# Smart Home IoT Automation Simulator

**PYTHON**

Md Rakibul Hasan

JR9RXP

Faculty of Informatics

Eötvös Loránd University

## Task Description:

In this assignment, we will need to develop a Python-based IoT simulator for a smart home automation system. The simulator should emulate the behaviour of various IoT devices commonly found in a smart home, such as smart lights, thermostats, and security cameras. We will also create a central automation system that manages these devices and build a monitoring dashboard to visualise and control the smart home. This assignment will help us apply our Python programming skills, including OOP, data handling, real-time data monitoring, and graphical user interfaces (GUIs).

## Objective:

1. Develop a Python-based IoT device simulator.
2. Implement basic data analytics and processing.
3. Create an eye-catching monitoring dashboard to display sensor data and analytics results.
4. Use Python data structures, classes, OOP objects, instance and class variables, OOP methods, pip, static methods, files & standard library, exception handling, modules, and packages

## Instructions for Running the Smart Home Automation System Simulation:

To successfully run the simulation for your Smart Home Automation System, follow these step-by-step instructions. Ensure that all necessary Python files (AutomationSystem.py, SmartLight.py, Thermostat.py, SecurityCamera.py, SmartHomeGUI.py, and main.py) are in the same directory.

### Prerequisites:

**Python Installation:** Ensure Python 3.x is installed on your system. You can download it from the official Python website.

**Library Dependencies:** Make sure all required libraries, especially Tkinter for the GUI, are installed. Tkinter usually comes pre-installed with Python. If it's not, you can install it using the command:

```
pip install tk
```

### Steps to Run the Simulation:

**Open Terminal or Command Prompt:** Navigate to the directory where your Python files are located.

**Run the Main Script:** Execute the main.py script which is the entry point of your simulation. You can do this by typing the following command in your terminal or command prompt:

```
python main.py
```

### Interacting with the Simulation:

The simulation should start, and the GUI window will open.

Through the GUI, you can monitor and control different IoT devices like Smart Lights, Thermostats, and Security Cameras.

Use the provided controls in the GUI to turn devices on/off, adjust settings like brightness or temperature, and observe the automated responses and changes in device states.

The console (terminal or command prompt) will also display log messages about device status and actions taken by the automation system.

Stop the Simulation: To stop the simulation, simply close the GUI window. The system will safely terminate the simulation and close the application.

## Class: SmartLight

**Description:**

Represents a smart light bulb with adjustable brightness and motion detection capabilities, typically used in smart home environments.

### Methods:

**__init__(self, device_id, automation_system):** Initializes the smart light with a device ID and a reference to the automation system.

**_update_status(self, status=None, brightness=None):** Private method to update the light's status and brightness. Updates status_var if set.

**turn_on(self):** Turns on the smart light to the default brightness level.

**turn_off(self):** Turns off the smart light.

**adjust_brightness(self, brightness):** Adjusts the smart light's brightness to a specified level.

**detect_motion(self):** Simulates motion detection and turns on the light if motion is detected.

**set_status_var(self, status_var):** Sets the status_var for GUI updates.

## Test Cases for SmartLight

**Initialization Test:**

Objective: Test the initialization of the SmartLight class.

Method: Create an instance of SmartLight with a specific device_id and automation_system.

Expected Result: The smart light should be initialized with the given device_id and associated with the provided automation_system.

**Brightness Adjustment Test:**

Objective: Test the adjustment of brightness levels.

Method: Adjust the brightness of a SmartLight instance to various levels.

Expected Result: The smart light's brightness should change as per the provided value.

**Motion Detection Test:**

Objective: Simulate motion detection and light response.

Method: Trigger the detect_motion method and check if the light turns on.

Expected Result: The smart light should turn on upon motion detection.

**Turn On/Off Test:**

Objective: Test the ability to turn the smart light on and off.

Method: Use the turn_on and turn_off methods to toggle the light's state.

Expected Result: The smart light should turn on and off as commanded.

## Class: Thermostat

**Description:**

Represents a smart thermostat device used for regulating temperature in a smart home environment.

### Methods:

**__init__(self, device_id, automation_system=None):** Initializes the thermostat with a device ID and optionally a reference to the automation system.

**_update_status_var(self):** Private method to update the status variable with the current status and temperature.

**toggle_thermostat(self):** Toggles the thermostat on or off.

**set_temperature(self, temperature):** Sets the thermostat's temperature, within the allowed range.

**increase_temperature(self):** Increases the thermostat's temperature by 1 degree, within the maximum limit.

**decrease_temperature(self):** Decreases the thermostat's temperature by 1 degree, within the minimum limit.

**set_status_var(self, status_var):** Sets the status_var for GUI updates.

## Test Cases for Thermostat

**Temperature Setting Test:**

Objective: Test setting the temperature within the allowable range.

Method: Set various temperatures on the thermostat and check if they are within the allowed range.

Expected Result: The thermostat should only accept temperatures within the defined range.

**Toggle Thermostat Test:**

Objective: Test the ability to toggle the thermostat on and off.

Method: Use the toggle_thermostat method to change the thermostat's state.

Expected Result: The thermostat should switch between on and off states as commanded.

**Temperature Adjustment Test:**

Objective: Test the increment and decrement functions for temperature.

Method: Increment and decrement the temperature using the increase_temperature and decrease_temperature methods.

Expected Result: The thermostat's temperature should adjust by 1 degree with each increment/decrement, staying within the allowed range.

**Initialization Test:**

Objective: Test the initialization of the Thermostat class.

Method: Create an instance of Thermostat with a specific device_id and optionally an automation_system.

Expected Result: The thermostat should be initialized with the given device_id and associated with the provided automation_system, if any.

## Class: SecurityCamera

**Description:** The SecurityCamera class represents a security camera device in the smart home automation system. It includes functionalities to manage the camera's status, recording, and infrared mode.

### Methods:

**__init__(self, device_id, automation_system):** Initializes the security camera with a device ID and a reference to the automation system.

**turn_on(self):** Activates the camera, sets its status to on, and starts motion detection.

**turn_off(self):** Deactivates the camera, stops recording, disables infrared, and sets its status to off.

**toggle_camera(self):** Toggles the camera's state between on and off.

**start_recording(self):** Starts video recording on the camera.

**stop_recording(self):** Stops video recording.

**enable_infrared(self):** Enables the infrared mode for low-light conditions.

**disable_infrared(self):** Disables the infrared mode.

**detect_motion(self):** Simulates motion detection functionality.

**_update_status_var(self, status):** Private method to update the status_var.

### Test Cases for SecurityCamera:

**Camera On/Off Toggle Test:**

Objective: To test the ability to toggle the camera's power state.

Method: Create an instance of SecurityCamera and use toggle_camera() to switch its state.

Expected Result: The camera should correctly switch between on and off states.

**Recording Functionality Test:**

Objective: To verify that the camera can start and stop recording.

Method: Turn on the camera and use start_recording() and stop_recording() methods.

Expected Result: The camera should start and stop recording as per the method calls.

**Infrared Mode Activation Test:**

Objective: To test the activation and deactivation of the infrared mode.

Method: Turn on the camera and use enable_infrared() and disable_infrared().

Expected Result: The infrared mode should be enabled and disabled as per the method calls.

**Status Update Verification Test:**

Objective: To ensure that the camera's status is updated correctly.

Method: Perform various operations (turn on, start recording, etc.) and check the status_var.

Expected Result: The status_var should reflect the current state of the camera accurately.

## Class: SmartHomeGUI

## Description:

SmartHomeGUI is a graphical user interface class built using Tkinter to interact with and monitor the smart home IoT automation system. This interface displays the status and allows control of the IoT devices in the system.

## Methods:

**__init__(self, automation_system):** Constructor that initializes the GUI, sets the window title and dimensions, configures styles, layout, and stdout redirection for logging.

**configure_styles(self):** Sets up the visual styles for different Tkinter components used in the GUI.

**configure_layout(self):** Defines the layout of the GUI components, including frames, buttons, and text areas.

**add_devices_to_frame(self):** Dynamically adds device control elements to the GUI based on the devices present in the automation_system.

**update_device_status(self, device_id, status):** Updates the display of a specific device's status in the GUI.

**run(self):** Starts the Tkinter event loop to run the GUI application.

## Test Cases for SmartHomeGUI

**GUI Initialization Test:**

Objective: To verify that the GUI initializes correctly with the given automation system.

Method: Create an instance of SmartHomeGUI with an AutomationSystem and call the run method.

Expected Result: The GUI should launch with the correct layout and style, displaying the status of all devices in the automation system.

**Device Status Update Test:**

Objective: To test the GUI's ability to update and display the status of devices.

Method: Change the status of a device in the automation system and verify that the GUI reflects this change.

Expected Result: The GUI should accurately display the updated status of the device.

**Layout and Style Configuration Test:**

Objective: To ensure that the GUI components are styled and laid out as defined.

Method: Inspect the GUI components for adherence to the style and layout configurations.

Expected Result: All GUI components should match the defined styles and layout configurations.

## Class: AutomationSystem

The AutomationSystem class represents the central control system for various IoT devices in a smart home environment. This system is responsible for managing devices, simulating device actions, and enforcing automation rules.

### Methods

**__init__(self):** Initializes the AutomationSystem with an empty list of devices.

**add_device(self, device):** Adds a new IoT device to the automation system. It verifies if the device has a 'device_id' attribute. The device is then appended to the devices list, and a reference to the AutomationSystem is set in the device.

**discover_devices(self):** Displays all discovered devices with their device IDs and types.

**run_simulation(self):** Simulates the automation system by processing actions for each device. It catches and reports any AutomationError.

**_process_device_actions(self, device):** Private method to process actions based on the type and status of the device. It calls specific handlers for SmartLight and Thermostat.

**_handle_smart_light_actions(self, light):** Private method to handle actions for SmartLight devices. Activates camera infrared if light's brightness is below 50.

**_handle_thermostat_actions(self):** Private method to handle actions for Thermostat devices. Starts camera recording if all lights are off.

**_are_all_lights_off(self):** Private helper method to check if all SmartLights are off.

**_start_camera_recording(self):** Private method to start recording on all active Security Cameras.

**_activate_camera_infrared(self):** Private method to enable infrared mode on all Security Cameras.

**check_and_start_recording(self):** Public method to check conditions and start camera recording if all lights and thermostats are off.

**_are_all_thermostats_off(self):** Private helper method to check if all Thermostats are off.

**_start_camera_recording_and_turn_on(self):** Private method to start recording and turn on the camera if the appropriate conditions are met.

### Test Cases

**Add Device Test:** Test adding a valid device with a 'device_id' to the system. Test adding a device without a 'device_id' and expect an AutomationError.

**Discover Devices Test:** Add multiple devices and check if discover_devices lists them correctly.

**Run Simulation Test:** Simulate with various devices in different states and verify appropriate actions are taken.

**Start Recording Conditions Test:** Verify that recording starts only when all lights and thermostats are off.

## Class: Main

The main.py file is responsible for initializing and running the IoT smart home automation system. It includes the setup of various IoT devices, the initiation of the automation system, and the execution of the simulation loop.

### Methods:

#### run_simulation(automation_system):

Runs the simulation loop for the automation system. It continuously triggers the automation system to run its simulation at regular intervals.

**Parameters:**

**automation_system:**

An instance of the AutomationSystem class.

**Functionality:**

In a loop, it calls the run_simulation() method of the passed AutomationSystem instance every 5 seconds. It handles keyboard interrupts and general exceptions, raising a SimulationError for any unexpected issues during the simulation.

### setup_devices(home_automation):

Sets up and adds IoT devices (like SmartLight, Thermostat, and SecurityCamera) to the automation system.

**Parameters:**

**home_automation:** An instance of the AutomationSystem class.

**Functionality:**

Creates instances of SmartLight, Thermostat, and SecurityCamera, initializes them with unique identifiers and the automation system instance, and adds them to the home_automation system.

## Test Cases

**Simulation Run Test:**

**Objective:** To ensure the simulation loop runs correctly and processes device actions.

**Method:** Initialize the automation system, setup devices, and call run_simulation() method.

**Expected Result:** The simulation should run continuously, processing device actions at regular intervals, until interrupted.

**Device Setup Test:**

**Objective:** To test the setup and addition of devices to the automation system.

**Method:** Call setup_devices() with an initialized AutomationSystem instance.

**Expected Result:** All devices (SmartLight, Thermostat, SecurityCamera) should be successfully created, initialized, and added to the automation system.

# Instructions for Using the Smart Home Automation System Dashboard

The Smart Home Automation System Dashboard is designed to provide an intuitive and interactive interface for monitoring and controlling the IoT devices in your smart home simulation. Here's how to navigate and use the various features of the dashboard:

**Opening the Dashboard**

The dashboard is part of the simulation and will automatically open when you run the main.py script. Ensure that the SmartHomeGUI.py file is in the same directory and properly linked to main.py.

## Dashboard Layout and Features

**Device List:**

The dashboard typically displays a list or grid of all connected IoT devices, such as Smart Lights, Thermostats, and Security Cameras.

Each device is represented with its name, status (on/off), and other specific attributes like temperature for thermostats or brightness for lights.

**Device Control:**

**Smart Lights:**

You can turn lights on or off using the provided toggle buttons.

Adjust the brightness of each light using a slider or input field.

**Thermostats:**

Toggle the thermostat on or off.

Set or adjust the temperature using a slider or input field.

**Security Cameras:**

View the status of each camera.

Activate or deactivate cameras.

**System Status and Logs:**

The dashboard might display a section for system logs or status updates, showing real-time information about what actions are being taken by the automation system.

**Real-time Updates:**

The dashboard updates in real-time, reflecting any changes in the state or settings of the devices.

## Interacting with Devices

**Toggle Devices:** Click on the toggle button next to each device to turn it on or off.

**Adjust Settings:** Use sliders or input fields to adjust device-specific settings like brightness or temperature.

**View Device Information:** Hover over or click on a device icon or name to view more detailed information about its current status and settings.

## Automation Features

The dashboard might also provide options to set up or modify automation rules (if implemented in your system).

You can observe how the system automatically adjusts device settings based on these rules or external triggers.

## Closing the Dashboard

To close the dashboard, simply close the GUI window. This action should also safely terminate the entire simulation.