Documentation

Python IOT Assignment

Shepelenko Mykhailo

DN36E8

25.10.2023

Assignment

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 (25 %) υ 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). υ 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. υ Randomization: Include a randomization mechanism to simulate changing device states and properties over time.

Part 2: Central Automation System (25 %) υ 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. υ Simulation Loop: Implement a simulation loop that runs periodically (e.g., every few seconds) to trigger automation rules, update device states, and simulate device behaviors.

Part 3: Documentation (30%) υ Documentation: Provide clear documentation for your code, including class descriptions, method explanations, and instructions on how to run the simulation and use the dashboard. υ 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 (20%) υ 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. υ 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. υ 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.

Class descriptions

AutomationSystem Class:

Represents the main system managing smart home devices.

- __init__(self):

  Initializes the system with an empty list of devices and a reference to the dashboard.

- discover_device(self, device):

  Adds a new device to the list of discovered devices.

- randomize_device_status(self):

  Randomly sets the status (on/off) for all devices.

- get_device_by_id(self, device_id):

  Retrieves a device from the list based on its ID.

- randomize_device_values(self):

  Randomly sets values for different types of devices, such as brightness and temperature.

Dashboard Class:

Represents the graphical user interface for interacting with the smart home devices.

- __init__(self, master, automation_system):

  Initializes the dashboard with a master window and an automation system.

- create_device_controls(self):

  Creates GUI controls for each discovered device.

- motion_camera(self, d):

  Handles motion camera functionality, toggling the device status and controlling a smart light.

- change_brightness(value, device):

  Static method to change the brightness of smart lights.

- set_temperature(value, device):

  Static method to set the temperature of thermostats.

- toggle_security_status(device):

  Static method to toggle the security status of security cameras.

- toggle_device_status(self, device):

  Toggles the status (on/off) of a generic device.

- update_device_status(self, device):

  Updates the displayed status of a device in the GUI.

- randomize_status(self):

  Randomly sets the status for all devices and updates the GUI.

- randomize_values(self):

  Randomly sets values for different types of devices and updates the GUI.

- update_slider_values(self):

  Updates slider values based on device states.

- rand(self):

  Initiates the randomization process for values and status.

- update_device_values(self):

  Updates the displayed values of devices in the GUI.

- run_simulation(self, count=0):

  Runs the simulation for a specified number of iterations, updating values and status.

Device Class:

Base class representing a generic smart home device.

- __init__(self, device_id):

  Initializes the device with a unique ID and sets the initial status to off.

SmartLight Class:

Represents a smart light device.

- __init__(self, device_id):

  Initializes the smart light with a unique ID and sets the initial brightness to 0.

- change_brightness(self, value):

  Changes the brightness of the smart light.

Thermostat Class:

Represents a thermostat device.

- __init__(self, device_id):

  Initializes the thermostat with a unique ID and sets the initial temperature to 20.

- set_temperature(self, value):

  Sets the temperature of the thermostat.

SecurityCamera Class:

Represents a security camera device.

- __init__(self, device_id):

  Initializes the security camera with a unique ID and sets the initial security status to off.

- arm(self):

  Arms the security camera.

- disarm(self):

  Disarms the security camera.

- toggle_security_status(self):

  Toggles the security status of the security camera.

MotionCamera Class:

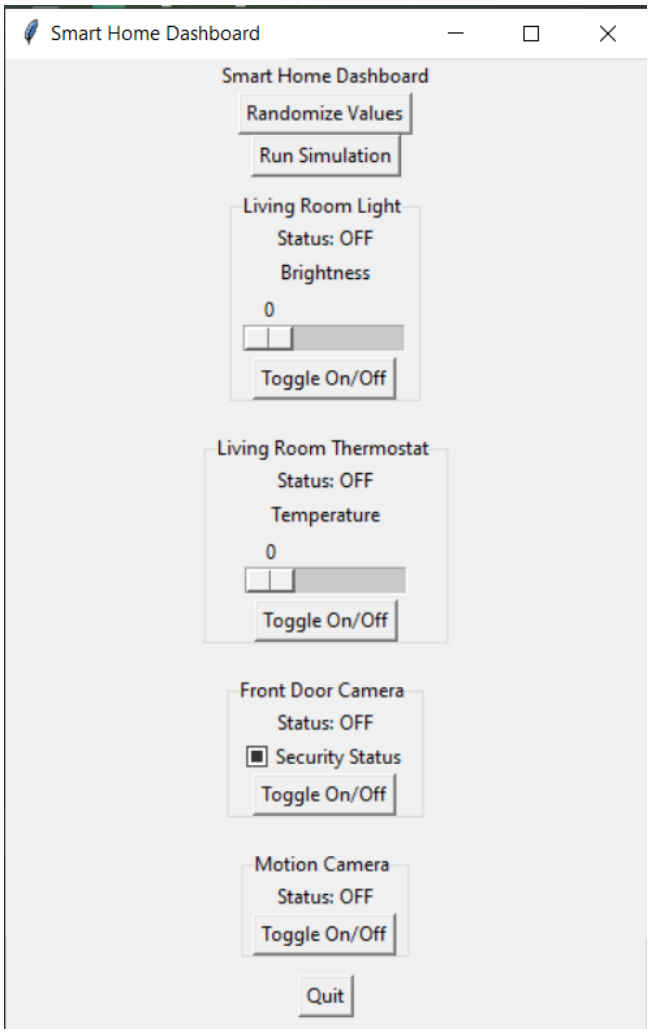
Represents a motion camera device, inheriting from the generic Device class.

- __init__(self, device_id):

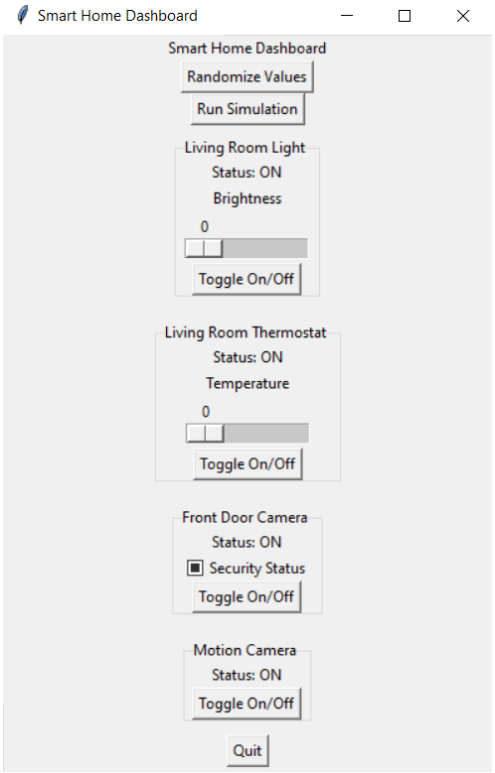
  Initializes the motion camera with a unique ID.

Tests

Initially the GUI looks like this:

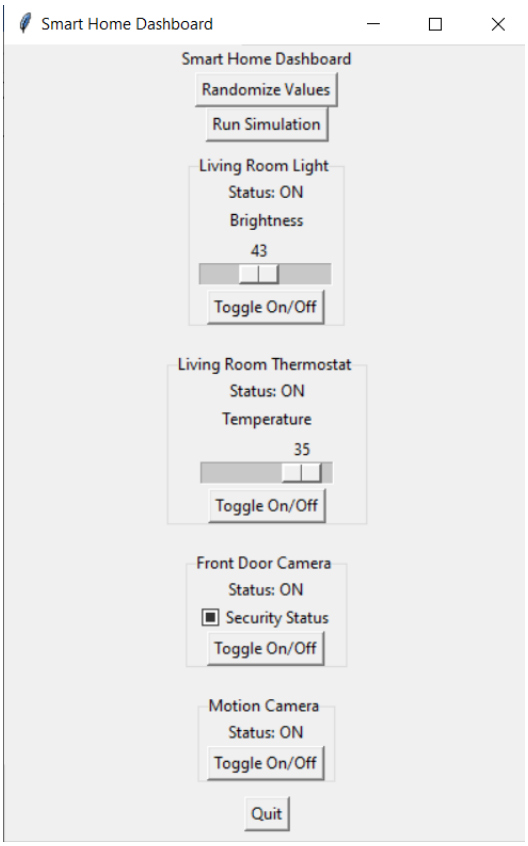


After the pressing button Toggle On/Off related to each device, the Status of it changes:
//the button Toggle On/Off pressed in each device block:



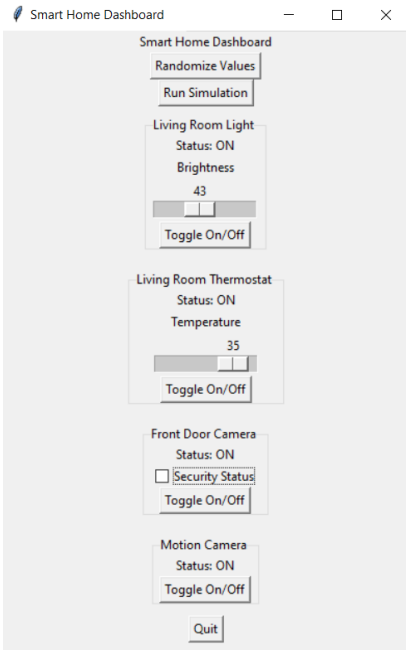
User can drag buttons on the sliders, it shows the current status of sliders:

//the brightness is set to 43%, the temperature is set to 35 degree:



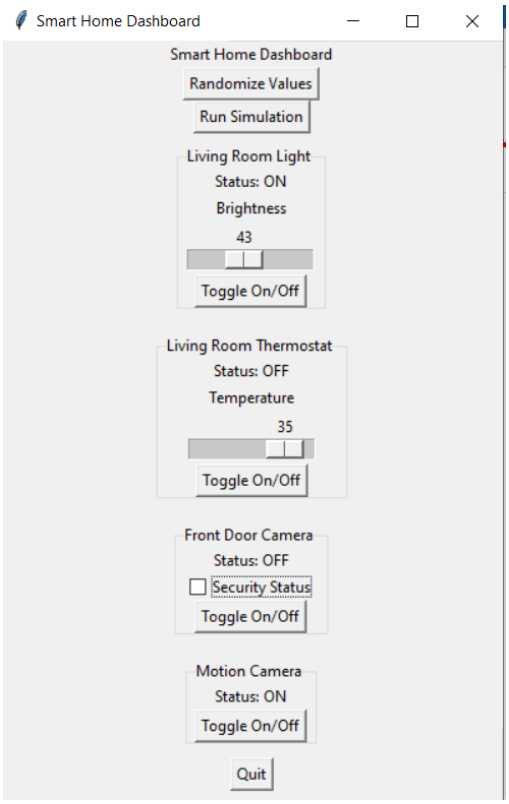
The Security Status button shows the status of front door camera:

//previously it was on, now it is off:



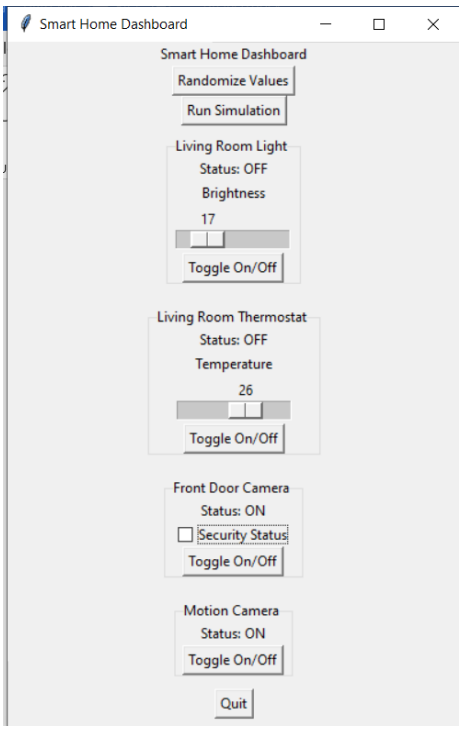
When the Motion Camera's status is on, it toggles on the Living room light, because it detects the motion in living room:

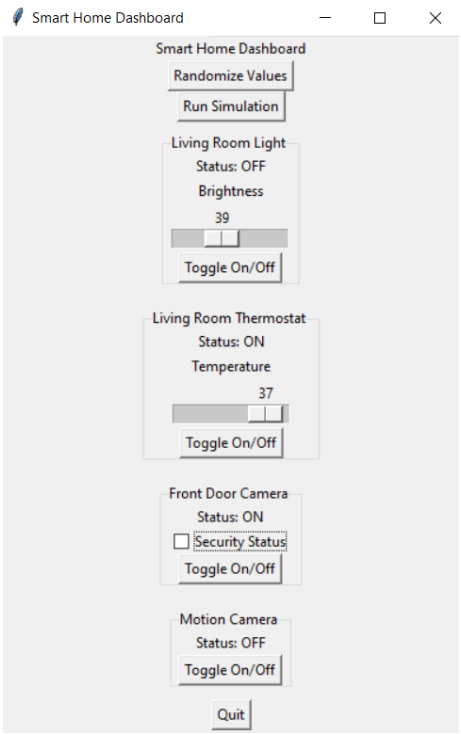
//Initially everything was off, I toggled on the motion camera (imagining that it detected the motion) and it turned on the living room light



Randomize Values button sets every status and value randomly:

//The button pressed 2 times





The Run Simulation button is simulating changes of every status and value randomly 5 times with 2 seconds interval.

The Quit button closes the program.