**Project Title:** Home Automation

**Name:** N.Venkata Bhargav

**Branch/Section:** CSE-C

**RollNo:** 23B81A05J8

**Abstract:**

An OOP smart home environment in Python provides a modular and scalable foundation for managing smart devices, with each type of device modeled as a class leveraging inheritance and polymorphism. The environment is structured around a device abstraction (SmartDevice base class) with concrete subclasses (LightDevice, ThermostatDeviceetc.) . encapsulating device-specific logic and behaviors.

The data layer uses Supabase’s PostgreSQL backend for persistent storage of device states, configuration, and user-related data. Python interacts with Supabase via RESTful or direct Postgres API calls, abstracted into repository or service classes to maintain OOP separation of concerns. This approach decouples business logic from database access, promotes code reuse, and facilitates robust unit testing.

Typical modules in this environment include:

* Device base and subclass definitions following OOP best practices.
* Repository/service classes to manage CRUD operations with Supabase/PostgreSQL.
* Application controller for orchestrating device interactions and data transactions.

This architecture ensures extensibility for new device types, clear business-data separation, and integration with a scalable cloud-native backend.

## **Introduction**

### **Background of the Domain**

Smart home systems are part of the rapidly growing Internet of Things (IoT) domain, where everyday devices such as lights, thermostats, and cameras are interconnected to enhance convenience, security, and energy efficiency. These systems require robust and scalable software architectures to manage diverse devices. Object-Oriented Programming (OOP) provides an ideal paradigm for designing such systems because it models real-world entities as classes and objects with shared and specialized behaviors. PostgreSQL adds a reliable data persistence layer to store and retrieve device information effectively.

### **Motivation for the Project**

The project aims to strengthen understanding of OOP principles by applying them to a realistic scenario. Instead of focusing on hardware integration, it models a smart home environment entirely in Python to practice inheritance, polymorphism, and modular design. Integrating PostgreSQL demonstrates how real-world applications persist data, bridging the gap between object-oriented design and database systems. This approach provides practical experience while creating a flexible foundation for future IoT development.

### **Key Objectives of the Project**

* Build a base Device class with shared attributes and methods.
* Create subclasses like Light, Thermostat, and Camera using inheritance.
* Develop a HomeController to manage and control multiple devices.
* Implement a DatabaseManager to handle PostgreSQL integration.
* Ensure modularity and scalability for easy extension to new devices or features.

This project delivers a Python-based smart home prototype showcasing OOP principles and database persistence in a clear, practical way.

**Problem Statement:**

In modern homes, managing multiple smart devices such as lights, thermostats, and cameras individually can be cumbersome and inefficient. Users often face difficulty in monitoring device status, controlling devices remotely, and maintaining consistency in device configurations. Additionally, without centralized control and data storage, tracking device usage, updating settings, or retrieving device history becomes error-prone and time-consuming.

This project aims to develop a centralized smart home management system that allows users to control, monitor, and maintain all connected devices efficiently. By providing a unified interface through a HomeController and ensuring persistent data storage via Database Manager, the system eliminates manual errors, improves automation, and enhances the overall user experience in managing smart home devices.

### **Scope of the Project**

**Include:**

* Centralized management of smart devices (Lights, Thermostats, Cameras)
* Adding, removing, and controlling devices
* Monitoring device status in real-time
* Storing and updating device information in a database
* Basic device operations like turn on/off, adjust brightness, temperature, or camera recording

**Exclude:**

* Graphical User Interface (GUI) or mobile/web app
* Advanced analytics or AI-based automation
* Integration with third-party smart home platforms (e.g., Alexa, Google Home)
* Multi-user or role-based access control

### **System Requirements**

**Functional Requirements (Features):**

* Add/Edit/Delete smart devices (Lights, Thermostats, Cameras)
* Control devices (turn on/off, adjust brightness, temperature, or camera settings)
* Monitor device status in real-time
* Manage device list (add/remove devices)
* Fetch device information from the database
* Update device status and settings in the database

**Non-Functional Requirements:**

* **Security:** Ensure secure storage of device data and user configurations
* **Scalability:** Support multiple devices and rooms in a home
* **Usability:** Simple, easy-to-use interface (CLI or API)
* **Reliability:** Devices and controller should function correctly even if some devices fail
* **Maintainability:** Easy to extend with new device types in the future

### **System Analysis**

**Existing System:**

* Smart home devices are often controlled individually or via multiple apps provided by different manufacturers.
* Users manually check each device, leading to inconsistent control and monitoring.
* No centralized storage of device data, making it difficult to track use or configurations.

**Proposed System:**

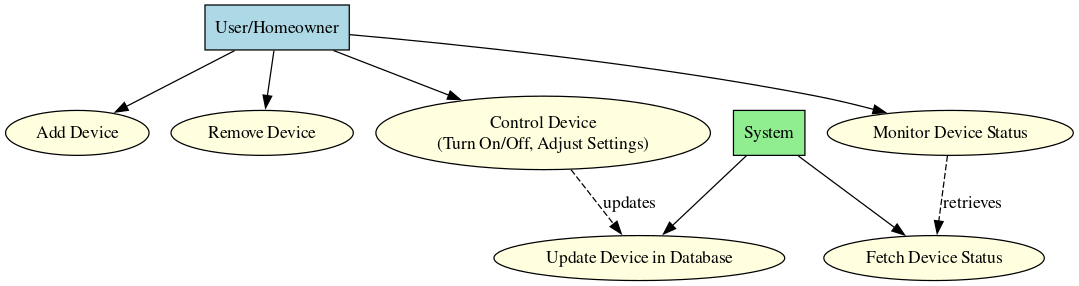
* An **automated centralized system** using Python for controlling devices and a database for storing device information.
* HomeController manages all devices from a single interface.
* DatabaseManager maintains persistent records of devices, their status, and settings.

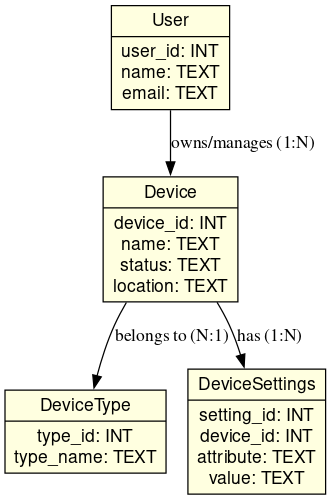
**Advantages of Proposed System:**

* **Reduces human error:** Users no longer need to remember manual settings or check devices individually.
* **Saves time:** Centralized control allows managing multiple devices quickly.
* **Accurate tracking:** Device status and configurations are stored in a database for easy retrieval and updates.
* **Scalable and maintainable:** New devices can be added without changing the system structure.

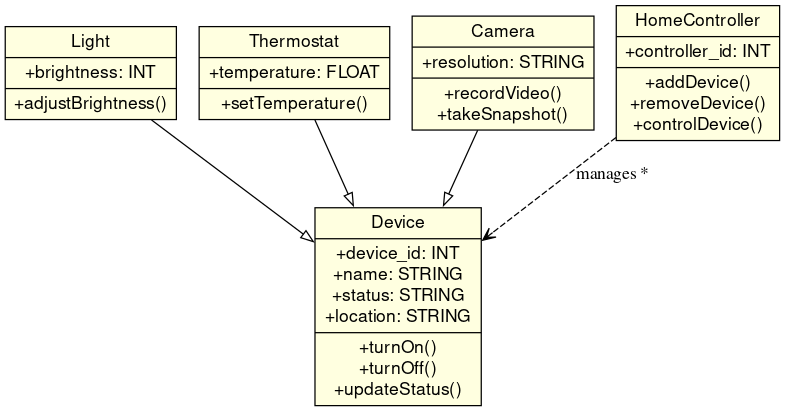
**System Design:**

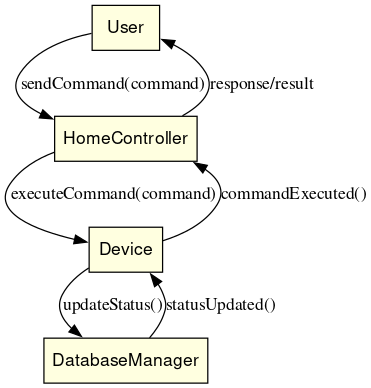
**Use case Diagram:**



**ER Diagram:  
 **

**Class Diagram:**

****

**Sequence Diagram:**

**Database Design:**

* **Users:** user\_id, name, email, phone, city
* **Devices:** device\_id, name, status, location, type\_id, user\_id
* **Device Types:** type\_id, type\_name
* **Device Settings:** setting\_id, device\_id, attribute, value
* **Home Controller:** controller\_id, name, location, user\_id
* **Device Logs:** log\_id, device\_id, timestamp, action

**System Architecture:**

User/Homeowner → HomeController → Device → DatabaseManager → Supabase (Database)

**Layer Explanations:**

1. **User/Homeowner**
   * End**-**user interacting with the system.
   * Sends commands to control devices or check their status.
2. **HomeController**
   * Acts as the **central coordinator** of smart home devices.
   * Receives user commands and forwards them to the appropriate devices.
   * Updates device states and fetches status when requested.
3. **Device**
   * Represents **smart devices** (Lights, Thermostats, Cameras, etc.).
   * Executes commands (turn on/off, adjust settings) and reports status.
   * Maintains device-specific logic.
4. **DatabaseManager (DAO Layer)**
   * Handles all **interactions with the database**.
   * Updates device status, fetches device settings, and stores logs.
5. **Supabase (Database)**
   * Stores all **persistent data**: users, devices, device types, settings, and logs.
   * Provides an API for reading and writing data efficiently.

### **Expected Outcomes**

* **Efficient Device Management**
  + Add, remove, and organize devices easily.
  + Devices linked to specific users for personalized control.
* **Real-Time Device Control**
  + Turn devices on/off and adjust settings through the HomeController.
  + Immediate feedback on device status.
* **Centralized Monitoring**
  + Monitor all devices’ statuses in one place.
  + Logs of device actions for tracking and auditing.
* **Data Persistence and Reliability**
  + Device information, user data, and settings stored reliably in Supabase.
  + Updates reflected accurately in the database.
* **Simple CLI for Operations**
  + User-friendly command-line interface for interacting with the system.
  + Commands to control devices, view status, and update settings.
* **Scalability and Extensibility**
  + Easily add new device types or extend functionality.
  + Modular architecture supports future smart home expansions.

### 

### **Future Enhancements**

* **GUI / Web App**
  + Develop a graphical interface for easier device control and monitoring.
* **Multi-Home / Multi-User Support**
  + Allow multiple homes or users to manage devices independently.
* **Real-Time Dashboards**
  + Visualize device status, usage statistics, and logs in real time.
* **Integration with Cloud Services**
  + Connect with platforms like Alexa, Google Home, or IoT hubs.
* **Notification System**
  + Alerts for device status changes or scheduled actions.
* **Integration with Payment Gateways** *(if applicable)*
  + For subscription-based device management or premium features.