# Smart Home IoT

Written and Implemented

By

Mohammed Efaz (ZTFK53)

3rd Semester | 11 November 2023

Course: Python

Faculty of Informatics

Eötvös Loránd University

## Contents

## Task

In this assignment, you 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. You 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 you apply your Python programming skills, including OOP, data handling, real-time data monitoring, and graphical user interfaces (GUIs).

**Part 1: IoT Device Emulation**. Device Classes: Create Python classes for each type of IoT device you want to simulate, such as SmartLight, Thermostat, and SecurityCamera. Each class should have attributes like device ID, status (on/off), and relevant properties (e.g., temperature for thermostats, brightness for lights, and security status for cameras). Device Behavior: Implement methods for each device class that allow for turning devices on/off and changing their properties. Simulate realistic behavior, such as gradual dimming for lights or setting temperature ranges for thermostats. Randomization: Include a randomization mechanism to simulate changing device states and properties over time.

**Part 2: Central Automation Syste**. Automation System Class: Create a central automation system class, e.g., AutomationSystem, responsible for managing and controlling all devices. It should provide methods for discovering devices, adding them to the system, and executing automation tasks. Simulation Loop: Implement a simulation loop that runs periodically (e.g., every few seconds) to trigger automation rules, update device states, and simulate device behaviours.

**Part 3: Documentation**. Documentation: Provide clear documentation for your code, including class descriptions, method explanations, and instructions on how to run the simulation and use the dashboard. Develop test cases to ensure that the simulator and automation system behave as expected. Test various scenarios, such as different automation rules and user interactions

**Part 4: Monitoring Dashboard**. Graphical User Interface (GUI): Create a GUI for monitoring and controlling the smart home system. You can use Python GUI libraries like Tkinter. The GUI should display the status and properties of each device, provide controls to interact with them, and visualize data. Real-time Data Monitoring: Display real-time data from the simulated devices on the dashboard. This includes temperature graphs for thermostats, motion detection status for cameras, and brightness levels for lights. User Interaction: Allow users to interact with devices through the GUI, such as toggling lights on/off, adjusting thermostat settings, and arming/disarming security cameras.

## Description of the task

The classes are created (apart from the main): `rubikGUI`, `rubikModel`, `rubikController`, `clockModel`. This follows the MVC (Model-View-Controller) design pattern which means that the code is reusable, maintainable, and easily editable (if need be, in the future).

## Description of the classes and methods

`securityCamera` – The camera class of the home system. It includes methods for managing the camera's operational status and security status.

- `__init__(self, deviceID, status, security_status)`: Constructor to initialize a `securityCamera` object.
- `set_security_status(self, security_status)`: Sets the `security_status` of the camera.
- `toggle_security_status(self)`: Toggles the `security_status` based on the current `status`. Prints said `status`.
- `turn_on(self)`: Turns the camera on and sets the `security_status` to 'secure'.
- `turn_off(self)`: Turns the camera of and sets the `security_status` to 'unsecure'

`smartLight` – The light class of the home system. It includes methods for managing the light's operational status and brightness level.

- `__init__(self, deviceID, status, brightness)`: Constructor to initialize a `smartLight` object.
- `set_brightness(self, brightness)`: Sets the brightness of the light, given it's on.
- `gradual_brightness(self)`: Gradually decreases the brightness by 5% if the light is on and brightness is above 50. Prints the current brightness level.
- `turn_on(self)`: Turns the light on, sets the `status` to 'on', and sets the brightness to 60%.
- `turn_off(self)`: Turns the light off, sets the `status` to 'off', and resets the brightness to 0%.

`thermoStat` – The thermostat class of the home system. It includes methods for managing the thermostat's operational status and temperature level.

- `__init__(self, deviceID, status, temperature)`: Constructor to initialize a `thermoStat` object.
- `set_temperature(self, temperature)`: Sets the thermostat's temperature to the specified value if the thermostat is on. If the thermostat is off, it prints a message indicating the thermostat is off.
- `turn_on(self)`: Turns the thermostat on, sets the `status` to 'on', and sets the temperature to a default of 20°C. Prints the status and the default temperature.
- `turn_off(self)`: Turns the light off, sets the `status` to 'off', and resets to 15°C. Prints the status.
- `randomise_temperature(self)`: Randomizes the thermostat's temperature between 10°C and 30°C and sets the thermostat to this new temperature.

`automationSystem` – The central automation class of the home systema centralised system to manage various smart devices in a home automation setup. It allows adding, removing, and managing devices: security cameras, smart lights, and thermostats.

- `__init__(self)`: Constructor to initialize an `automationSystem` object.
- `add_device (self, device)`: Adds a device from the `devices` list.
- `remove_device (self)`: Adds a device from the `devices` list.
- `get_device(self, deviceID)`: Searches for a device by `deviceID` in the `devices` list and returns it if found; otherwise, returns `None`.
- `execute_automation_jobs(self)`: Executes specific automation jobs for each device type in the system:
    - `For securityCamera:` If the camera is on, toggles its security status.
    - `For smartLight:` If the camera is on, toggles its security status.
    - `For thermoStat:` If the thermostat is on and its temperature is between 10°C and 30°C, randomises the temperature.

**`SmartHomeGUI`** – The GUI of the smart home system. It utilises the **`automationSystem`** class and individual device classes (**`securityCamera`**, **`smartLight`**, **`thermoStat`**) to provide a visual control panel for managing these devices.

- **`__init__(self, root)`**: Constructor that initializes the GUI elements and device instances. Sets up the control panel with device status labels, control sliders/buttons, and a logging area.
- **`create_device_status_labels(self)`**: Creates labels to display the status of each device in the automation system.
- **`add_Slider(frame, from_, to, command, initial_value)`**: Static method to add a slider control to the GUI for devices like thermostat and light.
- **`update_light_brightness(self, brightness)`**: Updates the brightness of the **`smartLight`** device and the corresponding label in the GUI.
- **`update_thermostat_temperature(self, temperature)`**: Updates the temperature of the **`thermoStat`** device and the corresponding label in the GUI.
- **`toggle_simulation(self)`**: Toggles the state of a simulation loop (if implemented) for the automation system.
- **`random_detect_motion(self)`**: Simulates random motion detection, updating the camera's motion status and potentially triggering light or camera actions.
- **`toggle_camera(self)`**: Toggles the state of the **`securityCamera`** between 'on' and 'off'.
- **`toggle_light(self)`**: Toggles the state of the **`smartLight`** between 'on' and 'off'.
- **`toggle_thermostat(self)`**: Toggles the state of the **`thermoStat`** between 'on' and 'off'.
- **`update_device_status_labels(self)`**: Updates the GUI labels to reflect the current status of all devices in the automation system.
- **`periodic_update(self)`**: Periodically executes automation jobs and updates the GUI. This function is called recursively at a set interval (e.g., every 1000 milliseconds).

## How To Run

### How To: Simulation

*Simulation Toggle Button*: In the GUI, there is a button labelled "Toggle Simulation." This button starts and stops the simulation. When you click it, it will either start or stop the simulation loop.

*Random Detect Motion Button*: This button simulates random motion detection by the front door camera. When clicked, it will trigger motion detection, and if motion is detected, the camera and light will be turned on automatically (the Camera's security status will be 'secure' and the light's brightness will be 60).

### How To: Dashboard

*Light Brightness Slider*: The slider labelled "Brightness" allows you to control the brightness of the smart light. You can drag the slider to increase or decrease the brightness level of the light.

*Thermostat Temperature Slider*: The slider labelled "Temperature (°C)" allows you to control the temperature setting of the thermostat. You can drag the slider to set the desired temperature in degrees Celsius.

*Toggle Camera Button*: The "Toggle Camera" button allows you to turn the security camera on or off. Clicking it will toggle the camera's status between "on" and "off."

*Toggle Light Button*: The "Toggle Light" button allows you to turn the smart light on or off. Clicking it will toggle the light's status between "on" and "off."

*Toggle Thermostat Button*: The "Toggle Thermostat" button allows you to turn the thermostat on or off. Clicking it will toggle the thermostat's status between "on" and "off."

*Random Detect Motion Button*: This button simulates random motion detection by the front door camera.

*Camera Motion Status Label*: This label displays the motion status of the front door camera. It will show "Motion: YES" (and subsequently show the status of security camera, which is secure) if motion is detected and "Motion: NO" if no motion is detected.

*Log Text Box*: The text box in the GUI displays log messages related to the simulation and device status changes. It provides information about what is happening in the simulation.

## Test Cases

### Test Case 1: Turning On the Simulation and Detecting Motion

1. Click the "Toggle Simulation" button to start the simulation.
2. Observe the log text box for simulation updates.
3. Click the "Random Detect Motion" button.
4. Check the "Camera Motion Status" label to see if it displays "Motion: NO."
5. Observe any changes in device statuses (e.g., camera and light).

### Test Case 2: Controlling the Smart Light

1. Start the simulation by clicking the "Toggle Simulation" button.
2. Use the "Brightness" slider to adjust the brightness of the smart light.
3. Observe the changes in the light's brightness on the GUI.
4. Toggle the light on and off using the "Toggle Light" button.
5. Observe the changes in the light's status on the GUI.

### Test Case 3: Adjusting the Thermostat

1. Start the simulation by clicking the "Toggle Simulation" button.
2. Use the "Temperature (°C)" slider to set the desired temperature on the thermostat.
3. Observe the changes in the thermostat's temperature setting on the GUI.
4. Toggle the thermostat on and off using the "Toggle Thermostat" button.
5. Observe the changes in the thermostat's status on the GUI.

### Test Case 4: Turning Off the Simulation

1. Start the simulation by clicking the "Toggle Simulation" button.
2. Observe the simulation running with device interactions.
3. Click the "Toggle Simulation" button again to stop the simulation.
4. Check if the simulation stops, and device interactions cease.
5. Observe any final status updates on the GUI.