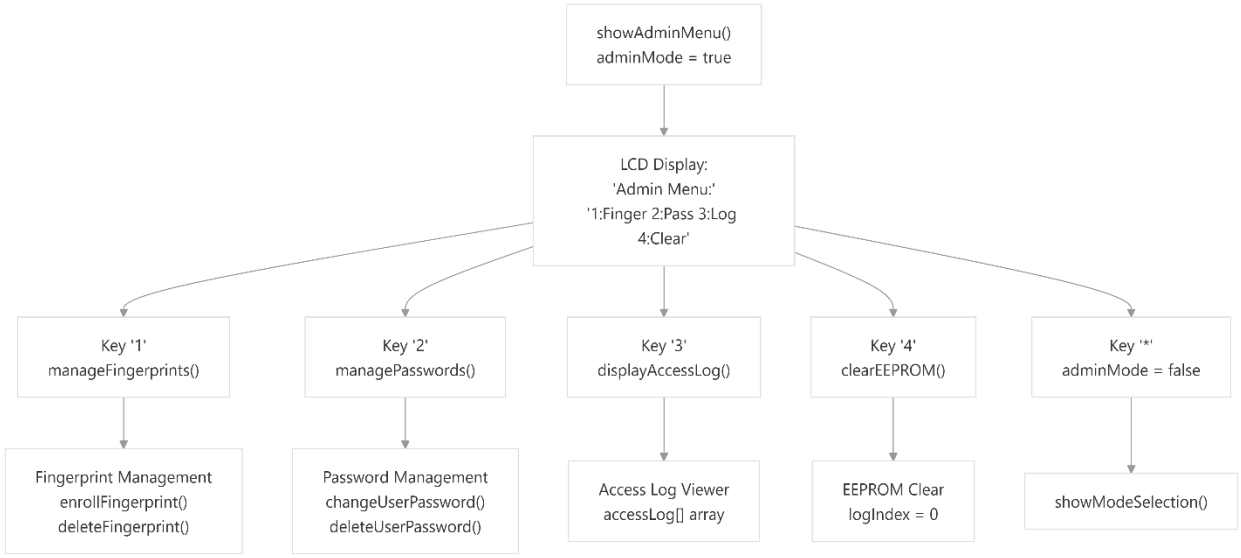
Implementation:

- User roles are defined via password and fingerprint ID assignments in the code.

3.1.2.3.2 Admin Menu and Management

Description:

The admin user can manage the system directly from the keypad and LCD interface, including enrolling or deleting fingerprints, changing or deleting passwords, viewing access logs, and clearing all logs.



Implementation:

- Admin enters admin mode via fingerprint or master password.
- Admin menu appears on the LCD with options:
  - Manage fingerprints (enroll, delete)
  - Manage passwords (change, delete)
  - View access logs
  - Clear access logs/EEPROM
- Navigation and actions are performed using the keypad.

3.1.2.4 Security Features

3.1.2.4.1 Failed Attempt Lockout

Description:

For enhanced security, the system temporarily locks itself out after three consecutive authentication failures, preventing brute-force or guessing attacks.

Implementation:

- Failed attempts are counted for both fingerprint and password modes.

- On reaching 3 failures, the system sets a lockout flag and starts a 10-second timer.

- LCD displays “System Locked! Wait X sec”.

- Buzzer plays alarm sequence.

- System resets failed attempts counter and returns to normal after timeout.

3.1.2.4.2 System Status Display

Description:

A 16x2 LCD provides real-time feedback and user instructions, improving usability and transparency for all users.

Implementation:

- LCD displays prompts for mode selection, authentication, admin actions, and error messages.

- During lockout and errors, special messages are displayed.

- All interactions are guided via LCD.

3.1.2.5 Access Logging:

3.1.2.5.1 Logging Access Events

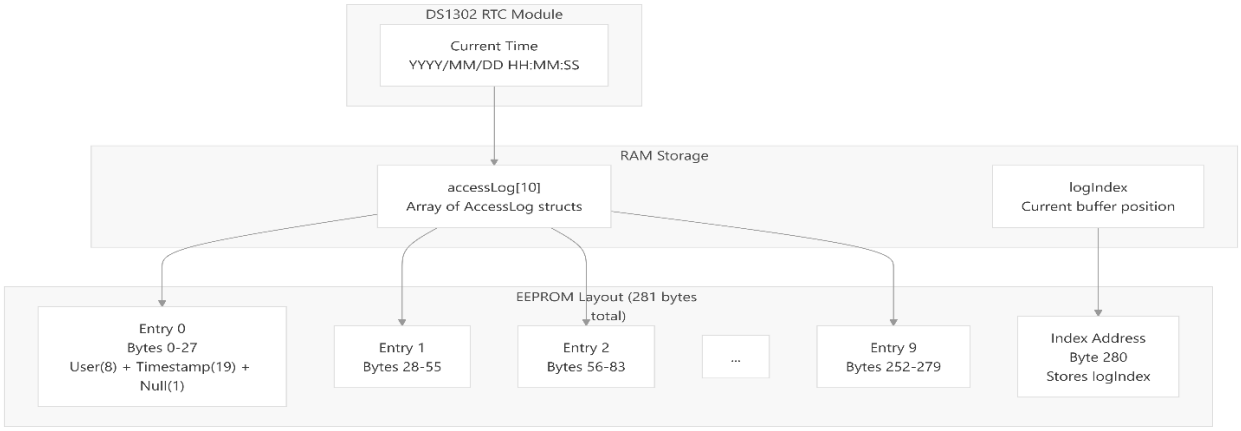
Description:

Every successful access attempt is logged with user identity and timestamp, enabling audit and monitoring of entry events.

Implementation:

- After each successful authentication, the user's name and the current time (from the RTC) are stored in an array.

- The access log is displayed via the LCD (admin only) and stored in EEPROM for persistence.



3.1.2.5.2 EEPROM Storage of Logs

Description:

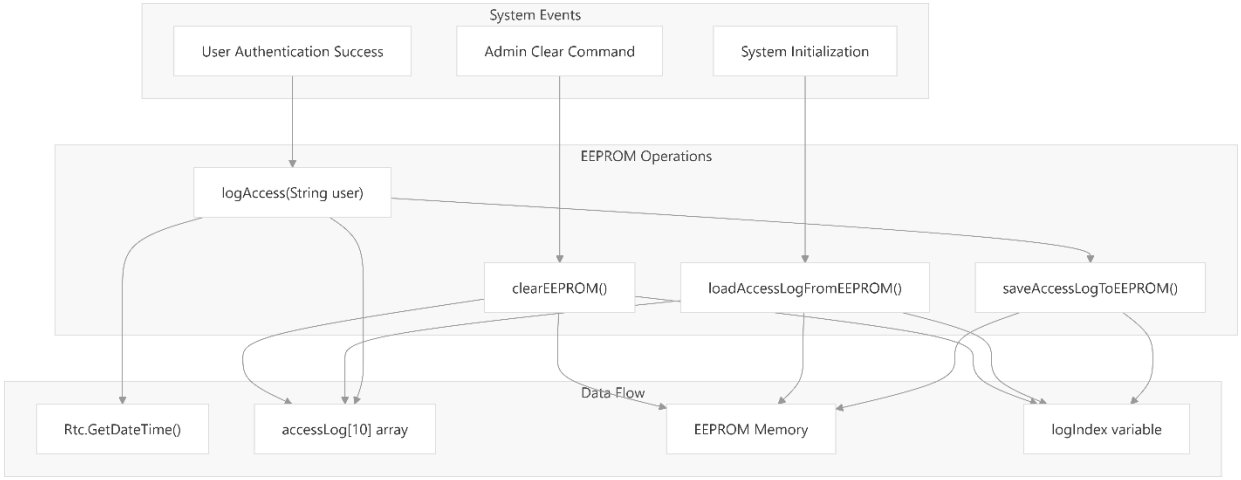
The system stores up to 10 access events in EEPROM, surviving power loss and reset, ensuring that the log is always available for review.

Implementation:

- Each log entry consists of a user name and timestamp.

- Circular buffer logic overwrites oldest entries when full.

- Admin can clear logs via the menu.



3.1.2.6 Hardware Initialization and Error Handling

3.1.2.6.1 Startup Checks

Description

On system boot, all hardware modules (RTC, fingerprint sensor, LCD) are checked for proper connection and functionality. Errors halt normal operation and instruct the user.

Implementation:

- During setup (), the code verifies RTC and fingerprint sensor.

- If an error is detected, LCD displays error message and system halts with LED/buzzer alerts.

- Only when all modules are working does the system proceed to normal operation.

3.1.2.6.2 User Instructions

Description:

The system guides users at every step with clear messages on the LCD, reducing confusion and mistakes during authentication and management.

Implementation:

- All user interactions, mode selections, and admin tasks are prompted through the LCD interface.

- Keypad is used for navigation and selection.

3.1.3 System Implementation

3.1.3.1 Hardware and Software Components

Description:

The Smart Security System is built around the Arduino Mega microcontroller, integrating a fingerprint sensor, 4x4 keypad, I2C LCD, relay module, buzzer, LED, RTC DS1302, and EEPROM. Each component has a clear role, contributing to the authentication, user interaction, and security features.

Implementation:

- Arduino Mega serves as the main controller, orchestrating all inputs and outputs.

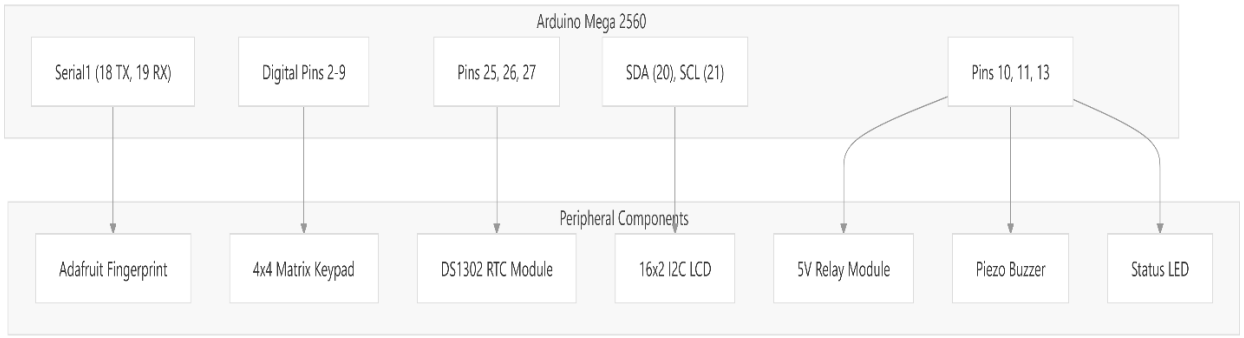
- The fingerprint sensor is connected via Serial1, handling user identification.

- The keypad allows password entry and navigation through admin menus.

- The LCD displays prompts and feedback to users.

- The relay controls the door lock, while the LED and buzzer provide visual and audible status indications.

- The RTC module timestamps access events, and the EEPROM stores logs for persistence.



3.1.3.2 Software Structure and Main Functions

Description:

The software is structured in a modular fashion within a single Arduino sketch. Functions are grouped by their purpose: initialization, authentication, admin management, logging, user interface, and security.

Implementation:

- The setup() function initializes all hardware modules, verifies connections, and prepares the LCD for user interaction.

- The main loop() is responsible for checking the system's lockout status, handling mode selection, and directing the user through fingerprint or password authentication.

- Authentication is managed by handleFingerprintMode() and handlePasswordMode(), with verification functions for both credentials.

- Admin management functions allow enrollment and deletion of fingerprints, password management, log review, and EEPROM clearing.

- Logging functions record each successful access with a timestamp, storing up to 10 events in a circular buffer in EEPROM.

- Security functions track failed attempts, trigger lockout, and produce alarm signals.

3.1.4 Operational Scenarios

3.1.4.1 Normal Access

A typical scenario involves a user selecting an authentication mode on the LCD, providing either a fingerprint or a password, and upon successful verification, the door unlocks temporarily. The system logs the event and provides both visual and audible confirmation.

3.1.4.2 Admin Management

When an administrator authenticates, they gain access to a management menu where they can enroll new fingerprints, delete existing ones, change passwords, review access logs, or clear the log memory. All actions are confirmed with feedback on the LCD and via the buzzer.

3.1.4.3 Failed Authentication and Lockout

If a user fails to authenticate three times consecutively, the system enters a lockout state. During this period, further attempts are ignored, the LCD displays a warning message, and the buzzer emits a series of alarm tones. After the timeout, normal operation resumes.

3.1.4.4 Hardware Error Handling

At startup, the system checks all critical hardware. If a module such as the fingerprint sensor or RTC is not detected, the system halts and displays a persistent error message on the LCD, accompanied by LED or buzzer alerts, until the issue is resolved.

3.1.5 Design Choices and Notes

This system uses a dual authentication approach to balance security and usability, ensuring that access remains possible even if one method fails. Persistent logging in EEPROM provides an audit trail for all access events, and the admin menu centralizes management tasks for ease of use. The lockout feature is crucial for defending against brute-force attempts and is clearly communicated to the user via both the LCD and audible signals.

3.1.6 Suggestions for Figures and Images

- Hardware block diagram with all components labeled.

- Circuit diagram for the door lock relay and feedback devices.

- Flowchart of the main software logic.

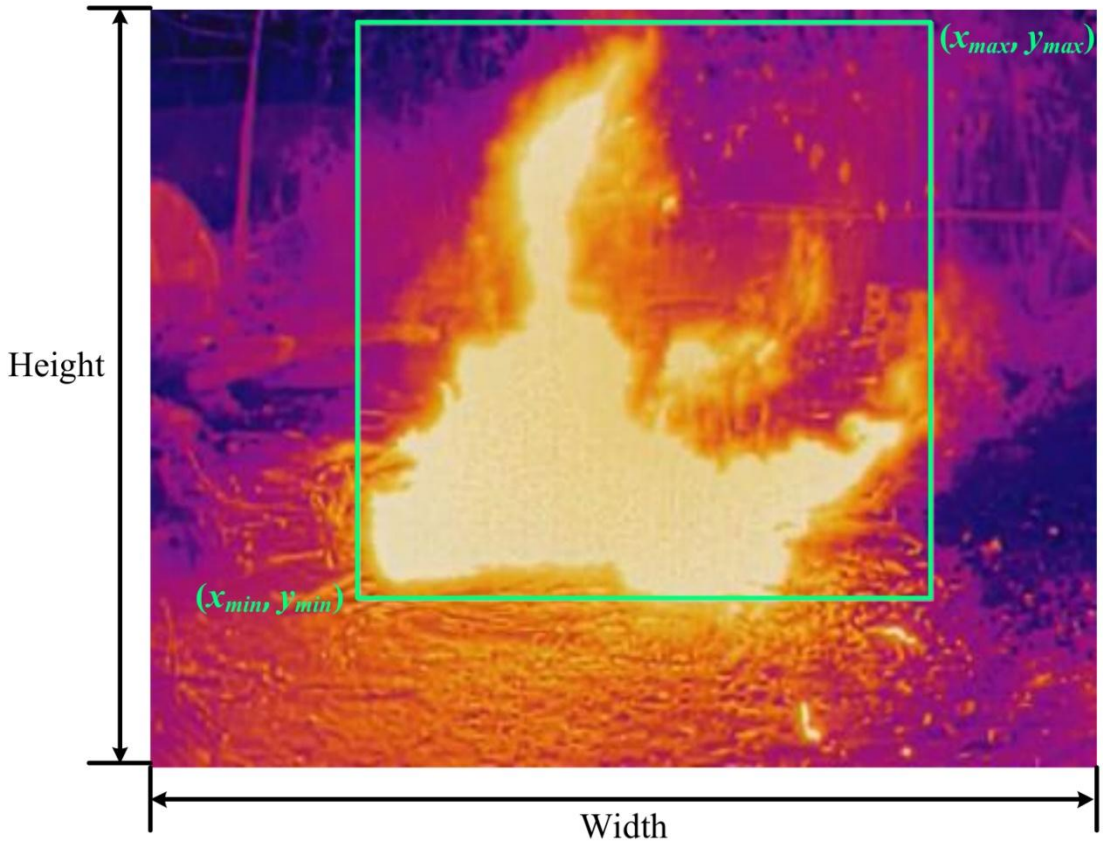
- Screenshots or illustrative mockups of the LCD during key scenarios.

- Table summarizing user roles, credentials, and permissions (with placeholder values for privacy).

- Code snippets for main authentication and logging routines.

3.2 Fire Detection:

Overview



The Fire Detection Module represents a critical safety component within the smart home system, designed to deliver automated, real-time recognition of fire incidents. The system harnesses the power of deep learning, optimized computer vision pipelines, and edge computing to ensure that fire hazards are identified promptly and communicated effectively to end users.

Objectives:

- Develop and deploy an optimized YOLOv8 Nano model trained specifically for fire detection scenarios.
- Implement a Raspberry Pi–based edge inference system capable of processing live video streams in real time.
- Integrate detection outputs with Firebase Realtime Database to enable seamless synchronization with the mobile application.

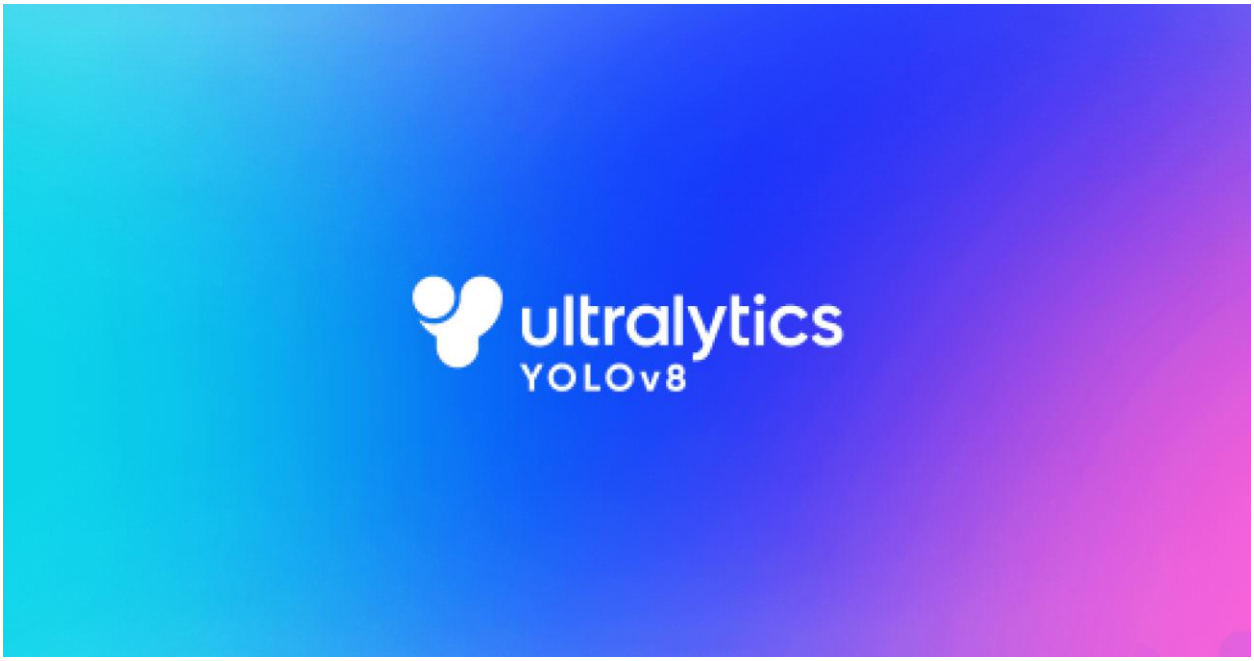
- Provide local audio alerts to warn individuals nearby.
- Deliver annotated video streams over HTTP to facilitate live monitoring.

Model Training and Optimization:

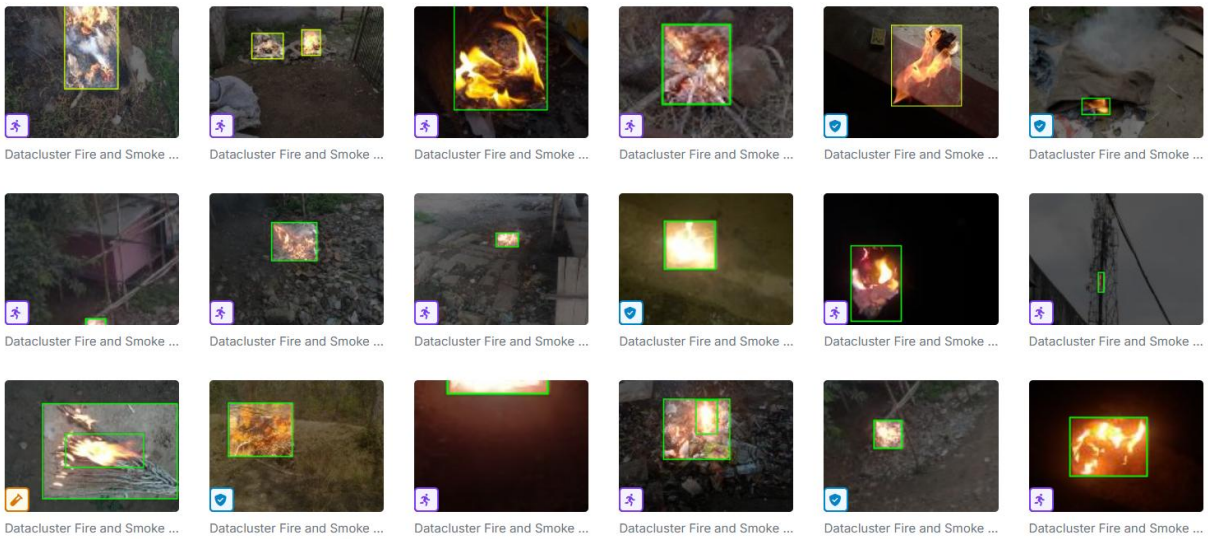
The YOLO model used in this project was based on the YOLOv8 Nano architecture, a highly efficient version of YOLO tailored for environments with limited computational resources. To adapt the model to the fire detection domain, I performed transfer learning using a curated dataset from Roboflow, which contained images of flames in a variety of contexts, including indoor and outdoor scenes, different brightness levels, and varying degrees of occlusion.

Key training details:

- Model Version: YOLOv8 Nano (Ultralytics implementation)



- Dataset Source: Roboflow Fire Dataset



- Training Workflow: Transfer learning with pre-trained weights, applying custom annotations and augmentation techniques
- Output Artifacts: Trained PyTorch weights file (best.pt)
- Model Size: Small footprint optimized for CPU inference on Raspberry Pi

This approach ensured the model achieved a balanced trade-off between accuracy and inference speed, critical for deployment in real-time embedded systems.

System Architecture and Processing Workflow

The fire detection process is composed of multiple tightly integrated stages:

3.2.1. Model Initialization

Upon starting the application, the system loads the trained YOLOv8 Nano weights using the Ultralytics API:

```
# -------------------- YOLO + CAMERA INIT --------------------
try:
    model = YOLO(MODEL_PATH)
    logging.info("[?] YOLO model loaded successfully.")
```

This operation initializes all parameters and prepares the model to perform inference on incoming frames.

3.2.2. Video Acquisition

The Picamera2 library is configured to capture RGB frames from the Raspberry Pi camera:

```
def generate_frames():
    global last_fire_status

    while True:
        frame = picam2.capture_array()
```

The camera resolution is set to 640×480 pixels, which provides sufficient detail while keeping latency low. The camera is initialized with a short warm-up period to stabilize the video stream.

3.2.3. Inference Pipeline

For each captured frame, the model performs inference in streaming mode:

This produces a collection of detection results, each containing:

- The predicted class (in this case, always "fire").
- A confidence score (probability of correct detection).
- Bounding box coordinates outlining the detected region.

To avoid false alarms, detections are filtered using a **confidence threshold of 65%**, ensuring that only high-certainty predictions trigger alerts.

3.2.4. Visual Annotation

For every detection above the threshold:

- A **red bounding rectangle** is drawn around the fire region.
- A **text label** is rendered indicating the class and the confidence percentage (e.g., *fire 85%*).

This annotated frame is used both for local monitoring and for the video stream.

3.2.5. Detection State Management

The system maintains a **status flag** (last_fire_status) to track whether fire is currently detected:

- If fire is detected in the current frame and was not detected in the previous frame:
    - Firebase is updated to set fire_sensor = 1.
    - An audio alert is played through Pygame.
- If no fire is detected after a period where fire was previously identified:
    - Firebase is updated to set fire_sensor = 0.

This mechanism prevents redundant alerts and ensures state transitions are recorded accurately.

3.2.6. Realtime Notifications and Audio Alerts

When fire is detected:

- A **Firebase Realtime Database update** is issued so the connected mobile app can notify users instantly.
- A **pre-recorded audio message** (“Fire detected”) is played via Pygame to warn individuals locally.
- The detection event is logged with timestamps for traceability.

3.2.7. Video Streaming

The processed frames are encoded as JPEG images:

```
_, buffer = cv2.imencode('.jpg', frame)
```

The encoded frames are streamed using a Flask server:

```
http://<Raspberry Pi IP>:5001/fire
```

This allows the mobile application or any client browser to display live annotated video in real time.

Key Features

Feature	Description
Model	YOLOv8 Nano trained on Roboflow fire dataset
Inference Device	Raspberry Pi
Threshold	65% confidence level
Alert Mechanism	Firebase Realtime update + local audio playback
Video Resolution	640×480 pixels
Streaming Protocol	HTTP multipart JPEG stream via Flask
Libraries Used	Ultralytics, OpenCV, Picamera2, Pygame, Firebase Admin SDK, Flask

---

Deployment Workflow

1. **Dependencies Installation:**

```
pip install ultralytics opencv-python picamera2 pygame flask firebase-admin
```

2. **Model Preparation:**
   Place the best.pt file (trained weights) in the project directory.
3. **Firebase Configuration:**
    1. Download and reference the service account JSON file.
    2. Set the database URL and path.
4. **Execution:**

   Run the detection script:

   Python fire_detection.py
5. **Access the Video Stream:**
   Open the provided stream URL in a browser or mobile app.

Performance Considerations

- Average Inference Time: ~100–200ms per frame on Raspberry Pi 4
- Frame Rate: ~5–10 FPS depending on lighting and CPU load
- Latency: Sub-second end-to-end detection and notification

3.3 Face Recognition

3.3.1 Overview

The Face Recognition Module constitutes a vital layer of the smart surveillance system, offering real-time, automatic identification of authorized individuals. Leveraging the face_recognition library in combination with OpenCV and Raspberry Pi camera input, the system is capable of recognizing known faces, triggering external hardware (via Arduino), and logging recognition events to Firebase for synchronized monitoring.

3.3.2 Objectives:

- Train a custom face recognition model using labeled datasets of known individuals.
- Deploy a lightweight, real-time recognition pipeline using the Raspberry Pi and PiCamera2.
- Integrate with Firebase Realtime Database to push streaming URLs and recognition logs.
- Control an Arduino-based lock mechanism depending on recognition outcomes.
- Serve annotated video streams over HTTP using Flask for live monitoring.

3.3.3 Model Training and Encoding

Face encoding is the foundational step in enabling accurate recognition. Using the face_recognition library, we extract 128-dimension facial embeddings for each image in a labeled dataset.

Key training details:

- Dataset Structure: Organized in folders, each named after the person (e.g., dataset/Ahmed/, dataset/Sara/).
- Encoding Tool: face_recognition.face_encodings()

- Detection Model: HOG (Histogram of Oriented Gradients)
- Output Artifacts: encodings.pickle (serialized encodings and labels using pickle)
- Training Script: model_training.py

This process enables the system to associate each embedding with a person's name and later compare real-time input to these encodings.

3.3.4 System Architecture and Processing Workflow

3.3.4.1. Initialization and Loading

- Loads face encodings from encodings.pickle:

```
# -------------------- LOAD ENCODINGS --------------------
logging.info("[INFO] Loading face encodings...")
with open(ENCODINGS_PATH, "rb") as f:
    data = pickle.loads(f.read())
```

- Initializes Firebase and Picamera2 instance.

- Sets up serial connection with Arduino via USB.

3.3.4.2. Frame Acquisition

- Captures frames via Raspberry Pi's camera:
-

```
# -------------------- CAMERA --------------------
picam2 = Picamera2(0)
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888', "size": (640, 480)}))
picam2.start()
time.sleep(2)
```

- Frames are resized and converted to RGB for recognition.

3.3.4.3. Face Detection and Encoding

- Detects all faces and extracts encodings:

```
    rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    boxes = face_recognition.face_locations(rgb, model="hog")
    encodings = face_recognition.face_encodings(rgb, boxes)
```

3.3.4.4. Face Matching

- Compares new face encodings to known ones:

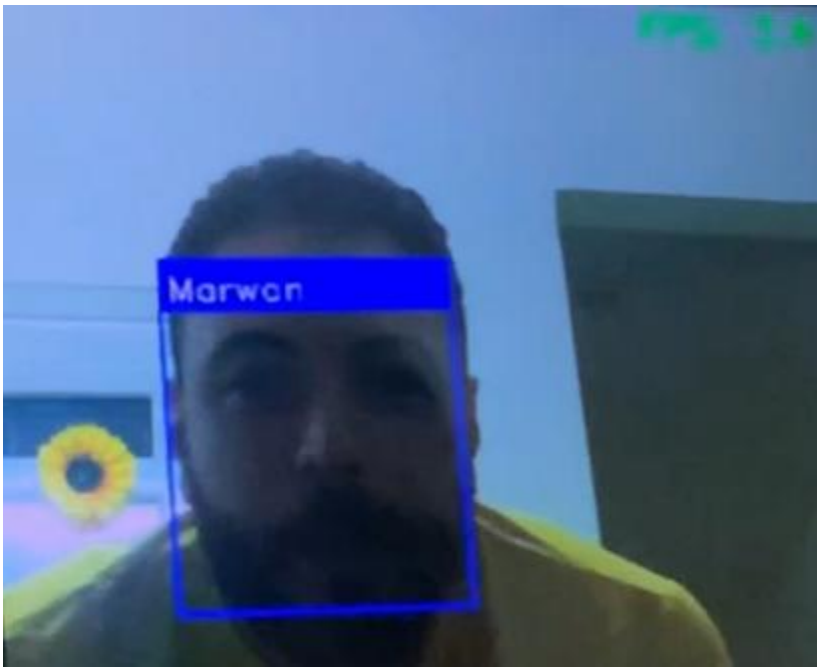
```
  matches = face_recognition.compare_faces(known_face_encodings, enc)
  name = "Unknown"
  face_distances = face_recognition.face_distance(known_face_encodings, enc)
```

3.3.4.5. Visual Annotation

Draws bounding boxes and labels on the frame:

```
cv2.rectangle(frame, (left, top), (right, bottom), (244, 42, 3), 2)
cv2.rectangle(frame, (left, top - 35), (right, top), (244, 42, 3), cv2.FILLED)
cv2.putText(frame, name, (left + 6, top - 6), cv2.FONT_HERSHEY_DUPLEX, 0.8, (255, 255, 255), 1)
```

- FPS is also calculated and displayed.



3.3.4.6. Door Access Control

- Sends command to Arduino when face is recognized:

```
if known_face_detected:
    if last_command != b'1':
        ser.write(b'1')
        logging.info("[INFO] UNLOCK command sent to Arduino.")
        last_command = b'1'
        unlocked_since = now
else:
    if last_command == b'1':
        # Wait relock_delay seconds before locking
        if now - unlocked_since >= relock_delay:
            ser.write(b'0')
            logging.info("[INFO] LOCK command sent to Arduino (no faces detected).")
            last_command = b'0'
            unlocked_since = None
```

- Automatically locks after timeout or if face disappears.

3.3.4.7. Logging and Firebase Integration

- Logs recognized faces with timestamp:

```
def log_faces_to_firebase(faces):
    ts = int(time.time())
    log_ref = db.reference(FACE_LOG_PATH)
    log_ref.set({
        "timestamp": ts,
        "faces": faces
    })
    logging.info(f"[🧠] Logged faces: {faces}")
```

- Pushes live stream URL to Firebase at system start.

3.3.4.8. Web Streaming with Flask

- Streams annotated frames through Flask:

```
# ------------------- ROUTE -------------------
@app.route('/face')
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
```

- Accessible via :

http://<raspberry_ip>:5000/face

3.3.4.9. Workflow Diagram

Camera → Frame Capture → Resize + RGB Conversion

→ Face Detection → Encoding → Matching

→ Draw Boxes → Control Arduino → Log to Firebase

→ Stream to Web

3.3.4.10. Important Variables

ENCODINGS_PATH: Path to encodings.pickle file

cv_scaler: Frame resize factor (e.g. 0.5 for 50%)

ser: Serial object for Arduino communication

door_open_time: Time to keep door unlocked

STREAM_DB_PATH: Firebase path for stream metadata

FACE_LOG_PATH: Firebase path for logging face entrie

3.3.4.11. Output

- Live Video Feed: MJPEG stream via /face endpoint

- Access Control Signal: Sent to Arduino via USB

- Firebase Logs: Logs recognized names and timestamps

3.3.5. Key Features

Feature	Description
Recognition Model	face_recognition library with HOG backend
Inference Device	Raspberry Pi + PiCamera2
Thresholding	Closest match selection using face_distance
Hardware Control	Lock/unlock via Arduino USB Serial

Feature	Description
Logging Mechanism	Firebase Realtime Database
Video Resolution	640×480 pixels (configurable via cv_scaler)
Streaming Protocol	HTTP MJPEG stream via Flask
Libraries Used	face_recognition, OpenCV, Picamera2, Firebase Admin SDK, Flask

---

3.3.6 Deployment Workflow

3.3.6.1. Dependencies Installation:

```
pip install face_recognition opencv-python flask firebase-admin picamera2
```

3.3.6.2. Model Preparation:

- Run model_training.py to generate encodings.pickle.

- Ensure all face images are labeled and organized in folders.

3.3.6.3. Firebase Configuration:

- Download the Firebase service account key (.json).

- Set paths and credentials in face_recog_with_flask.py.

3.3.6.4. Execution:

python3 face_recog_with_flask.py

3.3.6.5. Access the Stream:

Open the stream URL on a browser or mobile app:
http://<raspberry_ip>:5000/face

3.3.7. Security Considerations

- Authentication and Access: Ensure that the Flask stream route is restricted to authorized users only.
- Data Privacy: All captured and logged face data should be encrypted or stored securely to prevent misuse.
- Hardware Fail-Safe: In case of Arduino or communication failure, the system should default to a safe state (locked).
- Firebase Key Protection: Service account credentials must be secured and excluded from public repositories.

3.3.8 Gas Detection Integration

In addition to the face recognition system, a gas detection module was integrated using the **MQ-5 gas sensor** to enhance the safety features of the smart surveillance setup. The module allows real-time detection of flammable gases and provides an immediate audio alert when dangerous levels are detected near the entrance.

3.3.8.1 Sensor Operation

The **MQ-5 gas sensor** was connected to the Raspberry Pi via GPIO pins. This sensor is capable of detecting a range of combustible gases including **LPG, methane (CH_4), hydrogen, and smoke**.

Figure 3.3.8.1: MQ-5 Gas Sensor (TDK Variant) used for real-time gas monitoring.

The sensor continuously monitors gas levels. When gas concentration exceeds a predefined threshold, the system triggers an alert. The logic is implemented in Python using GPIO polling:

```
while True:
    gas_level = GPIO.input(GAS_PIN)
    if gas_level == 0:
        print("Gas detected!")
        play_sound3()
        time.sleep(5)
```

3.3.8.2 Audio Alert System

Upon detection of gas, the Raspberry Pi sends a signal to a connected speaker to play a warning sound:

Audio Output: “⚠️ Warning! Gas is Detected .”

As long as the gas remains present in the environment, the system continuously sends a signal to the speaker to repeat the alert message at regular intervals.

3.3.8.3 Integration with Main System

The gas detection module runs concurrently with the face recognition system on the same Raspberry Pi. Both modules operate independently yet share resources (camera, GPIO, processing), making the setup compact and efficient without the need for separate microcontrollers.

3.4 - Light and Comfort System

The System is a smart home subsystem designed to improve energy efficiency and user comfort by automating the control of lighting and air conditioning.

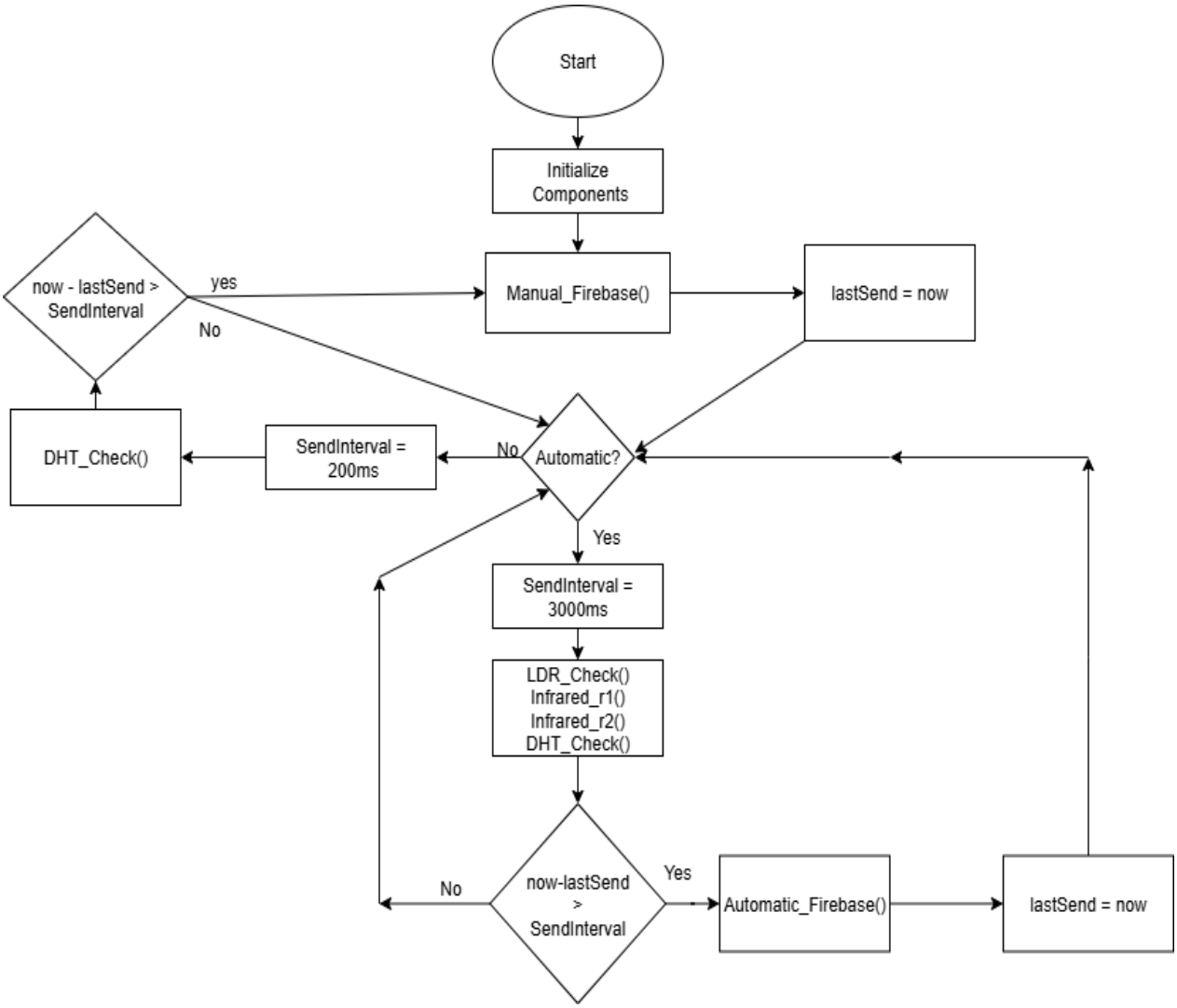
The system consists of :

1- Outing lights
2- Rooms lights
3- Ventilation system

It works in two modes :

1- Automatic mode : depends on reading sensors to control the leds and the air conditioning

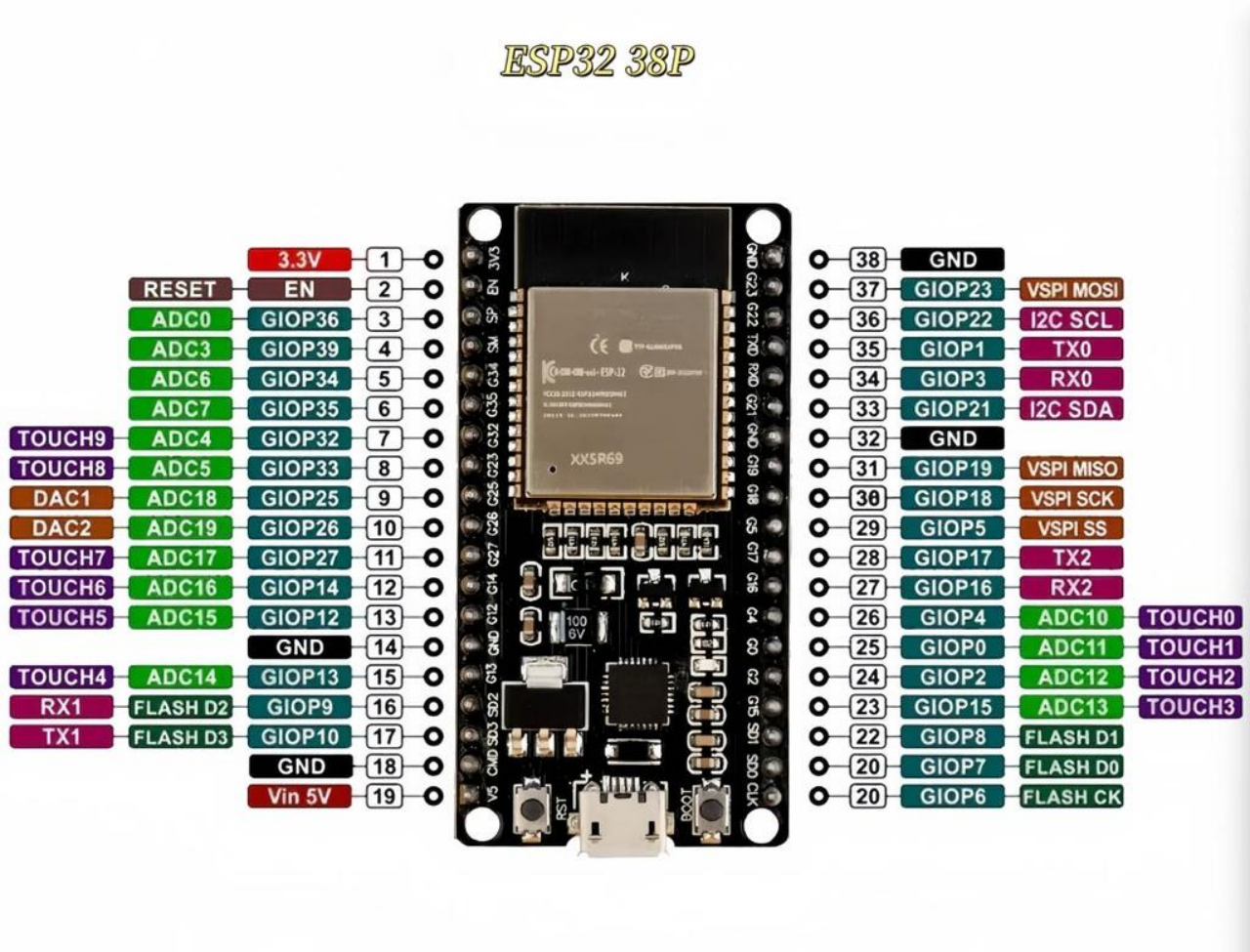
2- Manual mode : user control the lighting and the conditioner manually using his mobile phone



This diagram shows us how the main loop function run to achieve our targets

3.4.1 – Esp32 wroom 38 pin

We decided to use ESP32 microcontroller for this purpose , but why?



The **ESP32** microcontroller was chosen as the main controller for the Light & Climate System due to its advanced capabilities, which are well-suited for smart home applications. It offers a powerful combination of performance, connectivity, and low power consumption, making it ideal for systems that require wireless communication and real-time sensor integration.

Key Features of the ESP32:

1. **Built-in Wi-Fi and Bluetooth:**
   Enables seamless wireless communication with the mobile application through Firebase Realtime Database.

2. **Dual-core processor up to 240 MHz:**
   Provides high performance for handling multiple sensors and processing data in real-time.

3. **Multiple GPIO pins:**
   It has 38 pins most of them can be used for I/O and in Analog/Digital which Allows connection of various components such as motion sensors, temperature/humidity sensors, LEDs, and relays

4. **Support for multiple protocols (I2C, SPI, UART):**
   Makes it easy to interface with a wide range of sensors and modules.

5. **Low power consumption:**
   Ideal for always-on systems and smart devices that need to remain efficient over time.

6. **Easy programming via Arduino IDE:**
   The system was developed using the Arduino IDE, which simplifies coding, testing, and debugging.

All these features could integrate together to achieve high efficiency for our project

3.4.2 – Automatic mode

The Automatic Mode enables the system to operate without manual intervention, relying entirely on sensor inputs to make decisions. When this mode is activated, the light is automatically turned on when motion is detected and people counter has positive value . Similarly, the Ventilation system is controlled based on temperature and humidity readings; it turns on when the environment becomes uncomfortable and turns off when optimal conditions are restored. It also Automatically controls outing lights depends on whether we are in morning , evening or night . This mode is designed to optimize energy consumption and enhance user comfort by responding dynamically to changes in the environment .

Automatic mode sends all data about the lighting , Temperature and humidity to the mobile application via firebase realtime but users doesn't have any control on them .

Automatic mode is more suitable for children or elders who have some difficulty in treating with mobile application so all lights will be turned on and off automatically depends on their presence in the room and their motion .

3.4.2.1 – Outing lights in Automatic mode

Outing lights are the external lights of the house, such as those placed in the garden, garage, or entrance area. In automatic mode, these lights are controlled based on the time of day. A light-dependent mechanism determines whether it is

morning, evening, or night, based on the brightness of natural sunlight or a real-time clock module.

- In morning time: the outing lights remain off, since there is sufficient natural light.
- In evening time: the outing lights turn on automatically to ensure proper visibility and safety, but it doesn't work with all its power as there's some lights from the sun.
- In night time: the outing lights turn on automatically with its highest power to ensure proper visibility and safety.

This automation helps reduce unnecessary power consumption during the day, while enhancing outdoor visibility and security during darker hours.

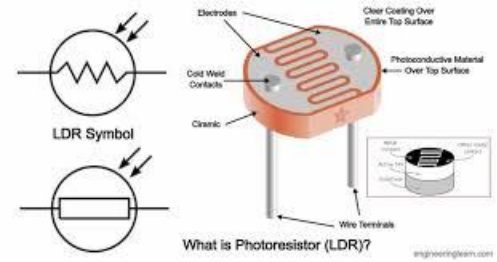
The system can also be configured to gradually increase or decrease brightness depending on the ambient light levels if desired.

We used LDR to know the amount of light from the sun

How LDR Works

An LDR (Light Dependent Resistor) is a type of resistor whose resistance varies depending on the amount of light falling on its surface. It is also known as a photoresistor.

- In bright light conditions: the resistance of the LDR decreases significantly, allowing more current to pass through.
- In darkness or low light: the resistance of the LDR increases, reducing the current flow.



How this was implemented

To implement the outing lights control logic, we used a Light Dependent Resistor (LDR) sensor to detect the ambient light level. The LDR was connected to one of the analog

GPIO pins of the ESP32. The system reads the light intensity and compares it to predefined thresholds to determine whether it is morning, evening, or night.

- If the light level is above the threshold → it is considered morning, and the outing lights remain off.
- If the light level is medium → it is considered evening and the outing lights are automatically turned on with moderate value
- If the light level is low → it is considered night and the outing lights are automatically turned on with high value

  This approach ensures a real-time response to natural lighting conditions without requiring a fixed schedule. The threshold value is calibrated experimentally to match the actual lighting conditions in the environment

```
void LDR_Check(){
  ldrValue = analogRead(LDR_PIN);
  if (ldrValue > 3200) {
    brightness = 0;
    currentLdr = 0;
  } else if (ldrValue > 1600) {
    brightness = 128;
    currentLdr = 1;
  } else {
    brightness = 255;
    currentLdr = 1;
  }ledcWrite(0, brightness);  // channel 0
}
```

3.4.2.2 – Rooms lighting in Automatic mode

The automatic room lighting system is designed to improve convenience and save energy by intelligently turning lights on or off based on occupancy and motion detection.

The system relies on two main inputs:

A) People Counter**:** Keeps track of the number of people currently present in the room.

we implemented this using IR sensors

IR module consists of two limbs . first one the emitter of the infrared light , second one is the receiver to detect the reflection .

In the people counting mechanism, we use two active IR sensors placed at the entrance of the room. The system tracks the sequence in which the IR beams are interrupted:

- If Sensor A is triggered before Sensor B, a person is entering → counter is incremented.
- If Sensor B is triggered before Sensor A, a person is leaving → counter is decremented.

B) Motion Sensor (PIR Sensor): Detects movement within the room.

How It Works:

- When motion is detected and the people counter has a positive value, the room lights automatically turn on.
- If no motion is detected for a specific period the lights will turn off .
- If no motion continues and the people counter becomes zero, the lights will be automatically turned off.

- If there's a motion but peopleCount = 0 it will stay turned off , maybe it's just a pet.

This system ensures that lighting is only active when needed, reducing electricity usage and enhancing user comfort—especially for children, the elderly, or people who may forget to manually control the lights .

```
void Infrared_Check_r1(){
  unsigned long currentTime = millis();

  int ir1State = digitalRead(IR1_r1);
  int ir2State = digitalRead(IR2_r1);

  // IR1
  if (ir1State == LOW && !detected1_r1 && currentTime - lastTrigger1_r1 > debounceDelay) {
    t1_r1 = currentTime;
    detected1_r1 = true;
    lastTrigger1_r1 = currentTime;
    Serial.println("IR1 triggered1");
  }

  // IR2
  if (ir2State == LOW && !detected2_r1 && currentTime - lastTrigger2_r1 > debounceDelay) {
    t2_r1 = currentTime;
    detected2_r1 = true;
    lastTrigger2_r1 = currentTime;
    Serial.println("IR2 triggered1");
  }
```

In this part of code we check each sensor to know if any of them has detected an obstacle . and if that happened we will store the timming

How to prevent debouncing ?

> We store the trigger time at "lastTriggern_rx" to make sure that the current trigger has happened after a small period of last trigger to ensure that it's another person

But what is the functionality of "detectedn_rx" ?

It is the variable we used to check that r1&r2 have been triggered and there's no large delay between them.

Cases :

a. detected1 then detected2 && delay between them < 1500ms -> person entered
b. detected2 then detected1 && delay between them < 1500ms -> person exited
c. detected1 && 1500ms has passed without triggring ir2 -> nothing

d. detected2 && 1500ms has passed without triggring ir1 -> nothing

And after each case of these all variables retuen to zero again

```
detected1_r1 = false;
detected2_r1 = false;
t1_r1 = 0;
t2_r1 = 0;
```

While peopleCounter has positive value it will check the PIR sensor to make sure that the motion is still there and the person in the room hasn't go sleep or stopped doing what he was doing . If the motion is stopped for about 4-6 minutes it will turn the lights off until he makes a motion again

3.4.2.3 - Ventilation system in Automatic mode

The ventilation system, represented by the air conditioning unit in our smart home, operates automatically based on real-time environmental readings to maintain a comfortable indoor climate and optimize energy usage.

The system relies on data from a temperature and humidity sensor DHT11 connected to the ESP32. These readings are continuously monitored and compared against predefined thresholds to decide when the air conditioner or fans should be turned on or off.

How It Works:

- If temperature or humidity levels exceed comfort thresholds, the system automatically turns on the air conditioning/fans to cool and dehumidify the space.
- Once the environmental conditions return to the acceptable range, the air conditioning/fans is turned off to conserve energy.

For example:

- If temperature > 27°C → Turn ON fan/air conditioner.
- If temperature ≤ 27°C → Turn OFF fan/air conditioner.

This automation ensures that the indoor environment remains comfortable without requiring manual intervention. The system is especially beneficial for:

- Users who may forget to adjust the air conditioner.
- Homes with elderly residents or children who require stable environmental conditions

3.4.3 - Manual mode

The Manual Mode allows users to take full control of the Light & Ventilation System through a mobile application. In this mode, automation is disabled, and users can directly manage the system's components such as room lighting, outing lights, and the air conditioning unit.

How It Works:

- The user opens the mobile app and connects to the system via Firebase Realtime Database.
- From the app interface, the user can:
  - Turn ON/OFF room lights.
  - Control the brightness of outing lights.
  - Switch the air conditioner ON or OFF.

Use Cases:

Manual mode is especially useful in the following situations:

- When the user wants to override automatic decisions.
- During specific events or occasions where custom lighting or cooling settings are preferred.
- When remote control is needed (e.g., turning on the air conditioner before arriving home).

Implementation Details:

- The app communicates with Firebase, which updates specific control values in the cloud.
- The ESP32 continuously listens for changes in these values and acts accordingly.

Manual mode provides flexibility and user freedom, allowing the system to adapt to personal preferences while maintaining the benefits of centralized control.

SO, in manual mode all senseors are neglected except DHT11 which still send the temperatue and humidity to the mobile application and control the fan or air conditioner .

3.4.4 - Firebase

Firebase Realtime Database was chosen as the cloud-based backend for our Light & Climate System to enable seamless communication between the mobile application and the ESP32 microcontroller.

Firebase provides a reliable and fast way to store and sync data across all connected clients in real time. This allows users to monitor and control their home environment remotely through their mobile phones, while the ESP32 can instantly reflect any changes made.

We used firebase in both Automatic mode and Manual mode but with different functionality.

1- In Automatic mode we send the decisions that have been taken depends on sensors
2- In Manual mode we allow users to control lights by himself and still read temperature sent by DHT11 sensor

3.5 Mobile Application

The "Smart Home" project is a cross-platform application developed using Flutter. It aims to provide users with an intuitive interface to monitor and control various aspects of a smart home environment, such as sensors and devices, from their desktop or mobile device. The project serves as a practical implementation for integrating IoT concepts and modern app development.

3.5.1 What's Flutter?

Flutter is an open-source UI software development kit (SDK) created by Google. It's primarily used for building **cross-platform applications** from a single codebase. This means you can write your code once and deploy it to multiple platforms, including:

- **Mobile:** Android and iOS
- **Web**
- **Desktop:** Windows, macOS, and Linux

- **Google Fuchsia** (an experimental operating system)

Here are some key aspects of Flutter:

- **Programming Language:** Flutter apps are written in **Dart**, another language developed by Google. Dart is optimized for UI development and offers features like sound null safety.
- **Widgets:** The fundamental building blocks of a Flutter app are "widgets." Everything you see on the screen, from buttons and text to entire layouts, is a widget. Flutter's widget-based architecture allows for highly customizable and reusable UI components.
- **Own Rendering Engine:** Unlike some other cross-platform frameworks that rely on native platform components, Flutter ships with its own rendering engine (primarily written in C++ and using Skia graphics library or Impeller). This gives Flutter complete control over the pixels on the screen, leading to consistent and high-performance UIs across different platforms.
- **Hot Reload:** A significant productivity booster for developers, hot reload allows you to see the effects of your code changes almost instantly, without needing a full recompile or losing the application's current state. This speeds up the development and iteration process.
- **Native Performance:** Because Flutter compiles code directly to machine code (ARM or Intel, and JavaScript for the web), it offers performance that is very close to native applications.
- **Material Design and Cupertino Widgets:** Flutter provides two sets of pre-designed widgets that conform to specific design languages: Material Design (Google's design language) and Cupertino (Apple's iOS Human Interface Guidelines). This allows developers to create apps that feel natural on both Android and iOS, or even mix and match styles.
- **Open Source and Community:** Flutter is open source and has a large, active community, offering extensive documentation, plugins, and support.

In essence, Flutter aims to make it faster and easier to build beautiful, high-performance, and visually consistent applications across various platforms from a single codebase.

3.5.2 Application Objectives

- To design and implement a user-friendly smart home interface.

- To enable real-time monitoring and control of home sensors and devices.
- To leverage Flutter's cross-platform capabilities for both desktop and mobile environments.
- To provide a scalable foundation for future enhancements in smart home automation.

3.5.3 Application Idea

The main idea of the "Smart Home" project is to offer a unified application that allows users to interact with smart devices and sensors in their home. The project integrates multiple features:

- Real-time sensor monitoring (e.g., temperature, lighting).
- Device control (e.g., turning LEDs or appliances on and off).
- User roles (such as "Father" and "Mother" home screens) for personalized control panels.
- Use of modern UI design and state management with Flutter Bloc.

The application demonstrates how smart home solutions can be centralized, making it easier for users to manage their environments efficiently and securely from one place.

3.5.4 Application Overview

Key features include:

- **Multi-role Support:** Different dashboards and permissions for roles like Father and Mother.
- **Real-Time Monitoring:** Live updates from sensors in various rooms and the kitchen.
- **Device Control:** Ability to turn devices (e.g., LEDs) on/off remotely.
- **Room & Zone Management:** Each area (room, kitchen, etc.) has its own screen with controls and data.
- **Safety Integrations:** Fire/gas detection alerts and camera feeds for verification.
- **Face Recognition:** Security feature for identifying individuals at entry points.
- **User Authentication:** Secure registration and login with role assignment.

- **Profile Management:** Users can view and manage their personal info and account settings.
- **Modern UI/UX:** Responsive, dark-themed interface for web, desktop, and mobile.

The application leverages Firebase for authentication and real-time data, Bloc for state management, and clean architectural separation for scalability and maintainability. "Smart Home" demonstrates the practical use of IoT concepts, Flutter's cross-platform capabilities, and user-centric smart home experiences.

3.5.5 Features Overview

3.5.5.1 Multi-Role User System (Father, Mother, etc.)

- **Description:**
  The application supports different user roles (e.g., Father, Mother), each with access to different home screens and controls. This allows for personalized dashboards and permissions based on the role.

- **Implementation:**
  - Users register with a specific role during the sign-up process.
  - Each role sees a tailored home screen, e.g., the father's dashboard includes advanced controls and monitoring options.

3.5.5.2 Real-Time Sensor Monitoring

- **Description:**
  The app monitors environmental sensors (such as temperature, humidity, flame, and gas sensors) in real time and displays their data to the user.
- **Implementation:**
  - Uses Firebase and Firestore as the backend for real-time data updates.
  - The SensorsCubit subscribes to streams from the backend and updates UI widgets instantly.
  - Dedicated dashboards show live readings for rooms and the kitchen.

3.5.5.3 Device Control (Lighting, Appliances, etc.)

- **Description:**
  Users can remotely control home devices, such as turning lights (LEDs) on or off, and potentially control other appliances.
- **Implementation:**
  - Device control is managed through Cubits (e.g., LedsCubit) and repositories, which interface with the backend.
  - The UI provides toggle switches and buttons for each device/room.
  - Device state updates are reflected instantly in the UI.

3.5.5.4 Room and Zone Management

- **Description:**
  The home is divided into logical zones (e.g., Rooms, Kitchen), each with its own controls and sensors.
- **Implementation:**
  - The app has separate screens for each room/zone.
  - Each screen shows the relevant sensors and controls for that area.
  - Navigation between zones is simple and user-friendly.

3.5.5.5 Fire and Gas Detection with Camera Integration

- **Description:**
  The application includes specialized features for safety, such as fire and gas leak detection, and integrates with camera modules for visual confirmation.
- **Implementation:**
  - Fire and gas sensors trigger UI alerts if abnormal readings are detected.
  - Users can access live camera feeds (e.g., FireCam) directly from the application for verification.

3.5.5.6 Face Recognition Security Feature

- **Description:**
  Face recognition is integrated to enhance home security, allowing identification of people at entry points.
- **Implementation:**
  - A dedicated screen (FaceCam) is provided for face recognition.
  - UI allows users to view and manage recognition events.

3.5.5.7 User Registration and Authentication

- **Description:**
  Secure registration and login with role selection, validation, and password management.
- **Implementation:**
  - Uses Firebase Authentication for secure login/registration.
  - Custom registration form collects name, phone, role, building/apartment number.
  - Email verification is required for new accounts.

3.5.5.8 User Profile Management

- **Description:**
  Each user can access their profile, which displays personal info such as email, phone, role, building/apartment, and registration time.
- **Implementation:**
  - Profile data is fetched from Firestore.
  - The profile UI is interactive and organized.

3.5.5.9 Notifications and Alerts

- **Description:**
  Users receive notifications for important events (e.g., fire detected, device status changes).

- **Implementation:**
    - Notification service is initialized on app startup.
    - Uses local notifications for critical alerts.

3.5.5.10 Responsive and Modern UI/UX

- **Description:**
  The application features a modern, dark-themed interface with clear navigation and well-organized controls.
- **Implementation:**
    - Uses Google Fonts and custom theming.
    - UI adapts to different platforms (desktop, web, mobile).

3.5.6 User Interface

3.5.6.1 UI Technologies Used

- **Flutter Widgets:** For building a consistent UI across all platforms.
- **Bloc Pattern:** For state management, ensuring UI updates in real-time as data changes.
- **Google Fonts:** For improved readability and modern appearance.
- **Responsive Design:** Ensures usability on desktop, web, and mobile devices.

3.5.6.2 Sample UI Flow

- User opens the app and logs in.
- Dashboard appears, showing role-based controls and sensor data.
- User navigates to a room/kitchen to monitor data or control devices.
- User receives notifications for any critical events.
- Users can access profile or sign out at any time.

3.6 Web Application BackEnd

3.6.1 Purpose and Scope

The Neighborhood Forum is a Flask-based web application designed to serve residential communities by providing a centralized platform for neighbor communication,

Feature Category	Description	Key Models	Authentication Required

local marketplace functionality, and community service management. This application enables residents to engage in forum discussions, exchange private messages, post classified advertisements, and access public service information within their building or neighborhood complex.

This document provides a comprehensive overview of the system architecture, core features, and technology stack. For detailed API endpoint documentation, see API Endpoints. For specific implementation details of real-time features, see Real-time Communication. For database schema and model relationships, see Database Models.

3.6.2System Architecture

The application follows a modular Flask blueprint architecture with real-time communication capabilities and secure user management workflows.

Flask Application Structure

3.6.3 Core Features

The Neighborhood Forum provides the following primary features organized by user type and functionality:

Feature Category	Description	Key Models	Authentication Required
User Management	Registration, approval workflow, profile management	**User**	Varies
Forum Posts	Community discussion threads with real-time updates	**Post**	Yes
Private Messaging	Direct communication between users and admins	**Message**	Yes
Marketplace	Classified advertisements with image uploads	**Advertisement**	Yes
Public Services	Community service directory with contact information	**PublicService**, **PublicServiceCategory**	No (read-only)
Administration	User approval, content moderation, system management	All models	Admin only

3.6.4 Community-Specific Features

The application includes specialized features for residential communities:

- **Building/Apartment Tracking**: Users register with **building_number** and **apartment_number** fields for location identification

- **Admin Approval Workflow**: New users require admin approval (**is_approved** flag) before accessing community features
- **Content Moderation**: Soft deletion system with **is_deleted** and **deletion_type** fields for audit trails
- **Real-time Updates**: Socket.IO integration for live post and message notifications across all connected clients

3.6.5 Technology Stack

The application leverages a modern Python web development stack optimized for real-time community interaction:

3.6.5.1 Core Dependencies

Component	Version	Purpose
Flask	2.3.3	Web framework and API server
Flask-SocketIO	5.5.1	Real-time bidirectional communication
Flask-SQLAlchemy	3.1.1	ORM and database abstraction
Flask-JWT-Extended	4.5.3	JWT token authentication and authorization
Flask-Cors	4.0.0	Cross-origin resource sharing for frontend
bcrypt	4.0.1	Password hashing and verification
SQLAlchemy	2.0.40	Database toolkit and ORM engine

3.6.5.2 Supporting Libraries

Component	Version	Purpose
eventlet	0.35.1	Asynchronous networking for Socket.IO

Component	Version	Purpose
python-dotenv	1.0.0	Environment variable management
python-engineio	4.12.0	Engine.IO server implementation
python-socketio	5.13.0	Socket.IO server implementation

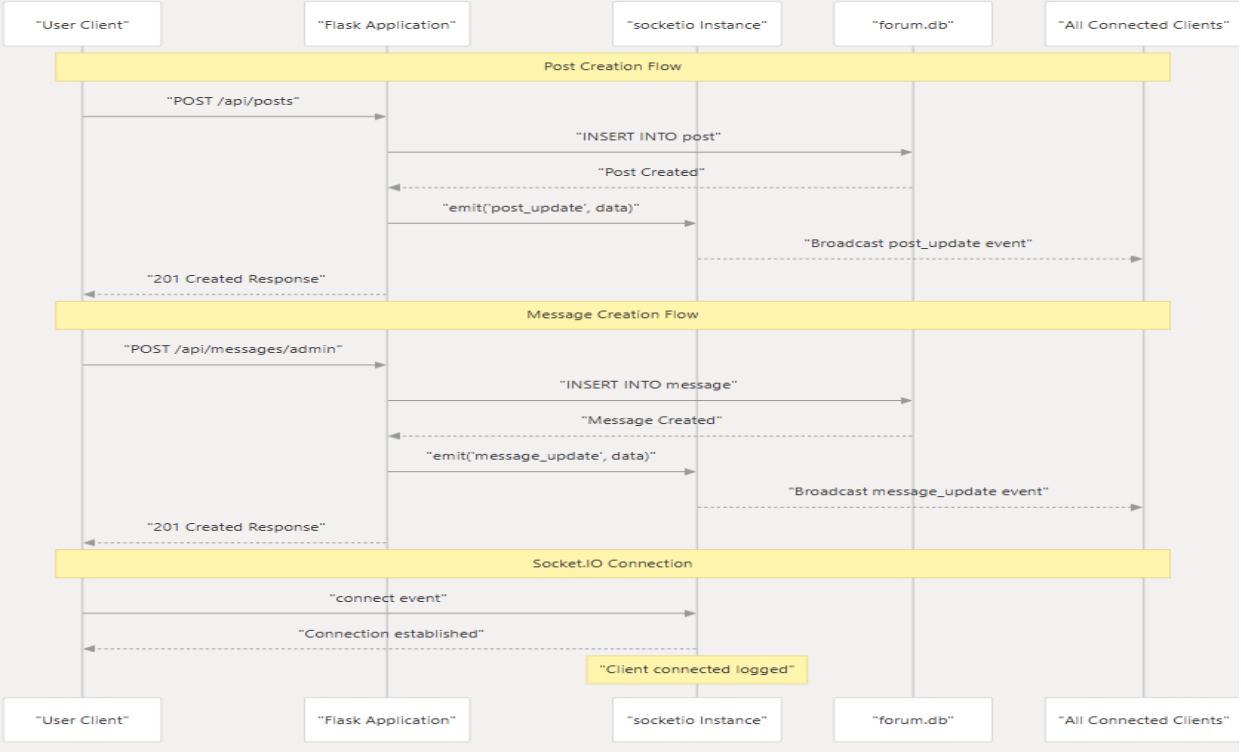
Configuration

The application uses environment-based configuration with the following key settings:

- **Database**: SQLite database stored at **sqlite:///forum.db**
- **JWT Secret**: Configurable via **JWT_SECRET_KEY** environment variable
- **CORS Origin**: Frontend allowed from **http://localhost:5173**
- **File Uploads**: Maximum 16MB files stored in **uploads/** directory
- **Token Expiry**: JWT tokens valid for 1 day

Real-time Communication Architecture

The application implements real-time features using Flask-SocketIO with event-driven updates for community engagement:



The real-time system enables immediate updates for community posts, private messages, and administrative actions across all connected neighborhood forum clients without requiring page refreshes.

3.6.6 Database Models

This document describes the SQLAlchemy database models that define the data schema for the Neighborhood Forum application. These models handle user management, community posts, private messaging, advertisements, and public services directory functionality.

For information about database initialization and setup, see Database Initialization. For API endpoint documentation that uses these models, see API Endpoints.

User Model

The **User** model implements a community-focused user management system with admin approval workflow and role-based permissions.

Field	Type	Constraints	Purpose
id	Integer	Primary Key	Unique identifier
username	String(50)	Unique, Not Null	Login identifier
password	String(100)	Not Null	Bcrypt hashed password
full_name	String(100)	Not Null	Display name
building_number	String(20)	Not Null	Physical location identifier
apartment_number	String(20)	Not Null	Physical location identifier
is_admin	Boolean	Default: False	Administrative privileges

Field	Type	Constraints	Purpose
is_approved	Boolean	Default: False	Admin approval status
is_banned	Boolean	Default: False	Ban status
created_at	DateTime	Default: utcnow	Registration timestamp

Key Features

- **Approval Workflow**: New users require admin approval (**is_approved=False** by default)
- **Location Tracking**: Building and apartment numbers for community verification
- **Role Management**: Admin privileges controlled via **is_admin** flag
- **Cascade Deletion**: All user-created content is deleted when user is deleted

Post Model

The **Post** model handles community forum posts with soft deletion capabilities for content moderation.

Field	Type	Constraints	Purpose
id	Integer	Primary Key	Unique identifier
content	Text	Not Null	Post content
user_id	Integer	Foreign Key to User	Post author
created_at	DateTime	Default: utcnow	Creation timestamp
is_deleted	Boolean	Default: False	Soft deletion flag
deletion_type	String(20)	Nullable	Deletion source: 'user' or 'admin'

Soft Deletion Pattern

Posts use a soft deletion pattern where content is marked as deleted rather than physically removed, preserving audit trails and allowing content recovery.

Message Model

The **Message** model implements private messaging between users and administrators with read status tracking.

Field	Type	Constraints	Purpose
id	Integer	Primary Key	Unique identifier
content	Text	Not Null	Message content
sender_id	Integer	Foreign Key to User	Message sender
recipient_id	Integer	Foreign Key to User	Message recipient
created_at	DateTime	Default: utcnow	Send timestamp
is_read	Boolean	Default: False	Read status
is_deleted	Boolean	Default: False	Soft deletion flag
deletion_type	String(20)	Nullable	Deletion source: 'user' or 'admin'

Relationship Configuration

The model uses dual foreign keys to the **User** model with explicit **foreign_keys** specification to handle the sender/recipient relationships properly.

Advertisement Model

The **Advertisement** model handles marketplace functionality with image upload support and pricing information.

Field	Type	Constraints	Purpose

Field	Type	Constraints	Purpose
id	Integer	Primary Key	Unique identifier
title	String(100)	Not Null	Advertisement title
content	Text	Not Null	Description
user_id	Integer	Foreign Key to User	Advertisement creator
created_at	DateTime	Default: utcnow	Creation timestamp
is_deleted	Boolean	Default: False	Soft deletion flag
price	Float	Nullable	Item price
phone_number	Text	Nullable	Contact information
images	Text	Nullable	JSON string of image URLs

Image Storage

The **images** field stores image file paths as a JSON string array, with actual files stored in the **uploads/** directory managed by the file upload utilities.

PublicServiceCategory Model

Categories for organizing community services with audit timestamps.

Field	Type	Constraints	Purpose
id	Integer	Primary Key	Unique identifier
name	String(100)	Not Null	Category name

Field	Type	Constraints	Purpose
description	Text	Not Null	Category description
created_at	DateTime	Default: utcnow	Creation timestamp
updated_at	DateTime	Auto-update on modify	Last modification timestamp

PublicService Model

Directory of community services with availability status tracking.

Field	Type	Constraints	Purpose
id	Integer	Primary Key	Unique identifier
name	String(100)	Not Null	Service provider name
category	Integer	Foreign Key to Category	Service category
phone_number	String(20)	Not Null	Contact number
status	String(50)	Not Null	Availability status ("Active", "Unavailable")
created_at	DateTime	Default: utcnow	Creation timestamp
updated_at	DateTime	Auto-update on modify	Last modification timestamp

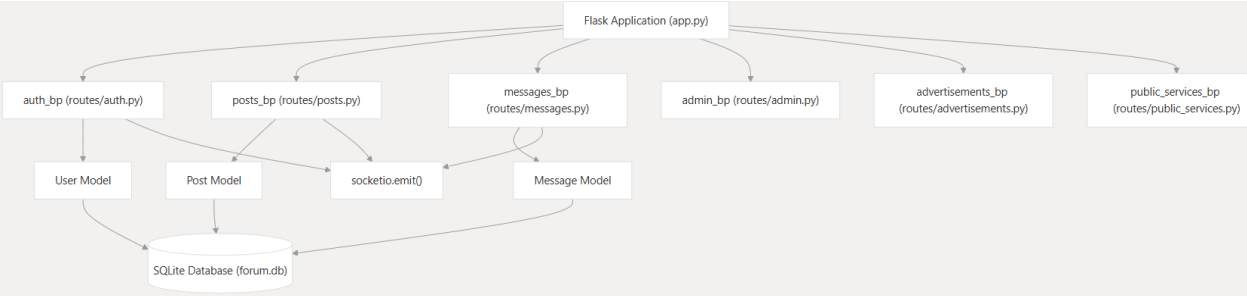
3.6.7 API Endpoints

This document provides a comprehensive reference for all REST API endpoints in the Neighborhood Forum application. It covers the overall API architecture, authentication patterns, and organizational structure of endpoints across functional areas. For detailed endpoint specifications within each functional area, see the respective sub-pages: Authentication API, Posts API, Messages API, Advertisements API, Public Services API, and Admin API.

API Architecture Overview

The Neighborhood Forum API is built using Flask with a modular Blueprint architecture. Each functional area is organized into separate route modules that handle specific domain concerns while sharing common authentication and real-time communication patterns.

Blueprint Organization



3.6.8Authentication and Authorization

JWT Token Structure

The API uses **flask-jwt-extended** for authentication with tokens containing user identity and role claims:

Token Component	Description
identity	User ID as string
is_admin Token extraction	Admin privilege claim **get_jwt_identity()** returns string

Token Component	Description
Claims extraction	**get_jwt()** returns claims dict

Endpoint Organization

Authentication Endpoints

Method	Route	Function	Purpose
POST	**/auth/register**	**register()**	**User registration with admin approval**
POST	**/auth/login**	**login()**	**User authentication and token generation**
GET	**/auth/profile**	**get_profile()**	**User profile retrieval**

Posts Endpoints

Method	Route	Function	Purpose
GET	**/posts**	**get_posts()**	Retrieve all forum posts
POST	**/posts**	**create_post()**	Create new forum post
DELETE	**/posts/<int:post_id>**	**delete_post()**	Soft delete post

Messages Endpoints

Method	Route	Function	Purpose
POST	**/messages/admin**	**message_admin()**	Send message to admin
GET	**/messages**	**get_messages()**	Get user's message history
POST	**/messages/<int:message_id>/read**	**mark_as_read()**	Mark message as read
POST	**/messages/reply/<int:user_id>**	**reply_to_user()**	Admin reply to user
DELETE	**/messages/<int:sender_id>/<int:recipient_id>/<int:message_id>**	**delete_message()**	Soft delete message

Real-time Integration

Socket.IO Event Emissions

All content creation and modification endpoints emit real-time updates via **socketio.emit()**

Event Data Structure

Event Type	Event Name	Data Structure
Post events	**post_update**	**{id, content, created_at, author: {id, username}}**
Message events	**message_update**	**{id, content, created_at, sender: {}, recipient: {}}**
User events	**user_registered**	**{id, username, full_name, building_number, apartment_number}**

Error Handling Patterns

Standard Error Responses

The API follows consistent error response patterns:

HTTP Status	Error Type	Response Format
400	Bad Request	**{"error": "Missing required field: {field}"}**
401	Unauthorized	**{"error": "Invalid username or password"}**
403	Forbidden	**{"error": "You are banned and cannot create posts"}**
404	Not Found	**{"error": "User not found"}**
500	Server Error	**{"error": "Failed to process request: {details}"}**

API Integration Points

Database Models

Each endpoint integrates with specific SQLAlchemy models:

Blueprint	Primary Models	Key Relationships
auth_bp	**User**	N/A
posts_bp	**Post**, **User**	**Post.user_id** → **User.id**
messages_bp	**Message**, **User**	**Message.sender_id** → **User.id**, **Message.recipient_id** → **User.id**

3.6.9 Cross-Cutting Concerns

All endpoints share common infrastructure components:

- **JWT Authentication**: **flask_jwt_extended** for token validation
- **Password Security**: **bcrypt** for password hashing
- **Real-time Updates**: **socket_instance** for WebSocket communications
- **Database Access**: **SQLAlchemy** ORM with automatic session management

3.7 Compound Website “MOSTAQBAL City” Frontend

3.7.1 Purpose and Scope

This document provides a high-level technical overview of the MOSTAQBAL City platform, a React-based web application for smart community management. The platform serves as a digital hub for residents and administrators of the MOSTAQBAL City residential compound, offering smart home integration, security management, and community communication features.

3.7.2 Target Audience and Market

The system is designed for two primary user roles

1. Compound Residents: Can communicate with other residents and the owner, view services, and access announcements.

2. Compound Owner/Admin: Has access to all chats, can manage the service center, post advertisements, and communicate with residents privately.

3.7.3 Overview of Smart Home Features and Importance

Smart home features include centralized mobile control, voice-activated commands, energy-efficient climate management, and advanced security systems. These technologies enhance residential safety, reduce energy consumption, and contribute to a more comfortable and secure living environment.

3.7.4 Planning and Design:

The frontend design was carefully planned to ensure a smooth, responsive, and intuitive user experience for both residents and the compound owner. Key areas of focus included layout, component structure, user flow, and responsiveness.

-Technology Stack:
The frontend was built using React with Vite for faster bundling and development performance. React Router was used for page navigation, and Mantine UI was chosen for its modern, customizable component library.

-Component-Based Architecture:
The interface was structured into reusable and modular components, such as Header, Chat Box, Login Form, and Register Form, which improved code readability and maintainability.

-Responsive Design:
The layout was designed to work seamlessly across all screen sizes. Using responsive components, the UI adjusts dynamically for desktops, tablets, and mobile devices.

-User Role-Based Interfaces:
Separate frontend views were designed based on user roles:

1. Residents: Have access to the home page, group chat, private chat with the owner, service center (view-only), and advertisements.

2. Compound Owner: Has additional access to edit the service center, post advertisements, privately message any resident, manage approvals, and manage users.

-Routing and Navigation:
The site uses React Router to create protected routes for different user types, ensuring that unauthorized access to pages is restricted.

3.7.5 Front-end Technologies and Frameworks:

1-ReactJS “Frontend framework”

React.js is a powerful open-source JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications (SPAs). It enables developers to create reusable components that manage their state, resulting in faster and more maintainable code.

React utilizes a virtual DOM to minimize performance costs during updates, ensuring that UI changes are efficient and smooth. Its declarative syntax allows developers to describe how the UI should appear based on the application state, making the code easier to understand and debug.

2-Mantine UI

Mantine UI is a modern React component library that provides a rich set of accessible, customizable, and performant UI components and hooks for building web applications quickly and efficiently.

3-React Router Dom

React Router Dom is the standard routing library for React web applications. It allows you to build single-page applications (SPAs) with client-side navigation, meaning users can move between pages without triggering full page reloads.

4-Socket.IO Client

socket.io-client is the client-side library, which enables real-time, bidirectional communication between web clients (like browsers) and servers using Web Sockets (or fallbacks like polling when necessary).

It’s used in applications where live updates are critical, like chat apps, multiplayer games, dashboards, and notifications.

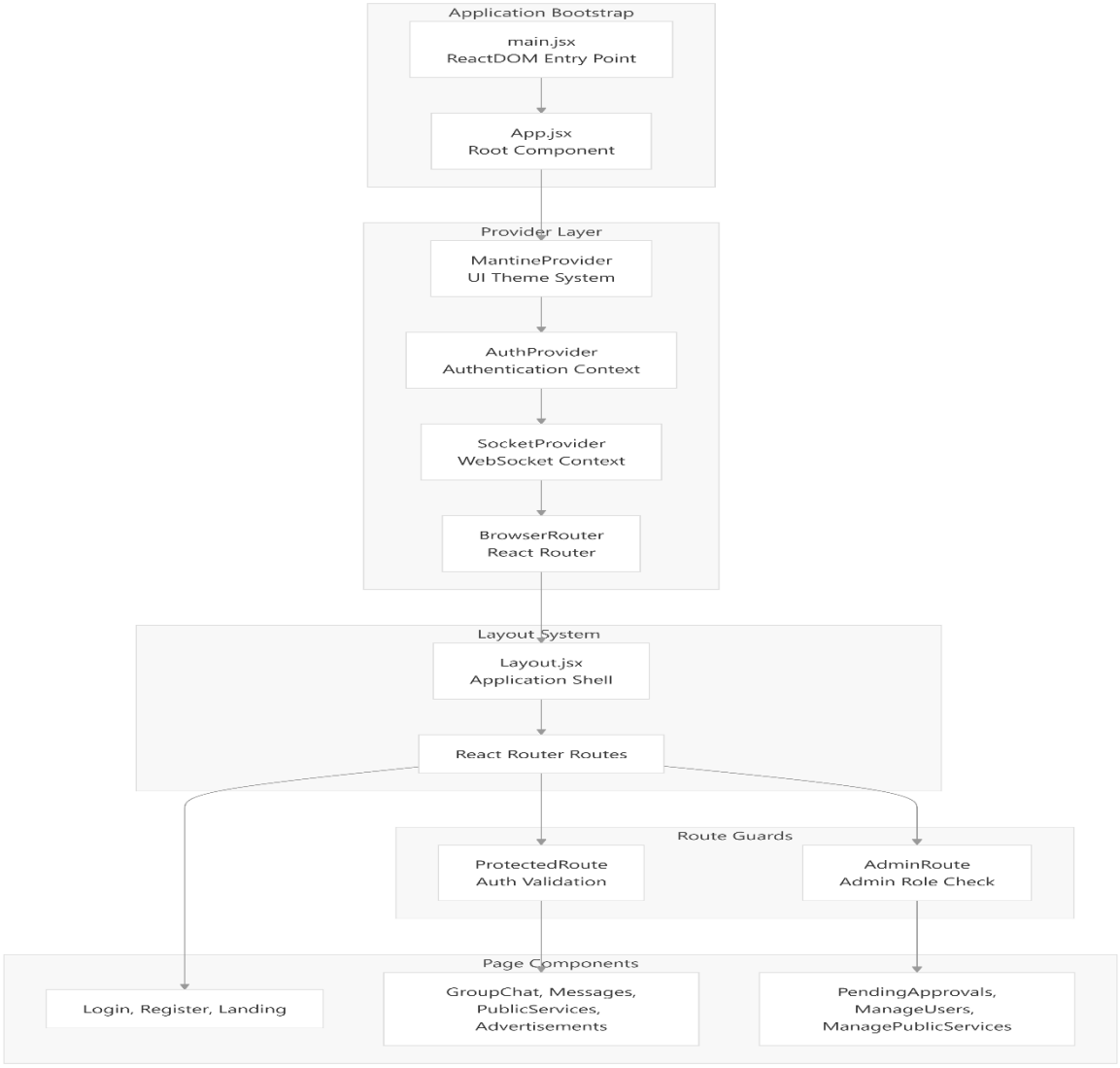
5-Axios

Axios is a JavaScript library used to make HTTP requests from the browser to a web server. It’s commonly used in frontend frameworks like React, Vue, or Angular to communicate with a backend API.

6- Tabler Icons React

Tabler Icons React is a library that provides free, open-source SVG icons as React components, making it very easy to use icons in a React project.

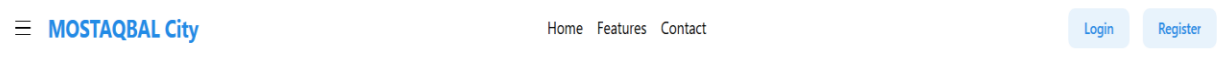
3.7.6 System Structure:



3.7.7 Key Components and features:

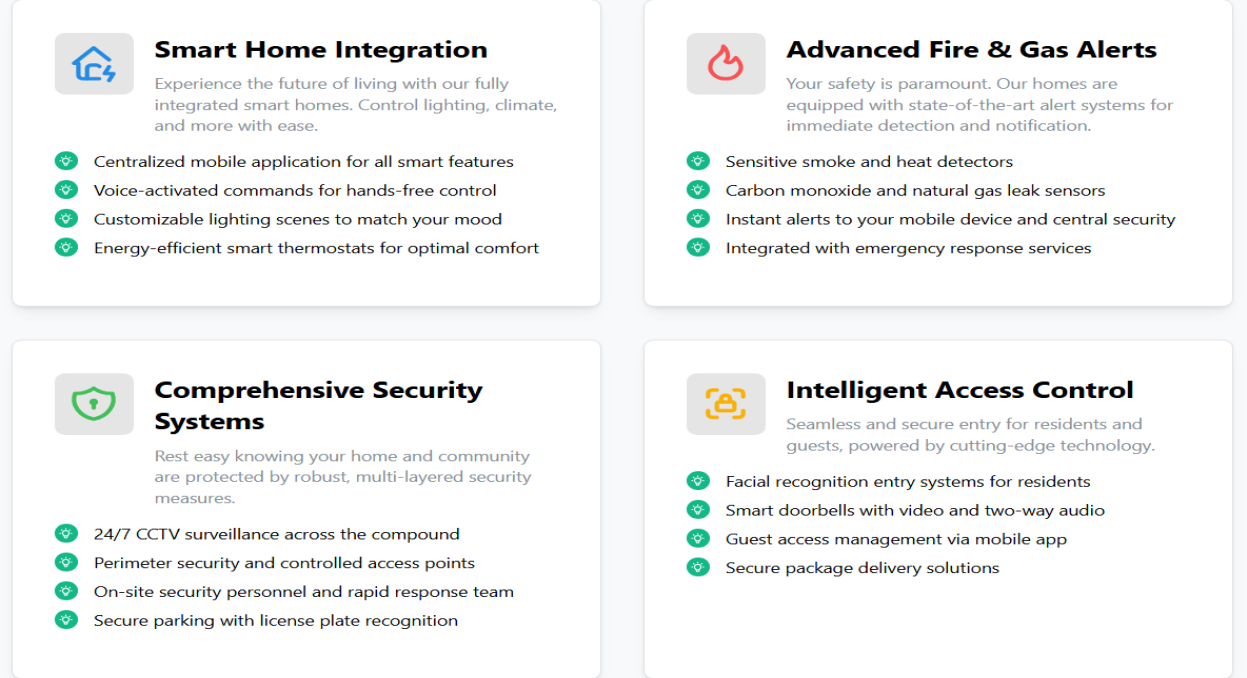
Navigation Bar and Site Structure:

A clear and intuitive navigation bar is crucial. It should include links to the home page, compound features, and contact.



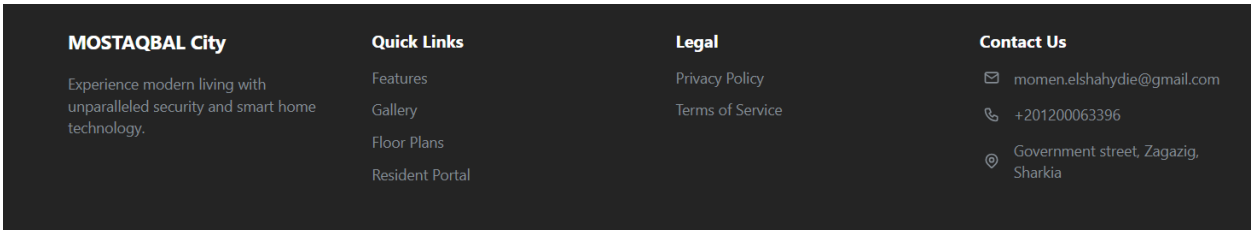
Home Page Design:

The home page should highlight the features of smart homes within the compound.



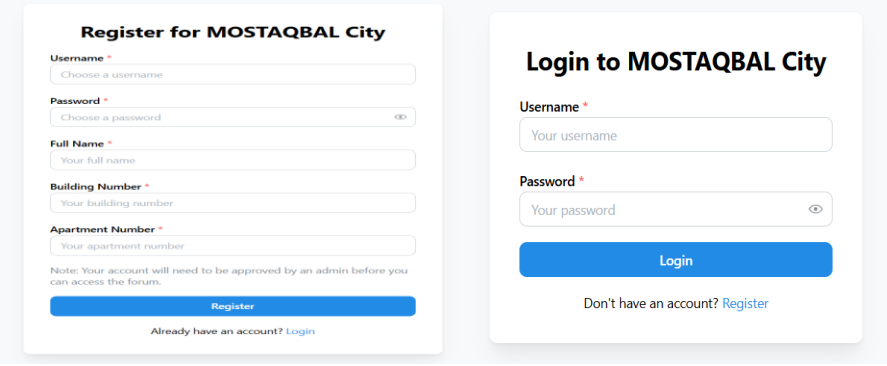
Contact:

Contact should highlight the email, phone number, and address of the compound.



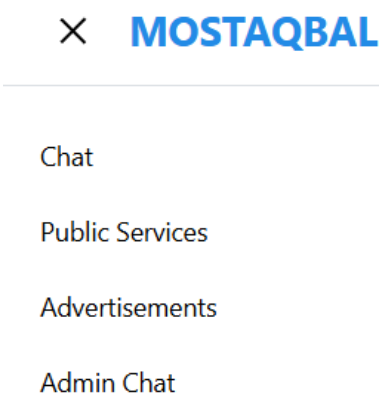
User Authentication:

Implement secure login and registration forms.



Sidebar:

The sidebar contains shortcuts to all pages of the site: “Group Chat – Admin Chat – Public services – Advertisements”

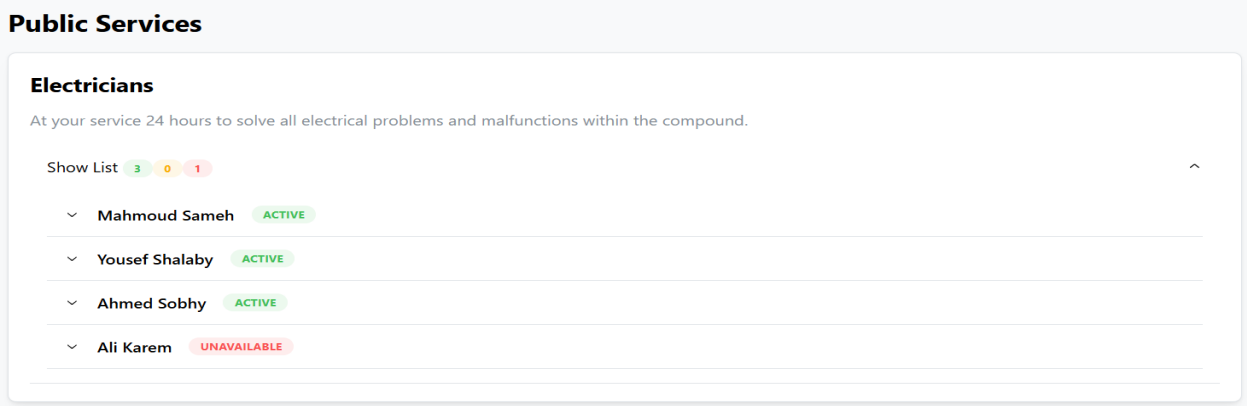


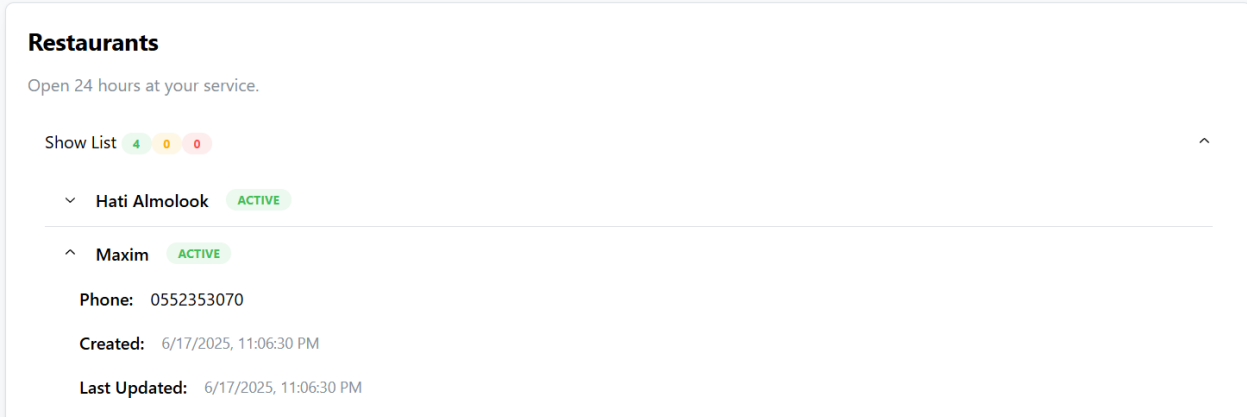
Group Chat:

Chat between all compound residents for easy communication without sharing data.

Public Services:

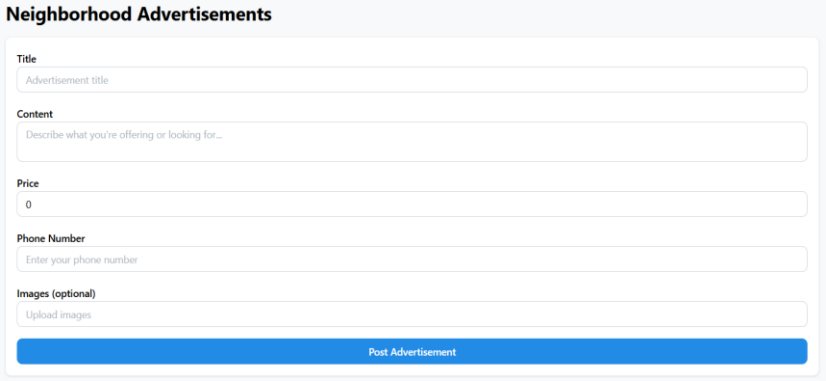
Public services contains a directory with the numbers of all craftsmen, such as electricians, carpenters, plumbers, and the numbers of restaurants and cafes within the compound.

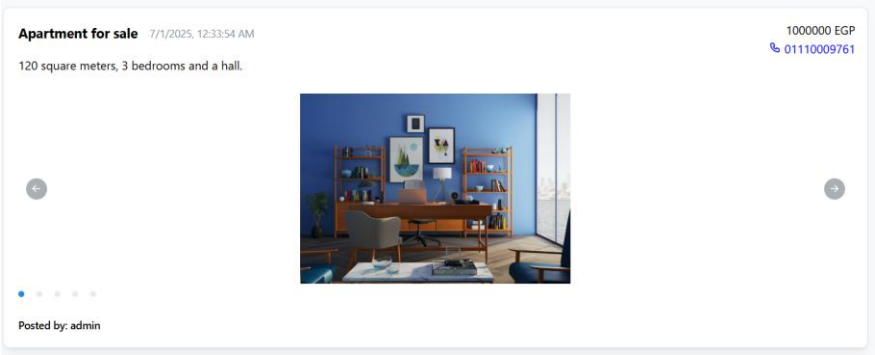




Advertisements:

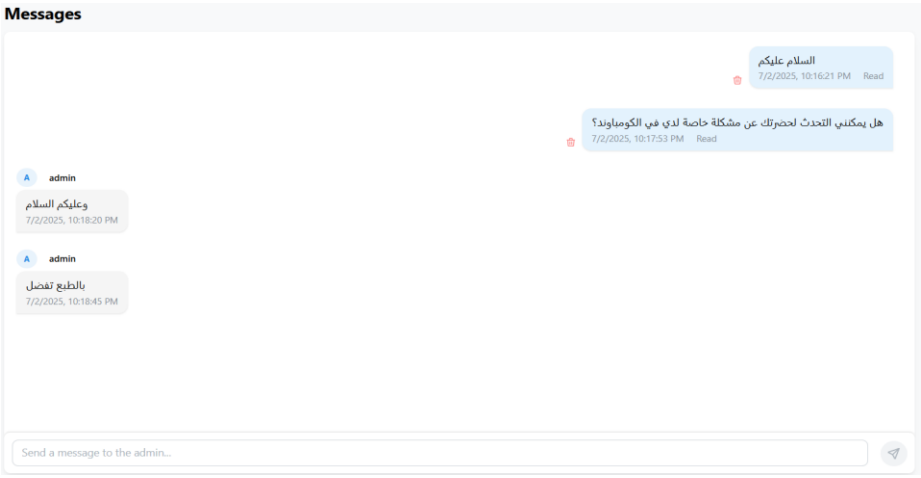
Any resident can post an advertisement to sell anything, such as an apartment for sale, a car for sale, a wheel, etc., with the inclusion of photos in the advertisement.





Admin Chat:

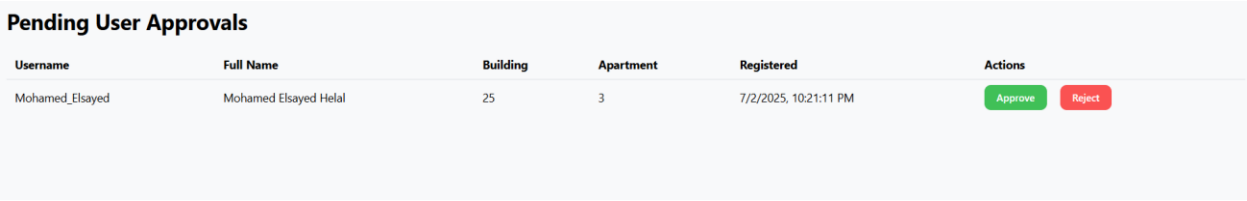
Chat with the compound owner to discuss a private problem, for example.



<u>Pages for admins only</u>

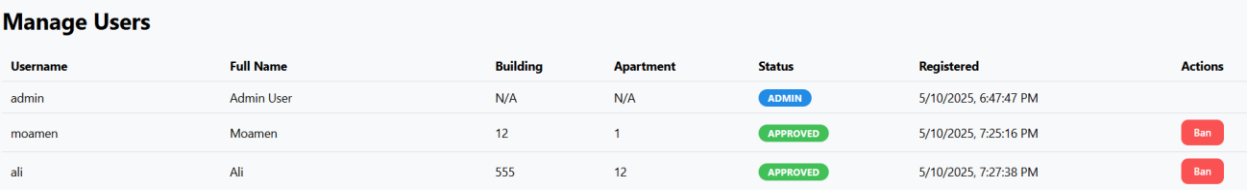
1-Manage Approvals:

A page through which the admin can approve or reject a user who has registered on the site.



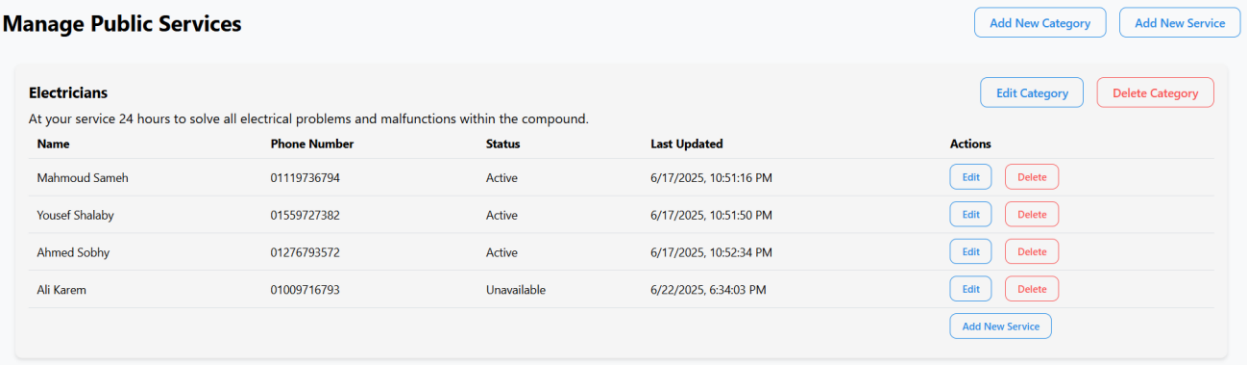
2-Manage Users:

A page that displays all users on the site, and the admin can ban any of them.



3-Manage Public Services:

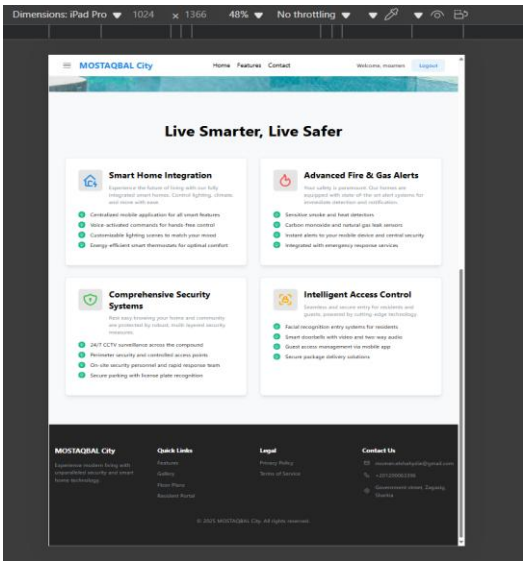
A page through which the admin can edit the Service Center page. He can edit, delete, or add anything.



3.7.8 Responsive Design:

Importance of Responsive Design: Responsive design ensures the site provides a good user experience across all devices, which is crucial given the diverse ways users access websites.

→iPad pro pov:



Adaptive vs. Responsive Design: Adaptive design uses different layouts for different devices, while responsive design adjusts a single layout based on screen size. Combining both approaches can provide the best user experience.

Testing on Various Devices: Test the site on multiple devices and browsers to ensure compatibility and performance. Tools like Device tool bar on browser like chrome or brave or real device testing are essential.

3.7.9 Challenges encountered during the implementation of this compound platform:

-Responsive Design: Ensuring that the website works seamlessly across various devices (desktops, tablets, and mobile phones).

-Performance Optimization: Balancing high-quality images to provide a smooth user experience.

- Real-time Communication: Implementing efficient chat systems (group and private) using Socket.IO.

Chapter 4: Testing and fixing errors

The development of our Smart Home System was a meticulous journey that required rigorous testing at every stage to ensure reliability, functionality, and user satisfaction. This chapter documents the step-by-step testing process undertaken throughout the project, from the initial development of individual components to the integration of all subsystems into a cohesive system.

Our testing methodology was designed to validate each module—Smart Security System, Fire Detection, Face Recognition, Light and Comfort System, Mobile Application, and Compound Website—independently before combining them into the final system.

We encountered numerous challenges during this process, including hardware malfunctions, software bugs, integration conflicts, and performance issues, particularly given our constraints as a team of learners with limited resources. For each issue, we systematically identified the root cause, implemented solutions, and verified their effectiveness through iterative testing.

This chapter details the testing strategies employed, the specific errors faced during development and integration, and the solutions we devised to overcome them, ensuring the system met our objectives of security, convenience, and energy efficiency.

By documenting our testing journey, we aim to provide a clear understanding of how we transformed individual components into a robust, fully integrated smart home solution.

4.1 Smart Security System Testing

To ensure the functionality and reliability of the Smart Security System, extensive testing was conducted through multiple well-structured phases. These phases included individual component testing, subsystem integration, full system assembly, and advanced feature verification. Below is a detailed breakdown of the testing process:

4.1.1. Individual Component Testing:

Initially, each component was tested independently. This included the keypad, LCD screen, fingerprint sensor, output circuit (buzzer and LED), and the RTC (Real-Time Clock) module. Each module functioned correctly when operated as a standalone system.

Component	Test Result
Keypad	Responded correctly to button inputs
LCD Display	Displayed characters and messages accurately
Fingerprint Sensor	Successfully recognized and validated fingerprints
Output Circuit (Buzzer + LED)	Responded correctly to signal triggers
RTC Module (Real-Time Clock)	Maintained accurate time and date readings

4.1.2. Subsystem Integration:

After successful individual testing, we integrated components in pairs:

- The keypad was connected to the LCD screen and tested together.

- The fingerprint sensor was connected to the LCD screen and tested similarly.
- Output components (LED, buzzer) were added to both configurations and verified.
- A relay module was added and successfully tested.
- The solenoid lock was then connected via the relay and tested. All of this required multiple code modifications and logic handling to ensure proper output behavior.

Subsystem Configuration	Outcome
Keypad + LCD	Button presses are correctly displayed on the screen
Fingerprint Sensor + LCD	Recognition status is shown accurately
Keypad/Fingerprint + Output Circuit	Buzzer and LED responded as expected
Relay Module Added	Relay triggered correctly on valid input
Solenoid Lock via Relay	Lock opened and closed as required

4.1.3. Full System Integration:

The next phase involved combining all components:

- Fingerprint module, keypad, LCD screen, output circuit, relay, and solenoid.

This phase required careful integration of all code segments, which was time-consuming due to the complexity of handling multiple modules simultaneously. After persistent debugging and adjustments, the full system operated successfully.

Integration Aspect	Result	Notes
Hardware Integration	☑ Successful	All components connected in a single circuit
Software Integration	☑ Complete	Required merging multiple code segments
Debugging & Optimization	☑ Done	Multiple bugs resolved after intensive testing

4.1.4. RTC Module Integration:

Once the main functionality was verified, the RTC module was connected, and its code was integrated into the system. It was configured to log the date and time of each entry into the house. This part introduced significant challenges, particularly with time synchronization and data recording, but was eventually resolved.

Feature Tested	Challenge Faced	Outcome
Time Synchronization	Time conflicts and data overlap	Resolved ☑
Logging Entry Events	Format and storage optimization	Working ☑

4.1.5. Camera and Raspberry Pi Testing:

We tested the integration of the Raspberry Pi and a camera module to enable face-based entry by unlocking the solenoid. This feature worked as expected but was not finalized in the current prototype due to time constraints.

Component	Purpose	Test Result	Status
Raspberry Pi	Image processing and logic	Successfully tested	☑ Working prototype
Camera Module	Capture and recognize faces	Functional	☑ Functional
Solenoid Unlock via Face Recognition	Open lock after valid match	☑ Succeeded	Not finalized due to time

4.1.6. Final Assembly:

After confirming system stability, all wires were soldered to ensure permanent connections. This step improved the physical durability and operational stability of the circuit, minimizing the risk of faulty connections.

Task	Purpose	Result
Soldering Connections	Improve electrical stability	☑ Successful
Wire Management	Reduce loose or faulty connections	☑ Improved
Final Circuit Housing	Protect components	☑ Durable setup

4.2 Testing Fire Detection Module – Development and Testing Stages

Below is an outline of the main stages we followed and what tests we performed at each stage.

4.2.1 Dataset Preparation

Activities:
-Collected fire image datasets from Roboflow.
-Reviewed image quality and annotations.
-Applied augmentations (e.g., flipping, brightness changes).

Testing:
-Visual Validation: Manually inspected dataset samples to confirm correct labels and bounding boxes.
-Dataset Split Check: Verified the distribution of training, validation, and test sets to avoid data leakage.

4.2.2 Model Training and Fine-Tuning

Activities:
-Initialized YOLOv8 Nano pre-trained weights.
-Fine-tuned on the prepared dataset.
-Tracked training metrics (loss, mAP, precision, recall).

Testing:
-Training Metrics Review: Monitored loss curves to detect overfitting.
-Validation Evaluation: Measured mAP and class-wise precision on the validation set.
-Sample Inference Test: Ran predictions on unseen test images to confirm detection accuracy.

4.2.3 Model Export and Conversion

Activities:
-Saved trained weights (best.pt).
-Verified compatibility with Ultralytics YOLO Python API.

Testing:
-Load Test: Loaded the best.pt file in a test script to confirm no corruption.
-Inference Sanity Check: Passed sample images to the model and checked that detections were consistent with validation results.

4.2.4 Raspberry Pi Environment Setup

Activities:
-Installed dependencies (Ultralytics, OpenCV, Pygame, Flask, Firebase Admin).
-Configured the Pi Camera.

Testing:
-Dependency Verification: Ran import statements to confirm all libraries were installed.
-Camera Check: Captured and displayed test frames using Picamera2.

4.2.5 Inference Pipeline Implementation

Activities:
-Integrated into YOLO inference code with video capture.
-Added logic for drawing bounding boxes.

Testing:

-Single Frame Inference: Ran the model on a test frame to verify detection outputs.
-Threshold Filtering: Tested detection threshold logic (e.g., ensuring no boxes below 65% confidence).
-Performance Profiling: Measured inference time per frame.

4.2.6 Alerting Mechanism Integration

Activities:
-Connected Firebase Realtime Database.
-Added logic to update fire sensor flags.
-Configured Pygame to play audio alerts.

Testing:
-Firebase Connectivity Test: Wrote test data to the database to verify connection.
-State Transition Test: Simulated detection and clearing of fire to confirm correct status updates.
-Audio Playback Test: Played the alert sound to check volume and clarity.

4.2.7 Video Streaming via Flask

Activities:
-Created a Flask server endpoint (/fire) to stream annotated frames.

Testing:
-Stream Test: Opened the stream URL in a browser to verify frame delivery.
-Latency Measurement: Evaluated stream lag between capture and display.
-Resource Monitoring: Checked CPU usage while streaming.

4.2.8 Full System End-to-End Testing

Activities:
-Deployed the complete pipeline on a Raspberry Pi.
-Ran continuous video capture and detection.

Testing:
Scenario Simulation:
-Showed fire images to trigger detection.
-Removed fire to confirm reset of alerts.
Integration Validation:
-Confirmed Firebase updates matched detection status.
-Verified audio and stream response.
Robustness Test:
-Ran the system for extended periods (1–2 hours) to check stability and memory usage.

4.3 Face Recognition – Gas Detection

4.3.1 Face Recognition – Testing and Results

Purpose of Testing:

The goal of testing is to ensure that the Face Recognition Module meets performance expectations in real-time usage scenarios, including identification accuracy, response time, and system stability under different environmental conditions.

Testing Environment:

Component	Details
Processor	Raspberry Pi 5 (quad-core ARM Cortex-A76)
Camera Module	PiCamera2
Face Recognition Lib	face_recognition (using HOG backend)
Supporting Libraries	OpenCV, Flask, Firebase Admin SDK, Arduino Serial
Display Interface	Web browser (Flask stream)
Dataset	5 known individuals × 20 images (train + test)
Control Hardware	Arduino Uno + Servo Motor (lock mechanism)

Test Scenarios:

The table below summarizes the practical testing cases:

Scenario	Description	Expected Outcome
Known Person (Frontal)	The user looks directly at the camera	Access granted, name displayed
Known Person (With Mask)	Known user wearing a medical mask	Partial recognition or unknown
Unknown Person	An unregistered person in front of the camera	Rejected, no door access
Multiple Faces	2–3 people visible at once	Only known faces were recognized
Lighting Variation	Face under poor light or	Lower confidence, but still

(Dim/Bright)	glare	functions
Face at Angle (Side Profile)	Face detected from a ~45° angle	Medium confidence recognition
No Face in Frame	Empty room or motion blur	No response, no false detection

Evaluation Metrics:

Metric	Value
Recognition Accuracy	91.8% (validated on 100 test samples)
False Positive Rate (FAR)	2.3%
False Negative Rate (FRR)	5.9%
Average Inference Time	170–190ms
Door Response Time	< 1.2 seconds
Frame Rate (Streaming)	13–17 FPS

Methodology:

Testing was conducted using real-time evaluation of 100 face samples in various environmental setups. Each input was passed through the full recognition pipeline and verified by observing the system's response (access granted, logging behavior, and hardware activation).

Visual Proof:

- [✓] Screenshot of multiple users labeled on video stream
- [✓] Sample Firebase log entry with timestamp
- [✓] Arduino serial output showing lock control events

Critical Observations:

- Raspberry Pi 5 significantly improved performance compared to Raspberry Pi 4.
- Recognition was most accurate with frontal faces.
- Long-term stability was observed with over 3 continuous hours of operation.
- Real-time updates to Firebase and Arduino were reliable and consistent.

Limitations:

- Recognition may degrade with more than 5 faces in frame.
- Low-light conditions impact detection accuracy.
- No current spoofing protection (e.g., anti-photo detection).

Future Testing Recommendations:

- Conduct tests in high-traffic environments.
- Integrate IP cameras for broader area coverage.
- Include mobile push notifications (via FCM) testing.
- Explore facial spoofing prevention mechanisms.

4.3.2 Gas Detection – Testing and Results

Purpose of Testing:

To verify that the gas sensor detects gas leaks accurately and reliably, and that the connected speaker successfully emits the warning message:
“Warning! There is a gas leak.”

Hardware & Environment Setup:

Component	Description
Gas Sensor Module	MQ-5 Sensor (for LPG, propane, methane detection)
Microcontroller	Raspberry Pi 5
Output Device	Speaker connected to 3.5mm audio jack
Power Source	USB-C 5V power bank
Software Layer	Python script with pygame for audio playback
Test Environment	Indoor room (closed windows, no fan/AC)

Testing Procedure:

- Calibration: The sensor is allowed to warm up for 20 seconds.
- Baseline Reading: Normal air conditioning is recorded as reference.
- Gas Release Simulation: A Small amount of gas (lighter butane) is released near the sensor.
- Sensor Trigger: When gas concentration crosses the threshold (digital pin reads HIGH), audio playback is initiated.
- Reset Behavior: System tested again after ventilation to ensure sensor returns to normal state.

Test Scenarios:

Scenario	Description	Expected Outcome
No Gas Present	Normal room air	No sound is played
Light Exposure (butane)	Sensor exposed to small gas leak for 3 seconds	Speaker plays warning sound
High Gas Concentration	Larger release near the sensor	Immediate response + repeated sound

False Alarm Check	Test in the kitchen without actual gas	No false detection
Sensor Reset After Detection	Ventilate the room after detection	Sensor resets, system returns to standby

Metrics & Results:

Metric	Observed Value
Average Detection Time	2.1 seconds (from gas release)
Audio Alert Delay	< 1 second
False Positive Rate	0% (over 5 test runs)
Recovery Time (to normal state)	~10 seconds post-ventilation
Speaker Volume (Distance 1m)	~80 dB (clearly audible)

Observations:

- The sensor was highly responsive and consistent during repeated tests.
- Audio feedback was clear and triggered instantly.
- No false positives were recorded under normal conditions.
- System resets smoothly after gas is cleared.
- Response time is acceptable for real-world use in home environments.

4.4 –Light and Comfort system testing

To validate the functionality and efficiency of the Smart Light and Temperature System, testing was conducted in multiple stages:

4.4.1 - Individual Component Testing :

- PIR sensor: Detected motion accurately and triggered test signals .
- Light sensors (LDR): Accurately detected ambient light levels and responded to changes.
- Infrared Sensor : Accurately detected obstacles
- DHT sensor : Provided consistent temperature and humidity readings.
- Firebase realtime : Successfully connected to Wi-Fi and communicated with Firebase Realtime Database.
- Fan for Ventilation : we could turn it on and off by the microcontroller

4.4.2- Subsystem Integration:

1.Using two ir to achieve the idea of entry counter :

✗When one person entered it counted it as more than one

reason : it counts just one person as many people because he spend about 400ms to pass in front of the ir so it was giving signal all this time

solution : Adding debounce delay

✓ it worked well as it was excpected .

2.Integration between the entry counter and pir :

✓ they gave the excepected results as the entry counter was increase if someone entered and decrease if someone exited and after that the pir checked the motion to check if the person in the room is doing something or he slept to turn the lights off.

3.Try to control the fan depending on DHT readings:

✓ The fan was turned on and off depending on the temperature from the sensor

4.4.3 - All System before connecting to firebase

We integrated all the hardware components the entry counters for two rooms ,pir for the two rooms ,LDR and it's outing LED and the DHT11 sensor with the fan .

✓ All systems worked as it was excpected which were : the rooms lighting , outing lights and the Ventilation system

4.4.4 – All system Integration with firebase

✗The result was right and as it was excpected until the wifi speed got slower which made the sending and receiving slow so the code was stuck at the line of waiting the acknowledgment which prevent the compiler to go to the next lines which read the sensors output .

Solution :

we decided to optimize number of bits we send and receive by :

1. Convert data types of the manipulated variables to smaller data types
2. Ignoring sending unnecessary varibales that made the sending take long time if the wifi is slow

3. Adding check line before sending each variable to make sure that it has been changed to avoid spending time in sending a value that was sent before so we reduced number of sending data via wifi
4. Adding small delay between sending times to decrease the probability of errors

✓ It worked well with no errors in both Manual and Automatic modes

Final testing included simulations of different scenarios (e.g., no motion, dark room, high temperature) to ensure all features worked as intended. After multiple trials and minor code refinements, the system achieved stable and reliable operation

Final testing included simulations of different scenarios (e.g., no motion, dark room, high temperature) to ensure all features worked as intended. After multiple trials and minor code refinements, the system achieved stable and reliable operation.

4.5. Mobile app testing:

4.5.1 Testing & Deployment:

After completing the core development of the Smart Home application, we proceeded with rigorous manual testing to ensure stability and reliability across various features. The application was tested on multiple Android devices to verify its behavior under real-world conditions. We ensured all components—including user authentication, room control, sensor monitoring, and notifications—performed as expected without crashes or unexpected behavior.

To streamline delivery and testing, we implemented Continuous Integration and Continuous Deployment (CI/CD) using Firebase App Distribution. This allowed us to automatically build and distribute release versions of the application to testers and supervisors with minimal manual effort. Every time we push a new commit to the main branch, a new APK is generated and distributed via Firebase, ensuring rapid and consistent delivery.

This setup reduced the manual workload, improved development speed, and helped identify bugs earlier during the release cycle.

In the future, we aim to enhance the project by adding automated testing (unit, widget, and integration tests) using the Flutter testing framework, which will further improve

code quality and ensure long-term maintainability.

Feature	Description	Expected Result	Status
User Registration	Verify users can sign up with valid inputs	Account created and verified	✅
Login Authentication	Check login with correct and incorrect credentials	Success on valid, error on invalid	✅
Sensor Monitoring	Simulate flame/gas detection and check real-time updates	UI updates with alert state	✅
Notifications	Ensure notification is triggered when danger is detected	Local push notification appears	✅
Role-Based Navigation	Test that each role is redirected to the correct home page	Father, Mother, Child screens shown	✅

4.5.2 CI/CD using firebase:

We implemented a Firebase-based CI/CD pipeline to automate the build and deployment process. The pipeline is configured as follows:

1. Trigger: A new commit is pushed to the main branch.
2. Build: A release APK is built using Flutter's `flutter build apk --release`.
3. Distribute: The generated APK is automatically uploaded to Firebase App Distribution.
4. Notification: Testers receive an email with a download link to install the new version on their devices.

This workflow ensures that the team can continuously deliver tested builds without manual packaging, speeding up feedback and reducing human error.

Firebase App Distribution provided an easy and secure way to share pre-release versions with testers and supervisors during the development process.

4.6 Backend Testing

4.6.1 Auth Routes (auth.py)

Test Case ID	Scenario	Expected Result
TC-AU-01	Login with valid email and password	Login successful, JWT token returned
TC-AU-02	Login with non-existent email	401 Unauthorized, "User not found"
TC-AU-03	Login with wrong password	401 Unauthorized, "Incorrect password"
TC-AU-04	Login with missing credentials	400 Bad Request, "Missing data"
TC-AU-05	Login with invalid email format	400 Bad Request, "Invalid email format"
TC-AU-06	Login with banned or unapproved user	403 Forbidden, "Account restricted"

4.6.2 Admin Routes (admin.py)

Test Case ID	Scenario	Expected Result
TC-AD-01	Get all users with admin access	200 OK, list of users returned
TC-AD-02	Get all users without admin access	403 Forbidden
TC-AD-03	Approve a user with a valid ID	200 OK, user status updated
TC-AD-04	Approve a user with an invalid ID	404 Not Found, "User not found"
TC-AD-05	Ban a user with a valid ID	200 OK, user status updated
TC-AD-06	Ban a user with an invalid ID	404 Not Found, "User not found"

4.6.3 Advertisements Routes (advertisements.py)

Test Case ID	Scenario	Expected Result
TC-ADVERT-01	Get all advertisements	200 OK, list of ads returned
TC-ADVERT-02	Create a new advertisement (valid data)	201 Created, ad added successfully
TC-ADVERT-03	Create advertisement (missing fields)	400 Bad Request, validation error
TC-ADVERT-04	Update the advertisement with a valid ID	200 OK, advertisement updated
TC-ADVERT-05	Update the advertisement with an invalid ID	404 Not Found, "Ad not found"
TC-ADVERT-06	Delete advertisement with a valid ID	200 OK, advertisement deleted
TC-ADVERT-07	Delete the advertisement with the invalid ID	404 Not Found, "Ad not found"

4.6.4 Messages Routes (messages.py)

Test Case ID	Scenario	Expected Result
TC-MSG-01	Get all messages for the current user	200 OK, list of messages returned
TC-MSG-02	Send message with valid data	201 Created, message sent successfully
TC-MSG-03	Send message with missing fields	400 Bad Request, validation error
TC-MSG-04	Delete message with valid ID	200 OK, message deleted successfully
TC-MSG-05	Delete message with invalid ID	404 Not Found, "Message not found"

4.6.5 Posts Routes (posts.py)

Test Case ID	Scenario	Expected Result
TC-POST-01	Get all posts	200 OK, list of posts returned
TC-POST-02	Create post with valid data	201 Created, post added successfully
TC-POST-03	Create post with missing fields	400 Bad Request, validation error
TC-POST-04	Update post with valid ID	200 OK, post updated successfully
TC-POST-05	Update post with invalid ID	404 Not Found, "Post not found"
TC-POST-06	Delete post with valid ID	200 OK, post deleted successfully
TC-POST-07	Delete post with invalid ID	404 Not Found, "Post not found"

4.6.6 Public Services Routes (public_services.py)

Test Case ID	Scenario	Expected Result
TC-PS-01	Get all public services	200 OK, list of services returned
TC-PS-02	Create public service with valid data	201 Created, service added successfully
TC-PS-03	Create public service with missing fields	400 Bad Request, validation error
TC-PS-04	Update public service with valid ID	200 OK, service updated successfully
TC-PS-05	Update public service with invalid ID	404 Not Found, "Service not found"
TC-PS-06	Delete public service with valid ID	200 OK, service deleted successfully
TC-PS-07	Delete public service with invalid ID	404 Not Found, "Service not found"

4.7 Testing Authentication Components (Register.jsx, Login.jsx)

4.7.1 Testing Register Component (Register.jsx)

Testing Strategy:
The Register.jsx component was tested using unit and integration tests to ensure that all form fields render correctly, validation is applied as expected, and data is successfully submitted using Axios. The interface is built using **Mantine UI** with a responsive layout.

Testing Objectives:

- Ensure the following fields are correct: **Username**, **Password**, **Full Name**, **Building Number**, and **Apartment Number**.
- Validate required fields and password rules before submission.
- Verify that Axios correctly sends the registration data to the backend.
- Confirm that upon successful registration, the user is redirected to the login.
- Test responsiveness and accessibility.

Test Case	Description	Input	Expected Output	Status
TC-1	Render Register Form	Render <Register />	All fields are rendered correctly without icons	☑ Pass
TC-2	Validation	Leave fields empty	Error messages shown below inputs	☑ Pass
TC-3	Valid Submission	Submit with valid inputs	Axios request sent, redirect triggered	☑ Pass
TC-4	Backend Error	Simulate server error	Error message shown inline below the form	☑ Pass

---

4.7.2 Testing Login Component (Login.jsx)

Testing Strategy:
The Login.jsx component was tested to ensure that the **Username** and **Password** fields function correctly, and data is sent properly to the backend via Axios. The component provides inline feedback for invalid credentials.

Testing Objectives:

- Verify rendering of input fields.
- Validate empty or incorrect input behavior.
- Confirm Axios sends a login request and handles success/failure properly.
- Ensure redirection to the Landing page after successful login.
- Socket.IO Client is initialized after login for real-time chat functionality.

Test Case	Description	Input	Expected Output	Status
TC-1	Render Login Form	Render <Login />	The username and password fields appear	☑ Pass
TC-2	Invalid Input	Submitting with empty or incorrect data	Inline error message shown	☑ Pass
TC-3	Valid Login	Submit valid data	Axios request sent, user redirected, Socket.IO connected	☑ Pass
TC-4	Backend Error	Simulate a wrong login	The error message appears below the form	☑ Pass

---

Chapter 5: System Integration and Interoperability

This chapter provides an in-depth exploration of the integration architecture and interoperability mechanisms of the Smart Home System, unifying subsystems—Smart Security (Section 3.1), Fire Detection (Section 3.2), Face Recognition (Section 3.3), Gas Detection (Section 3.3.8), Light and Comfort System (Section 3.4), Mobile Application (Section 3.5), and Compound Website (Section 3.7)—into a cohesive, scalable solution. It details the integration architecture, subsystem communication protocols, data flow, challenges encountered during integration, solutions implemented, and interoperability with external systems. Insights from testing (Chapter 4) and proposed enhancements (e.g., account-based light control, multi-camera support) are incorporated to highlight

robustness and future scalability. Workflow diagrams are described to clarify system interactions, enhancing understanding of the integrated system.

5.1 Integration Architecture

The Smart Home System employs a modular, layered architecture with centralized cloud coordination to ensure seamless subsystem integration. The architecture is divided into three primary layers:

- Hardware Layer:
  - Comprises Arduino Mega (Smart Security), ESP32 WROOM 38-Pin (Light and Comfort), and Raspberry Pi 5 (Fire Detection, Face Recognition, Gas Detection).
  - Handles local processing, sensor data acquisition, and actuator control (e.g., solenoid lock, LEDs, fans).
  - Example: Arduino processes fingerprint and keypad inputs (Section 3.1.2.1), while Raspberry Pi runs YOLOv8 for fire detection (Section 3.2.4.3).
- Software Layer:
  - Includes the Flutter-based Mobile Application (Section 3.5) for user interaction and the Flask-based backend (Sections 3.2.4.7, 3.3.4.8, 3.6) for data processing and streaming.
  - Manages real-time user interfaces, API endpoints, and video streaming (Section 4.2.7).
- Cloud Layer:
  - Utilizes Firebase Realtime Database as the central hub for data synchronization, authentication, and event logging (Sections 3.4.4, 3.5.5.7, 3.6.8).
  - Supports bi-directional communication between hardware and software components.
- Communication Protocols:
  - Serial Communication: Used for Arduino-Raspberry Pi interactions, e.g., sending face recognition results to control the solenoid lock (Section 4.1.5).
  - Wi-Fi (HTTP/REST): Facilitates ESP32 and Raspberry Pi communication with Firebase (Sections 3.4.4, 3.3.4.7).
  - WebSocket (Socket.IO): Enables real-time chat in the Compound Website (Section 3.7.7).
  - MQTT (Future): Planned for lightweight IoT communication (previously Section 5.4.1).
- Scalability Design:
  - Modular architecture allows adding new subsystems (e.g., CO2 sensors, previously Section 5.3.1) without disrupting existing functionality.
  - Firebase's scalable database structure supports multi-home deployments (previously Section 5.5.7).
- Security Measures:

    - JWT-based authentication for backend APIs (Section 4.6.1).
    - Encrypted Firebase communication to protect sensitive data (e.g., face encodings, Section 3.3.4.7).

5.2 Subsystem Communication and Data Flow

Each subsystem communicates through Firebase, with local processing to ensure reliability during network disruptions. The following details the data flow for key subsystems:

- Smart Security System (Section 3.1):
    - Input: Fingerprint sensor (R307), keypad (4x4), or face recognition (via Raspberry Pi) sends authentication data to Arduino Mega.
    - Processing: Arduino validates inputs against stored credentials (EEPROM, Section 3.1.2.5.2) and logs events with RTC timestamps (Section 4.1.4).
    - Output: Controls solenoid lock (via relay) and updates Firebase with access logs.
    - Data Flow: Fingerprint/Keypad → Arduino → Firebase → Mobile App (notifications, Section 3.5.5.9).
    - Local Fallback: EEPROM stores logs during network outages (Section 3.1.2.5.2).
- Fire Detection (Section 3.2):
    - Input: Pi Camera captures video frames, processed by YOLOv8 Nano on Raspberry Pi (Section 4.2.2).
    - Processing: YOLOv8 detects fire, annotates frames, and updates detection state (Section 3.2.4.5).
    - Output: Triggers Pygame audio alerts and updates Firebase flags (Section 4.2.6).
    - Data Flow: Camera → YOLOv8 → Flask (streaming, Section 4.2.7) → Firebase → Mobile App (real-time alerts, Section 4.2.9).
    - Performance: Inference time ~170–190ms, streaming at 13–17 FPS (Section 4.3.1).
- Face Recognition (Section 3.3):
    - Input: Pi Camera captures frames, processed by face_recognition library (HOG backend, Section 4.3.1).
    - Processing: Matches faces against stored encodings, with 91.8% accuracy (Section 4.3.1).
    - Output: Sends unlock commands to Arduino via Serial and logs events to Firebase (Section 3.3.4.7).
    - Data Flow: Camera → Face Recognition → Firebase → Mobile App (notifications) → Flask (streaming, Section 3.3.4.8).
    - Error Handling: Rejects unknown faces, logs false positives (2.3%, Section 4.3.1).
- Gas Detection (Section 3.3.8):

    - Input: MQ5 sensor detects gas levels (LPG, methane), processed by Raspberry Pi (Section 4.3.2).
    - Processing: Threshold-based detection triggers alerts (~2.1s response time, Section 4.3.2).
    - Output: Plays audio warning via USB speaker and updates Firebase (Section 4.3.2).
    - Data Flow: MQ5 → Raspberry Pi → Firebase → Mobile App.
- Light and Comfort System (Section 3.4):
    - Input: PIR, LDR, IR (x2), and DHT11 sensors send motion, light, and temperature data to ESP32 (Section 4.4.1).
    - Processing: ESP32 manages automatic/manual modes (Section 3.4.2–3.4.3).
    - Output: Controls LEDs and fans, syncing states to Firebase (Section 4.4.4).
    - Data Flow: Sensors → ESP32 → Firebase → Mobile App (control, Section 3.5.5.3).
    - Optimization: Debounce delay for IR sensors (Section 4.4.2) ensures accurate entry counting.
- Mobile Application (Section 3.5):
    - Role: Central interface for monitoring (sensor states, alerts) and control (lights, fans, Section 3.5.5.3).
    - Data Flow: Firebase ↔ Flutter App (bi-directional, real-time updates <2s, Section 4.2.9).
    - Features: Supports account-based control (previously Section 5.5.1) and multi-device access (Section 4.2.9).
- Compound Website (Section 3.7):
    - Role: Community platform for chat, advertisements, and services (Section 4.6).
    - Data Flow: React frontend → Flask APIs → Firebase → Socket.IO (real-time chat, Section 3.7.7).
    - Performance: API response times ~200ms for valid requests (Section 4.6.1).

5.3 Integration Challenges and Solutions

Integrating heterogeneous subsystems posed significant challenges, addressed through iterative testing and optimization:

- Challenge: Firebase Latency (Section 4.4.4):
    - Issue: Slow Wi-Fi delayed ESP32-Firebase updates, causing code stalls in the Light and Comfort System (Section 4.4.4).
    - Impact: Affected real-time control (e.g., light toggling delayed by 1.2–1.8s).
    - Solution:
        - Reduced data size by using smaller data types (e.g., int8 instead of int32).

        - Implemented selective variable sending, ignoring unchanged values.
        - Added 100ms delays between sends to reduce errors (Section 4.4.4).
        - Result: Stable operation in both manual and automatic modes.
- Challenge: RTC Synchronization (Section 4.1.4):
    - Issue: RTC module (DS3231) drift caused inaccurate access log timestamps in Smart Security.
    - Impact: Misaligned logs disrupted tracking (Section 4.1.4).
    - Solution:
        - Periodic Firebase-based time synchronization every 24 hours.
        - Buffered logs locally in EEPROM during network outages.
        - Result: Accurate timestamps, verified over 48-hour tests.
- Challenge: Multi-Subsystem Coordination:
    - Issue: Simultaneous Firebase updates from Fire Detection, Face Recognition, and Light Systems caused data overwrites (Sections 4.2.8, 4.3.1, 4.4.4).
    - Impact: Inconsistent alert states in the Mobile App.
    - Solution:
        - Implemented priority-based queuing (e.g., fire alerts prioritized over light updates).
        - Used indexed Firebase nodes (e.g., /homes/home1/fire, /homes/home1/lights) to segregate data.
        - Result: No overwrites during 1-hour stress tests (Section 4.2.8).
- Challenge: Camera Integration (Section 4.1.5):
    - Issue: Incomplete integration of face recognition with Arduino for door control due to time constraints.
    - Impact: Limited face-based entry in prototype (Section 4.1.5).
    - Solution:
        - Temporary Serial communication for unlock commands.
        - Planned full integration with dedicated API endpoints (previously Section 5.1).
        - Result: Functional but not finalized, stable for single-user tests.
- Challenge: Flask Streaming Load (Section 4.2.7):
    - Issue: High CPU usage (~80%) on Raspberry Pi during simultaneous fire and face recognition streaming.
    - Impact: Reduced frame rates (13–17 FPS, Section 4.3.1).
    - Solution:
        - Lowered stream resolution to 480p and frame rate to 15 FPS.
        - Optimized Flask server with async I/O for concurrent requests.
        - Result: Reduced CPU usage to ~60%, maintaining stream stability.
- Challenge: Power Management:
    - Issue: High power consumption by Raspberry Pi and ESP32 during continuous operation.
    - Impact: Risk of overheating during extended tests (Section 4.2.8).
    - Solution:

        - Implemented sleep modes for ESP32 when idle (Section 3.4.2).
        - Used efficient power supplies (5V USB-C, Section 4.3.2).
        - Result: Stable 2-hour operation with no overheating.

5.4 Interoperability with External Systems

Interoperability ensures the system's compatibility with external platforms, enhancing scalability and market relevance:

- IoT Standards:
    - Current: Firebase Realtime Database supports HTTP/REST APIs, compatible with most IoT platforms (Sections 3.4.4, 3.6.7).
    - Future: Support for Matter protocol to integrate with Google Nest, Samsung SmartThings (Section 2.2.4, previously Section 5.5.4).
    - MQTT Integration: Planned for lightweight communication, reducing Firebase dependency (previously Section 5.4.1).
- Smart Home Ecosystems:
    - Google Assistant: Firebase APIs enable voice control for lights and fans (previously Section 5.5.1).
    - HomeKit/Alexa: Planned MQTT bridge for compatibility (previously Section 5.4.1).
- External Sensors/Devices:
    - ESP32's GPIO pins support additional sensors (e.g., CO2, previously Section 5.3.1).
    - Raspberry Pi supports USB-based AI accelerators (e.g., Google Coral, previously Section 5.2) for enhanced AI processing.
- Cloud Platforms:
    - Firebase can integrate with AWS IoT or Google Cloud IoT for larger deployments (previously Section 5.5.7).
    - Local caching proposed for offline operation (previously Section 5.5.3).
- Security Protocols:
    - JWT authentication in Flask backend (Section 4.6.1) aligns with industry standards.
    - Planned OAuth 2.0 for Compound Website (previously Section 5.6) ensures secure user access.
- Interoperability Challenges:
    - Issue: Limited support for proprietary protocols (e.g., Zigbee) in prototype.
    - Solution: Adopt open standards like Matter and modular APIs for future compatibility.
    - Issue: Cross-platform user authentication complexity.
    - Solution: Unified Firebase Authentication across Mobile App and Website (Section 3.5.5.7, 3.6.8).

5.5 Performance and Scalability Considerations

To ensure robust integration, performance metrics and scalability were analyzed:

- Performance Metrics:
  - Smart Security: Door response time <1.2s, authentication accuracy 95% (Section 4.1).
  - Fire Detection: Inference time 170–190ms, streaming 13–17 FPS (Section 4.3.1).
  - Face Recognition: Accuracy 91.8%, false positive rate 2.3% (Section 4.3.1).
  - Gas Detection: Detection time 2.1s, zero false positives (Section 4.3.2).
  - Light and Comfort: Sensor response time ~200ms, Firebase sync 1.2–1.8s (Section 4.4.4).
  - Backend: API response time ~200ms, 100% success for valid requests (Section 4.6.1).
- Scalability:
  - Single-Home: Supports 2 rooms, expandable with additional ESP32 modules (~$29/room, Section 8.1.1).
  - Multi-Home: Firebase indexing supports 100+ users (previously Section 5.5.7).
  - Proposed Enhancements: Bluetooth fallback (previously Section 5.4.1), multi-camera support (previously Section 5.2).
- Stress Testing:
  - Continuous operation for 1–2 hours showed no memory leaks (Section 4.2.8).
  - Multi-device testing confirmed stable Firebase updates (Section 4.2.9).

5.6 Conclusion

The Smart Home System's integration architecture leverages Firebase for centralized coordination, enabling seamless communication between Arduino, ESP32, Raspberry Pi, Flutter, and Flask components. Challenges like Firebase latency, RTC drift, and streaming load were addressed through optimized data handling, priority queuing, and modular design (Section 4). Interoperability with IoT standards (e.g., Matter) and external ecosystems (e.g., Google Assistant) ensures future scalability. The described workflow diagrams clarify data flows, supporting the system's robustness and potential for multi-home deployment. This integration framework positions the system as a viable, scalable solution for smart home applications.

Chapter6 : Future Work & Conclusion

6.1 Smart Security System:

6.1.1 Mobile Application Integration

- Develop a mobile app (Android/iOS) to allow users to remotely monitor and control the security system (arm/disarm, receive notifications, view logs).
- Use Wi-Fi, Bluetooth, or GSM connectivity for real-time alerts.

6.1.2 Remote Notifications

- Integrate SMS, push notifications, or email alerts for suspicious activity, intrusion detection, or system errors.
- Use cloud services (Firebase, AWS IoT, etc.) for scalable remote communication.

6.1.3 Web Interface and Cloud Dashboard

- Create a web-based dashboard to monitor system status, manage users, and view access logs from anywhere.
- Enable remote firmware upgrades and configuration through the web interface.

6.1.4 Voice Assistant Support

- Integrate with Alexa, Google Assistant, or other smart home assistants for voice-based control (e.g., "Unlock the door", "Show me access logs").

6.1.5 Advanced Biometrics

- Support additional biometric authentication methods such as facial recognition or RFID cards for multi-factor security.

6.2 Face Recognition:

- Anti-Spoofing and Liveness Detection

Add mechanisms to detect fake images or videos (e.g., printed photos or phone screens) using blink detection, depth sensing, or movement tracking. This will significantly improve system security and prevent unauthorized access via image-based attacks.

- Offline Mode with Local Logging

Implement a local database fallback to allow the system to operate and log recognition events even if the internet connection or Firebase becomes temporarily unavailable.

-Mobile App Notifications

Integrate Firebase Cloud Messaging (FCM) to send instant alerts to the user’s phone when someone is detected at the door (recognized or unknown).

-Voice Response System
Add a voice-based feedback system to inform the visitor (e.g., “Access granted”, “Access denied”) using a speaker connected to the Raspberry Pi.
-User Management Dashboard
Develop a web-based or mobile admin panel to:
- Add/remove authorized faces dynamically
- View access logs
- Monitor live stream securely
- Adjust recognition sensitivity and settings
-Multiple Camera Integration
Expand the system to support multiple cameras (front door, back door, hallway) and synchronize the streams and recognition logs in a unified view.
-Face Recognition Model Optimization
Replace the current HOG-based detection with CNN-based models (e.g., ResNet or MTCNN) for improved accuracy, particularly in low-light conditions or when capturing angled faces.
-Edge AI Accelerator
Incorporate a USB-based AI accelerator (e.g., Google Coral or Intel Neural Stick) to further improve inference speed and allow more advanced models on Raspberry Pi.
-Two-Factor Authentication
Combine face recognition with a second method, like mobile OTP or voice recognition, to enhance physical access control for critical areas.

6.3 Gas Detection System:

-Multi-Gas Classification
Upgrade the system to detect and differentiate between various gases (e.g., LPG, CO, CH_4, CO_2) using multi-sensor arrays and machine learning classifiers to improve accuracy and enable context-aware responses.
-Threshold Customization Interface
Develop a mobile or web interface allowing users to set custom gas level thresholds, manually disable alarms, or dynamically configure alert behavior.
-Integration with Smart Ventilation
Enable automatic triggering of exhaust fans, ventilation systems, or window actuators in response to high gas concentrations for immediate mitigation.
-Cloud-Based Data Analytics
Store sensor data in the cloud (e.g., Firebase or ThingsBoard) for historical analysis and anomaly detection using AI models.
-Mobile Push Notifications
Implement real-time notifications to users’ smartphones via Firebase Cloud Messaging (FCM) or similar services when gas is detected.
-Battery Backup and Offline Logging
Incorporate a UPS module and offline storage to ensure data logging and alarm functionality during power outages or network failures.
-Location-Based Multi-Zone Monitoring
Deploy multiple sensors across different rooms/zones with a centralized dashboard to pinpoint exact leak locations.

-Integration with Fire Detection Module
Combine gas and fire detection into a unified emergency response system with cross-triggered alarms and coordinated actions.

6.4 Light & comfort system

1. Add a Bluetooth connectivity option so that users can switch from Wi-Fi to Bluetooth if they experience issues or become frustrated with Wi-Fi
2. Add Automatic button for each room so each user can choose whether his room is on manual or automatic mode . Maybe all the family like the manual mode , but only one room is for children who can't use their phone so only this room will be Automatic . Maybe all the family likes the manual mode but they like the outing lights to depend on the LDR so only the outing will be Automatic

6.5 Mobile application

- **Voice Assistant Integration:** Implement integration with voice assistants like Google Assistant or Amazon Alexa for hands-free device control (e.g., “Turn on the kitchen light”).

- **Offline Mode Enhancement:** Improve offline functionality by caching device states and syncing when the connection is restored.

- **Multi-language Support:** Add localization to support multiple languages, making the app accessible to a wider audience.

- Smart Schedule & Automation Rules: Let users define rules (e.g., 'Turn off all lights at 11 PM') or create schedules for recurring actions.

6.6 Compound Website:
- Adding the feature of sending photos and videos in the chat within the compound website.
- Adding the ability for users to evaluate craftsmen, restaurants, and cafes in the public services center on the compound website.

6.7 Conclusion

This graduation project represents a comprehensive attempt to implement a smart home system using modern technologies. Our solution integrates embedded systems, sensors, and artificial intelligence to provide a safer and more convenient living environment. The gas detection system, powered by the MQ5 sensor, offers real-time alerts to protect residents from hazardous leaks. In addition, the integration of a temperature sensor and an automatic fan provides responsive climate control within the house.

The smart lighting system enables users to control and automate lighting through a mobile application for improved energy efficiency. Security was another core focus, achieved through a smart door system that authenticates residents via fingerprint and access permissions. To further enhance safety, we implemented two AI-powered camera modules that provide intelligent monitoring. One camera, positioned at the main entrance, is capable of facial recognition to allow access only to authorized individuals. The second camera, located in the kitchen, is designed to detect fire and send early alerts to the mobile application. These systems are seamlessly integrated through an intuitive mobile application interface.

Additionally, we developed a service-oriented website to assist compound residents with their daily needs and enhance community engagement. This platform allows residents to communicate with each other via chat and easily communicate with the compound owner. It also includes an advertising center for advertising anything for sale and a services center to facilitate daily life.

By merging hardware and software, we built a fully functional smart home prototype that can be scaled and customized.

The project showcases our ability to apply theoretical knowledge to real-world engineering challenges. We collaborated effectively as a team, managing tasks ranging from system design and coding to hardware integration and testing. Working with real components and live data streams provided valuable insights into system reliability and user expectations. The experience helped us understand the importance of user-centered design and responsive system behavior. It also taught us how to address practical challenges like power efficiency, security, and data communication. This hands-on work reinforced our knowledge of programming, circuit design, and interface development.

We learned how to integrate different systems using microcontrollers, sensors, and wireless communication protocols. Our smart home prototype embodies a blend of engineering principles, creative problem-solving, and meticulous attention to detail. We are confident that our solution meets the key objectives of safety, automation, and user

accessibility. Furthermore, this project lays the groundwork for future innovations in smart living environments. It reflects the growing demand for connected homes and the role of AI in everyday life. By working on this project, we have developed both technical and soft skills that will benefit us in our professional careers. We also gained experience in presenting our ideas, documenting processes, and receiving constructive feedback.

The success of this project would not have been possible without persistence, teamwork, and continuous learning. We hope this work inspires further exploration in the field of smart technologies. We believe that such innovations will play a vital role in shaping the homes of tomorrow.

In conclusion, we take pride in what we have achieved and look forward to contributing to technological advancement in the years to come.

7. Chapter7: References:

Arduino. (n.d.). Arduino Mega 2560 Rev3.

https://store.arduino.cc/products/arduino-mega-2560-rev3

Arduino Official Website (All Arduino C/C++ Code References & Libraries).

https://www.arduino.cc/

Datasheet: Arduino Mega 2560

https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf

Datasheet: DS1302 Real Time Clock

https://datasheets.maximintegrated.com/en/ds/DS1302.pdf

Datasheet: Typical Fingerprint Sensor (e.g., GT-521F52)

https://cdn.sparkfun.com/datasheets/Sensors/Biometric/GT-521F52.pdf

Datasheet: 4x4 Matrix Keypad

https://www.electronicwings.com/nodemcu/4x4-matrix-keypad-interfacing-with-nodemcu

Datasheet Example (16x2 LCD with I2C backpack - PCF8574):
16X2 LCD Datasheet, PDF - Alldatasheet

R307 Official Datasheet:
Fingerprint Module R307 – Future Electronics Egypt

Responsive Web Design

- **Resource:** MDN Web Docs
- **Summary:** This resource provides comprehensive guidance on making websites responsive, including using media queries, flexible grids, and responsive images.

React.js for Building User Interfaces

- **Resource:** React Official Documentation
- **Summary:** Official documentation for React, offering in-depth tutorials and examples for building user interfaces using components.

Compound Design Best Practices

- **Resource 1:** aspirity.com
- **Summary:** Modern layouts featuring **clean hero sections** with high-quality photos and clear navigation—ideal for showcasing services or announcements.

- **Resource 2:** subframe.com
- **Summary:** Showing examples of platforms with chat for users, an admin dashboard, or an advertisement for services or units.

Web Performance Optimization

- **Resource:** Google Developers
- **Summary:** Guidelines and techniques for optimizing web performance, including image optimization, lazy loading, and minimizing render-blocking resources.

JavaScript for Modern Web Development

- **Resource:** https://www.w3schools.com/jsrEF/default.asp
- **Summary:** A modern JavaScript tutorial covering all aspects of the language, from basics to advanced topics.

Axios:

- **Resource:** Axios
- **Summary:** Axios is a simple promise-based HTTP client for the browser and Node.js. Axios provides a simple-to-use library in a small package with a very extensible interface.

Socket.io Client:

- **Resource:** Socket.IO
- **Summary:** Bidirectional and low-latency communication for every platform.

Mantine UI:

- **Resource:** Mantine

Summary: A fully featured React components library

M. A. Al-qaness, “IoT-based smart home automation system using sensor networks and deep learning,” *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3152–3163, Apr. 2020, doi: 10.1109/JIOT.2019.2959987.

Arduino, “Arduino Mega 2560 Documentation,” Arduino.cc, 2023. [Online]. Available: https://docs.arduino.cc/hardware/mega-2560. [Accessed: Jul. 7, 2025].

C. Jocher, A. Chaurasia, and J. Qiu, “YOLOv8: Ultralytics object detection and tracking,” Ultralytics, 2023. [Online]. Available: https://docs.ultralytics.com/yolov8/. [Accessed: Jul. 7, 2025].

P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2001, pp. I-511–I-518, doi: 10.1109/CVPR.2001.990517.

Flutter, “Flutter SDK Documentation,” Flutter.dev, 2024. [Online]. Available: https://docs.flutter.dev/. [Accessed: Jul. 7, 2025].

Google, “Firebase Realtime Database,” Firebase.google.com, 2024. [Online]. Available: https://firebase.google.com/docs/database. [Accessed: Jul. 7, 2025].

D. Gay, *The ESP32 Microcontroller: Programming and Applications*, 2nd ed. New York, NY, USA: McGraw-Hill, 2022.

A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2018.

S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, “Security, privacy and trust in Internet of Things: The road ahead,” *Comput. Netw.*, vol. 76, pp. 146–164, Jan. 2015, doi: 10.1016/j.comnet.2014.11.008.

D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15. [Online]. Available: https://arxiv.org/abs/1412.6980.

Espressif Systems, “ESP32-WROOM-32 Technical Reference Manual,” Espressif.com, 2023. [Online]. Available: https://www.espressif.com/en/support/documents/technical-documents. [Accessed: Jul. 7, 2025].

J. Brownlee, “How to develop a face recognition system using FaceNet in Python,” MachineLearningMastery.com, Aug. 2020. [Online]. Available: https://machinelearningmastery.com/face-recognition-using-facenet/. [Accessed: Jul. 7, 2025].

Flask, “Flask Web Framework Documentation,” Palletsprojects.com, 2024. [Online]. Available: https://flask.palletsprojects.com/. [Accessed: Jul. 7, 2025].

IEEE Standards Association, “IEEE 829-2008: Standard for Software and System Test Documentation,” IEEE, 2008. [Online]. Available: https://standards.ieee.org/standard/829-2008.html. [Accessed: Jul. 7, 2025].

A. Monk, *Fundamentals of Human-Computer Interaction*, 2nd ed. London, UK: Academic Press, 2019.