SOFTWARE TECHNICAL MANUAL TABLE OF CONTENT

1. FEATURES

a. Open Source

- SmartFlow is a free software, with open source code. This allows any user to customize the tool for their specific needs.

b. Multi Platform Compatible

- SmartFlow was developed on a microcontroller with a Linux operating system, but can easily be integrated into a Windows 10 or MacOS environment. Customization and Testing of SmartFlow can be done on a seperate machine and henceforth transferred onto the microcontroller. This allows for the user to work on the source code from their regular machines as opposed to directly on the microcontroller.

c. Lower Electricity Consumption

- The Machine Learning model will help the user save money every month on electricity consumption.

d. Increasing Physical and Virtual Security

- The Machine Learning model will help the user better analyze their smart home. This can include physical security as SmartFlow will detect and log if someone has physically or virtually entered the ecosystem.

2. SPECIFICATIONS (PRODUCT/DEVICE SPECIFICATIONS)

The SmartFlow monitoring system is focused on identifying potentially anomalous situations in a smart home based on the data recorded by smart devices installed in this home. This system can be divided into a hardware and software component. The former is represented by a Raspberry Pi 4 while the latter is a software suite that is installed on the Raspberry Pi. These software components perform several key tasks; First, collection of data from the smart home sensors. Second, parsing data from multiple sources into a standardized format. Third, assessing the security state of this smart home based on said data. Finally, visually displaying the security state of your smart home through a Graphics User Interface (GUI) dashboard. Collection of data from multiple sources is completed by OpenHab, which is an open-source offering that ensures compatibility with a wide range of smart devices. As for parsing the data, this is completed by a proprietary solution developed by the SmartFlow team. Moving onto analyzing the security state, this is completed by a proprietary machine learning model that is designed to monitor and adapt to the user's living habits. Once this "learning" process is complete, the model is then fed the output of the parsing process to generate a binary 0 or 1 value that indicates whether a recent event is anomalous or not. 0 signifies that the information is non-anomalous while 1 indicates that the event is anomalous. The GUI is generated through an in-house script that was developed using the Plotly library, written in the Python language.

In terms of limitations, the model is only ever able to predict an anomalous situation based on a single event. That is, it cannot determine whether a sequence of events together form an anomalous period, only whether one or more single events within that group of events is anomalous or not.

3. PART NAMES AND THEIR FUNCTIONS

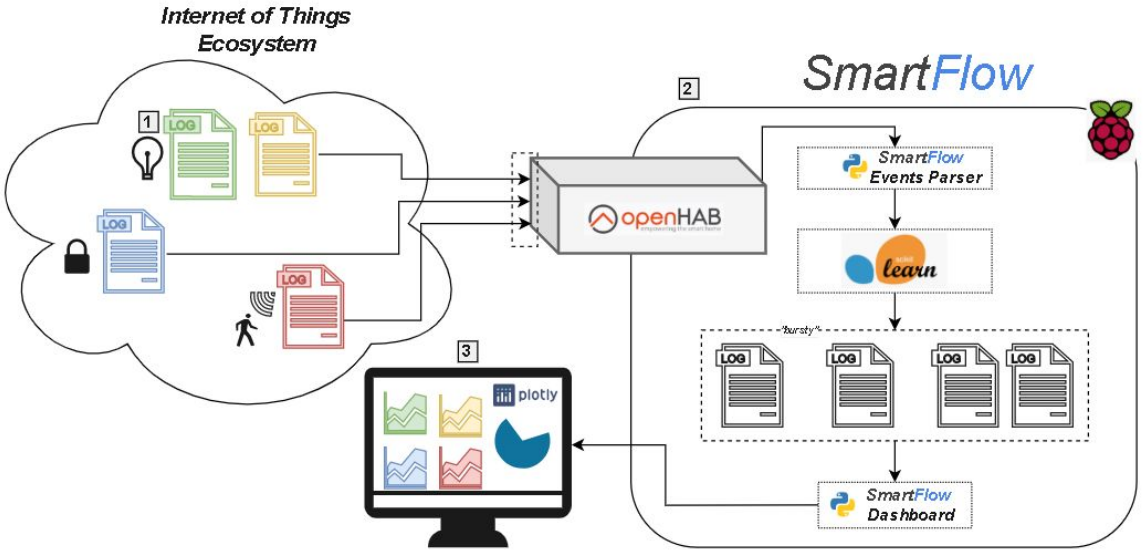


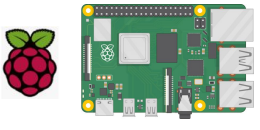
Figure 1: SmartFlow Parts

1 **Internet of Things Ecosystem**: the entities within the network produce the logs of the smart home. These logs are then fed into SmartFlow for further analysis.

- Philips Hue Smart Lights and Motion Sensors
- Samsung SmartThings Contact Sensors

2 **Microcontroller**: The Raspberry Pi is used for hosting all the software required for SmartFlow. This includes hosting OpenHab (smart home hub), a custom Python parsing script (SmartFlow Events Parser), the SmartFlow Machine Learning model and finally a customer data visualization Python program.

- Raspberry Pi 4



3 **Visualization**: The output of SmartFlow is displayed on a monitor via a browser, running on localhost port 8050.

- A monitor used for visualizing the SmartFlow dashboard 

4. BLOCK DIAGRAM/FLOW CHART DESCRIPTION

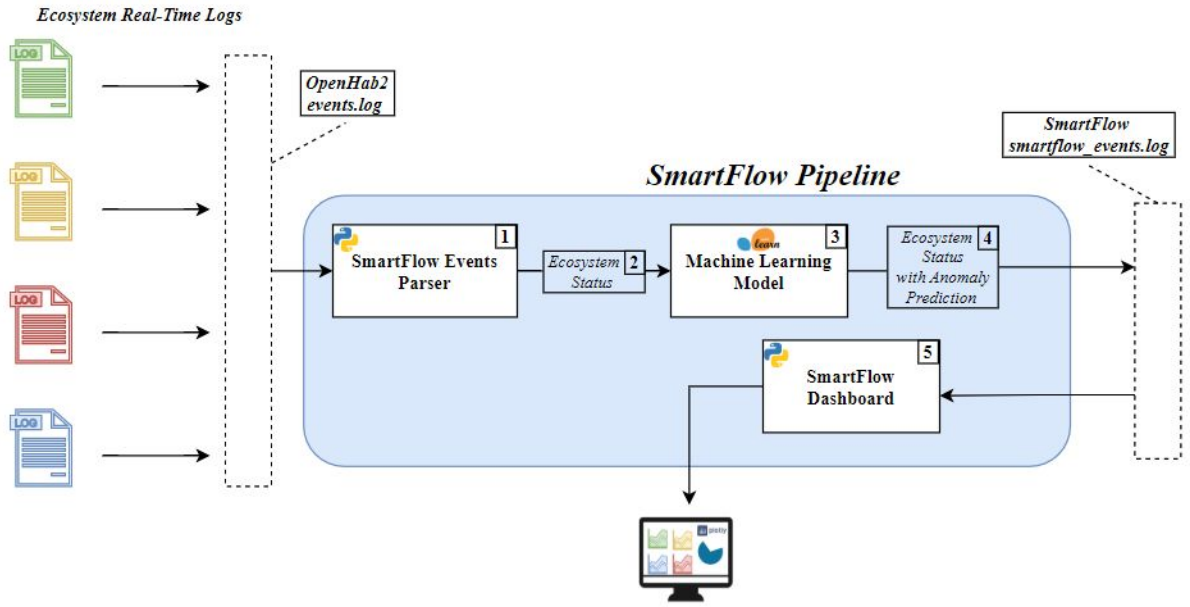


Figure 2: SmartFlow Block diagram

1 **SmartFlow Events Parser**: Acting as the ingress node to the SmartFlow pipeline, the parser reads raw logs from the OpenHab2 events.log file. Each log is processed and if necessary, passed along to the next stage of the pipeline.

2 **Ecosystem Status**: The output of the Events Parser is a Python dictionary containing a status (0 or 1) for each device in the smart home.

3 **Machine Learning Model**: The Ecosystem Status is fed into the SmartFlow Machine Learning Model to predict if the smart home is in an anomalous state. The anomalous prediction will be appended to the Ecosystem Status. The dictionary is then sent to the next stage of the pipeline.

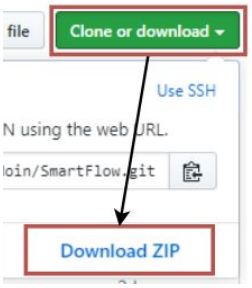
4 **Ecosystem Status with Anomaly Prediction**: The output of the Machine Learning Model is a Python dictionary which is logged in the smartflow_events.log file.

5 **SmartFlow Dashboard**: New logs appended to the smartflow_events.log file are immediately piped to the Dashboard. The dictionary obtained containing the status of the smart home is displayed via graphs. The Dashboard is the egress node of the pipeline.

5. FOR SYSTEM ADMINISTRATORS

❖ **Installation**: Explains how to download and install SmartFlow. It is assumed the System Administrator has all the prerequisites for the program and has the Anaconda virtual environment activated. Consult the user manual for all prerequisites.

➢ Via Download:



1. https://github.com/fjodoin/SmartFlow
2. Bring the cursor to the "Clone or download" drop down menu and select "Download Zip"
3. Extract the SmartFlow-master.zip to i.e. your Desktop
4. Navigate to the SmartFlow-master/SmartFlow directory and install required Python3 libraries with the install command:
    - *[PATH_to_dir]>pip3 install -r requirements.txt*

➢ Via Github:

1. Navigate to i.e. your Desktop, create a new folder which will contain the SmartFlow software.
2. Initialize the new folder as a git repository using the "git init" command:
    - *[PATH_to_dir]>git init*
3. Clone the SmartFlow repository using the "git clone" command:
    - *[PATH_to_dir]>git clone https://github.com/fjodoin/SmartFlow.git*
4. Pull from the SmartFlow repository master using the "git pull" command:
    - *[PATH_to_dir]>git pull*
5. install required Python3 libraries with the install command:
    - *[PATH_to_dir]>pip3 install -r requirements.txt*

❖ **Configuration**: After installing SmartFlow, it is necessary to configure paths used according to the user's machine. For this explanation, it is assumed the System Admin installed SmartFlow on their Desktop. It is assumed the System Administrator has all the prerequisites from the User Manual and has installed a Linux Bash Shell.

➢ **events_monitoring.sh**, line 3 (Figure 4)

1. Windows 10:

- *tail -n0 -f C:\openHAB2\userdata\logs\events.log | python3 C:\Users\[username]\Desktop\SmartFlow\events_parser.py*

2. MacOS: Assuming the user installed OpenHab2 on the Desktop

- *tail -n0 -f /Users/[username]/openhab2/userdata/logs/events.log | python3 /Users/[username]/Desktop/SmartFlow/events_parser.py*

3. Linux:

- *tail -n0 -f /var/log/openhab2/events.log | python3 /home/[username]/Desktop/SmartFlow/events_parser.py*

➢ **smartflow_dashboard.sh**, line 3 (Figure 8)

1. Windows 10:

- *tail -n0 -f C:\Users\[username]\Desktop\SmartFlow\smartflow_events.log | python3 C:\Users\[username]\Desktop\SmartFlow\dashboard.py*

2. MacOS: Assuming the user installed OpenHab2 on the Desktop

- *tail -n0 -f /Users/[username]/Desktop/SmartFlow/smartflow_events.log | python3 /Users/[username]/Desktop/SmartFlow/dashboard.py*

3. Linux (recommended):

- *tail -n0 -f /home/[username]/Desktop/SmartFlow/smartflow_events.log | python3 /home/[username]/Desktop/SmartFlow/dashboard.py*

❖ **System Administrative**:

➢ It is recommended to use the Linux Raspbian operating system as SmartFlow was designed for being used on a microcontroller. It is, nevertheless, compatible with Windows and MacOS and can be hosted on these operating systems. If Windows is the hosting operating system, the System Administrator must install a Linux Bash Shell and run all commands there as a portion of SmartFlow relies on Bash scripting.

➢ If any issues are discovered, please communicate the bug and the context in which it appeared on the Github page https://github.com/fjodoin/SmartFlow/issues

❖ **Upgrading**:

➢ It is recommended to keep the hosting machine update to date. This includes all the software as well. As the project is available on Github, two simple commands will verify for updates and install them if necessary. Navigate to the SmartFlow directory on the host machine and do the following commands:

1. *[PATH_to_dir]>git pull*
2. *[PATH_to_dir]>pip3 install -r requirements.txt*

- If the first command doesn't pull new updates, step 2 can be skipped

6. FOR DEVELOPERS

- **Architecture**: The pipeline is separated into two sections; the ingress node being the Parser and the egress node being the Dashboard
  - SmartFlow Events Parser (*events_parser.py*)
    - Module responsible for parsing incoming data shipped from the *events_monitoring.sh* Bash script. The *events_parser.py* module is launched inside the Bash script.
      - Lines 1 to 148 consists of Python imports and for variable, function and class declaration. The *class OpenhabAgent* is used for HTTP requests to the OpenHab2 smart home hub. It is used for obtaining initial temperatures and for synchronizing lights when the tenant changes rooms.
      - Lines 149 to 174 assign unique IDs of known devices within the ecosystem to variables and the temperature of each room is obtained. An analysis is done on the smart home and the results are sent to the Dashboard (by logging results to *smartflow_events.log*).
      - Lines 175 to 285 are the Main Loop: interrupt driven by line 179: *data = sys.stdin.readline()*. The system will wait at this line for a new log from *events_monitoring.sh* to be piped. Regular expression parsing is applied on the new log to get the unique ID, status, date and time. An analysis is done on the smart home and the results are sent to the Dashboard (by logging results to *smartflow_events.log*).

- SmartFlow Dashboard (*dashboard.py*)
  - Module responsible for displaying the ecosystem status from new logs added to *smartflow_events.log*. The *dashboard.py* module is launched inside the *smartflow_dashboard.sh* Bash script.
    - Lines 1 to 16 are for Python imports.
    - Lines 17 to 36 declare the ecosystem status dictionary which will be used as a global variable to keep the current status available throughout the module.
    - Line 37 to 61 declare, instantiate and start the *EventAgentThread* class. This thread is responsible for obtaining piped data from the *smartflow_dashboard.sh* script; it will wait on line 46 (*data = sys.stdin.readline()*) until a new log from *smartflow_events.log* is obtained. The ecosystem status global dictionary is then updated.
    - Line 62 to 126 consists of variable declarations used for the dashboard. Each device in the ecosystem has its own list of values, up to a certain length (i.e. 20 most recent values). These values are ordered in a FIFO list; when the 21st value is added to the list, the 1st value added is removed, hence keeping the 20 most recent readings. All these individual lists are kept in a Python dictionary known as *data*.
    - Lines 127 to 529 are used for the Web-Based User Interface Dashboard, leveraging the Plotly libraries. Line 529 launches the application, making it available on *localhost:8050*.

- **Database Layout**: SmartFlow keeps a log file from the ecosystem events. The file must be rotated and archived manually so that it does not grow too large. This file can be used to further train the Machine Learning Model integrated in the ecosystem, therefore it is the user's job to decide how much / what data to keep. The log file is *smartflow_events.log*, where each entry is formatted as a json string. The example below has a newline for each dictionary entry, which is not the case when found in the log file.

  {"date": "2020-03-23",
  "time": "17:13:18.792",
  "kitchen_light": 1,
  "kitchen_motion_sensor": 0,
  "kitchen_door_sensor": 0,
  "kitchen_temperature": "19.0",
  "office_light": 0,
  "office_motion_sensor": 0,
  "office_temperature": "19.74",
  "living_room_light": 0,
  "living_room_motion_sensor": 0,
  "living_room_door_sensor": 0,
  "living_room_temperature":
  "17.77", "bedroom_light": 0,
  "bedroom_motion_sensor": 0,
  "bedroom_temperature": "17.6",
  "smartflow_status": 0}

  Whenever the user would like to search for anomalies in their ecosystem, they can simply search the log file for *"smartflow_status": 1* and analyze the entire ecosystem status at that timestamp.

- **Developing Extensions**: As SmartFlow is open-source, the source code can be modified to accommodate for any ecosystem. There are sections which must be modified accordingly. The following steps will allow any developer to customize SmartFlow for their unique environment:

  1. The **Installation** and **Configuration** steps from the System Administrators section above are assumed completed.

2. The developer must know how many rooms and devices are part of their ecosystem.
3. The Machine Learning Model must be trained accordingly
4. The Source Code is separated into two section: Events Parser and Dashboard:
    - *events_parser.py:*
        a. Line 10 and 11 must point to the OpenHab2 installation. If the service is running on a different IP and/or Port, it is essential to adjust them here.
        b. Line 51 to 71 consists of the Ecosystem Status which is given to the Machine Learning Model to analyze. The *smartflow_list* on line 54 MUST be identical to the list used to train the model from step 3.
        c. Line 74 to 92 contains the Python dictionary which will be logged after analysis is complete by the model. This dictionary is initialized with all devices being off. It must contain all devices found in the ecosystem.
        d. Line 94 to 147 are used to synchronize the lights in the smart home. The developer must adjust this function according to the smart light found in their ecosystem.
        e. Line 151 to 170 hardcodes the unique IDs of each device to a variable and also obtains the current temperature for each room. Unique IDs can be found in the OpenHab2 panel or from raw logs in the OpenHab2 *events.log* file. Every device must be found in this section, along with their unique IDs.

f. Line 192 to 285 (Main Loop) must be adjusted according to the device found in the ecosystem. Any device added / removed from step e above must be added / removed from the Main Loop.

- *dashboard.py*
    a. Line 18 to 34 should be adjusted accordingly to the ecosystem devices, similar to step c from the *events_parser.py* above.
    b. Lines 67 to 122 must also be adjusted according to the ecosystem devices.
    c. Line 127 to 529 are for the Web-based Dashboard UI. The developer must know how many rooms are in their smart home. Each room will have its own subsection. The description that ensues is identical for all rooms in the ecosystem. The description uses the Kitchen as an example:
        i. Add the room into the HTML divider, similar to line 132. The room will require its own unique ID.
        ii. Refer to lines 146 to 229: A call back function must be created for each room in order to render new data in near real-time on the dashboard.
        iii. Lines 156 to 161 append the latest value of the devices found in the room.
        iv. Lines 163 to 175 creates a container for the graphs required for the room.

v. Lines 177 to 229 populate the graphs with the most recent values (depending on their length).

- **Web Access**:
  - SmartFlow Dashboard is accessible, by default, at localhost:8050
  - OpenHab2 is accessible, by default, at localhost:8080
  - SmartFlow is focused on being secure, therefore all services are run locally, never putting Ecosystem Logs or communication on the internet.

7. INSTALLATION REQUIREMENTS

- As SmartFlow is a software, a suitable microcontroller (or computer) must host the program and the ecosystem peripherals.
  - Microcontroller / Computer Hardware:
    1. RAM: Minimum [2GB], Recommended [4GB]
    2. Processor: [64-bit System on a Chip @ 1.5GHz]
    3. Storage: Minimum [128GB Secure Digital Card]
    4. Gigabit Ethernet Port x1
  - Peripherals Hardware:
    1. Keyboard
    2. Mouse
    3. Monitor
    4. Internet Connection (for updating Ecosystem Hardware)
    5. Ecosystem IoT Devices
    6. Gigabit Ethernet wire x(1 + number of IoT hubs)
  - Software:
    1. Python: Minimum [3.6]
    2. Bash
    3. IF WINDOWS 10: Linux Bash Terminal
    4. Python Libraries:
       a. Anaconda
       b. Plotly
       c. joblib

APPENDIX

A. SYSTEM BLOCK DIAGRAM

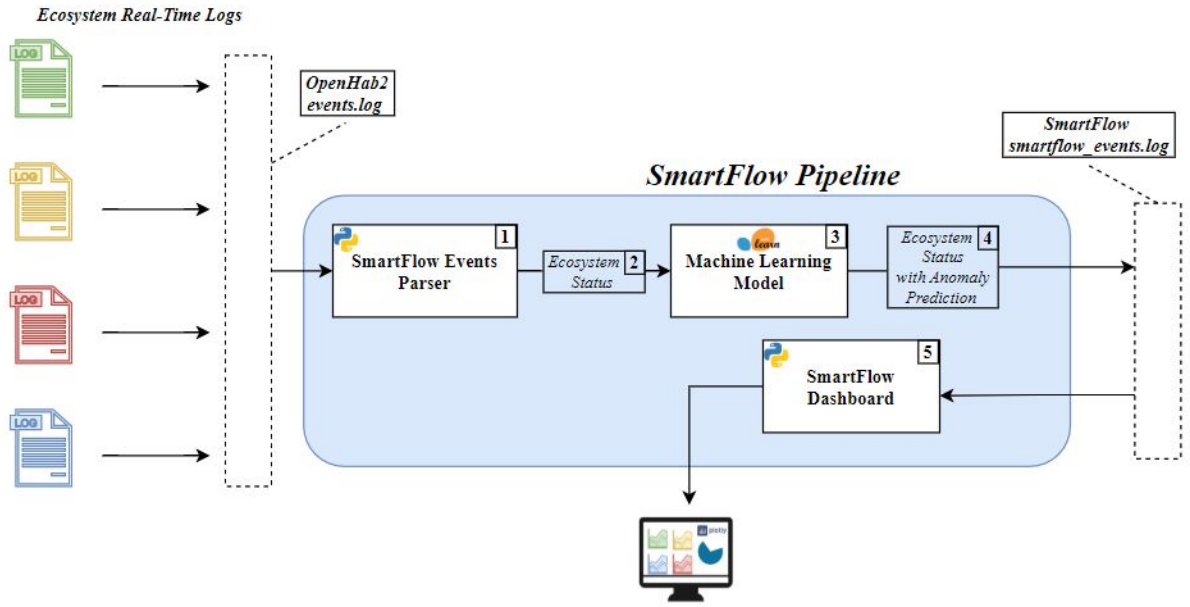


Figure 3: SmartFlow Block diagram

B. FUNCTIONS AND CLASSES

❖ Events Monitoring (*events_monitoring.sh*)

1. Script responsible for listening to the events.log file with *tail -n0 -f* Bash command. When a new log is captured, it is piped (|) into the python3 events_parser.py module.

```
#!/usr/bin/env bash
echo [SMARTFLOW] openhab2/events.log monitoring scripted launched [...]
tail -n0 -f /var/log/openhab2/events.log | python3 /home/corelee/Desktop/SmartFlow/events_parser.py
```

Figure 4: events_monitoring.sh script

❖ Events Parser (*events_parser.py*)

1. ***log_system_status****:* Used to log the most recent ecosystem status to the *smartflow_events.log* file.

```
def log_system_status(smartflow_status):
    """"""
    # for key, value in smartflow_status.items():
    #   print(key,value)
    log_info_string = json.dumps(smartflow_status)
    with open("smartflow_events.log", "a") as log_file:
        log_file.write(log_info_string + "\n")
```

Figure 5: events_parser.py logging function

2. ***analyze_smart_home:*** Function used for obtaining a Machine Learning prediction: the *smartflow_list* is given to *joblib.model.predict(smartflow_list)* which will in turn output the status: 1 for anomalous and 0 for normal environment.

```
def analyze_smart_home(smart_flow_dict):
    """""
    s_flow_dict = smart_flow_dict
    smartflow_list = [[
        0,                                          # Time
        s_flow_dict['kitchen_light'],               # Kitchen Light(M015)
        s_flow_dict['kitchen_motion_sensor'],       # Kitchen Motion(M019)
        s_flow_dict['living_room_motion_sensor'],   # Living Room Motion(M020)
        s_flow_dict['bedroom_motion_sensor'],       # Bedroom Motion(M007)
        s_flow_dict['office_motion_sensor'],        # Office Motion(M027)
        s_flow_dict['living_room_door_sensor'],     # Living Room Door(D001)
        s_flow_dict['kitchen_door_sensor'],         # Kitchen Door(D002)
        s_flow_dict['living_room_light'],           # Living Room Light(M013)
        s_flow_dict['office_light'],                # Office Light(M026)
        s_flow_dict['bedroom_light']                # Bedroom Light(M005)
    ]]

    smartflow_status = joblib_model.predict(smartflow_list)
    print(f'Status of Smart Home: {smartflow_status}: {smartflow_list}')
    s_flow_dict.update({"smartflow_status": str(smartflow_status[0])})
    log_system_status(smartflow_dict)
```

Figure 6: events_parser.py machine learning prediction function

3. ***OpenhabAgent:*** Class used for communicating with OpenHab2: smart home hub software. Custom functions within this class can be changed and adjusted depending on the developer and the ecosystem.

```
class OpenhabAgent:
    def __init__(self, ip='127.0.0.1', port=8080):
        self.base_url = "http://localhost:8080/rest/items"
        self.header = {'accept': 'application/json'}
        self.params = {'text': ""}
        self.post = 'curl -X POST --header "Content-type: text/plain" --header "Accept: application/json" -d'
        self.get = 'curl -X GET --header "Accept: application/json '
        try:
            # NOTE: openHAB api GET responds normally with python requests lib
            openhab_response = requests.get(self.base_url, headers=self.header, params=self.params)
            print("Connection established with openHAB")
        except:
            print("Failed connection to openHAB ....")
            sys.exit()

    # NOTE: MUST use curl for openHAB api POST as python requests lib does NOT trigger physical light change; although it does trigger an event...
    def on(self, item):
        # print("TURNING ON: " + item)
        os.system(self.post + ' "ON" "' + self.base_url+item)

    def off(self, item):
        # print("TURNING OFF: " + item)
        os.system(self.post + ' "OFF" "' + self.base_url+item)
    def get_temp(self, item, room):
        openhab_response = requests.get(self.base_url+item, headers=self.header, params=self.params)
        # openhab_response = json.loads(openhab_response)
        response_dict = json.loads(openhab_response.content.decode())
        # print(room, response_dict['state'][:5])
        #     log = re.match(r'(.*) - (.*)', data)
        room_temperature = re.match(r'(.*) .*', response_dict['state'])
        # print(room_temperature)
        smartflow_dict[room] = room_temperature.group(1).split()[0]
```

Figure 7: events_parser.py OpenHabAgent class

❖ **Smartflow Dashboard (*smartflow_dashboard.sh*)**

1. Script responsible for listening to the smartflow_events.log file with *tail -n0 -f* Bash command. When a new log is captured, it is piped (|) into the python3 dashboard.py module.

```
#!/usr/bin/env bash
echo [SMARTFLOW] SmartFlow/smartflow_events.log monitoring scripted launched [...]
tail -n0 -f /home/corelee/Desktop/SmartFlow/smartflow_events.log | python3 /home/corelee/Desktop/SmartFlow/dashboard.py
```

Figure 8: smartflow_dashboard.sh script

❖ **Dashboard (*dashboard.py*)**

1. ***EventAgentThread:*** Thread Class used for listening to incoming logs from smartflow_dashboard.sh script.

```
class EventAgentThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        counter_check = 0
        global ecosystem_status
        while True:
            try:
                # print("waiting...")
                data = sys.stdin.readline()
            except KeyboardInterrupt:
                break
            if not data:
                break
            data_dict = json.loads(data)
            # print(data_dict)
            counter_check += 1
            ecosystem_status = data_dict
            # print(counter_check)
            # print(ecosystem_status)

event_agent_thread = EventAgentThread()
event_agent_thread.start()
```

Figure 9: dashboard.py EventAgentThread Class

2. ***@app.callback & update_graph_live:*** The callback function is called for rendering purposes. The update graph live function will update all data to the most recent entries obtained from the EventAgentThread Class. Found for every graph displayed on the dashboard.

```
# KITCHEN GRAPH
@app.callback(Output('kitchen-graph', 'figure'),
              [Input('interval-component', 'n_intervals')])
def update_graph_live(n):
    t = datetime.datetime.now()
    # Synchronize TIME
    times.append(t)
    temperature_times.append(t)
    overview_times.append(t)

    # Room 1; KITCHEN
    # print(ecosystem_status)
    kitchen_light.append(ecosystem_status['kitchen_light'])
    kitchen_motion_sensor.append(ecosystem_status['kitchen_motion_sensor'])
    kitchen_door_sensor.append(ecosystem_status['kitchen_door_sensor'])
    kitchen_temperature.append(str(ecosystem_status['kitchen_temperature']) + "°C")

    # Create the graph with subplots
    fig = plotly.subplots.make_subplots(...
    fig['layout']['margin'] = {
        'l': 100, 'r': 10, 'b': 0, 't': 50
    }
    fig['layout']['legend'] = {'orientation': 'v',...

    # Room 1; KITCHEN
    fig.append_trace({...
    }, 1, 2)
    fig.append_trace({...
    }, 1, 2)
    fig.append_trace({...
    }, 1, 2)
    fig.append_trace({...
    }, 1, 7)

    fig.update_layout(title_text="[---KITCHEN---]", font=dict(family='Courier New, monospace',color='#1A1A1A'),...
    return fig
```

Figure 10: dashboard.py Plotly Rendering and Updating function

C. **BOM**

- Refer to *smartflow_bom.xls* included in SmartFlow submission package