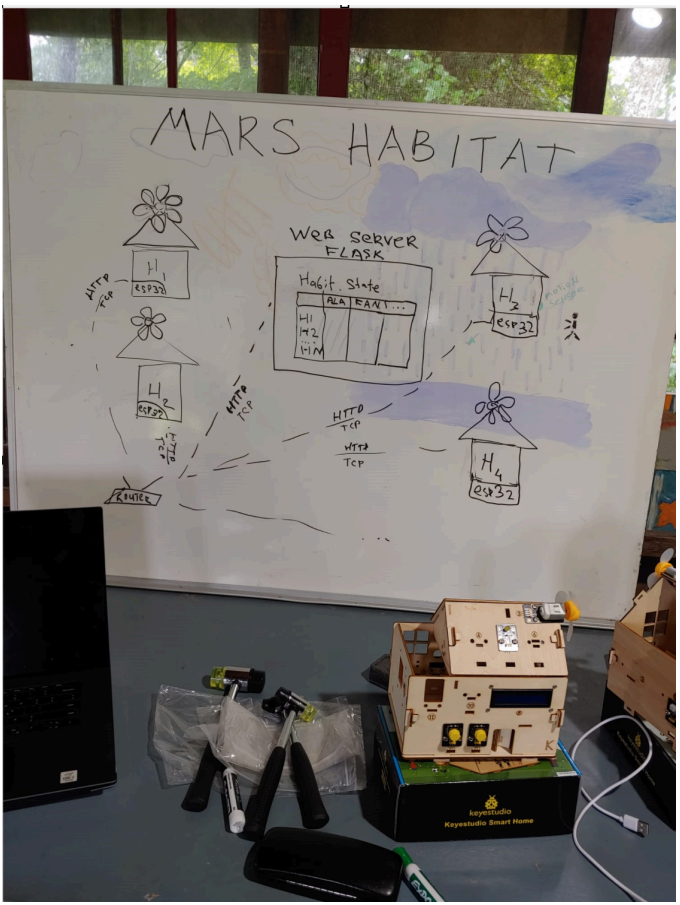
IoT and Web Dev

Day 1



What does it take to build a smart house? Smart city? Smart planet?

What are components of the smart house?

What is IoT device? Is dusk-to-dawn light bulb an IoT device?

What is a sensor? Actuator?

What is WiFi? 3g? 5g?

What is a server?

<u>Day 1:</u>

End-to-End demo, Key concepts, WiFi hello world test

<u>Day 2:</u>

Micropython examples: buttons, polling for button(s) state(s), LED, fading LED (PWM)

<u>Day 3:</u>

Buzzer, Noise vs Music, how to program music: tempo, rhythm, tone

Build House

<u>Day 4:</u>

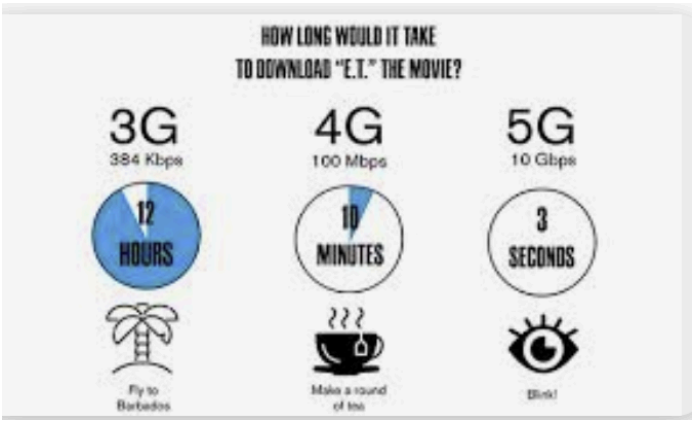
Web Server, how Internet Works: Internet Protocols, DNS, HTML, javascript, async communication,

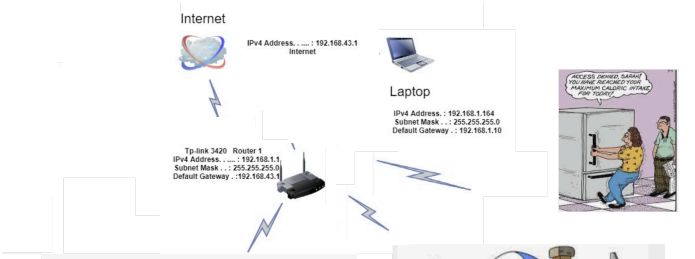
Hello World Examples, REST API

<u>Day 5:</u>

Connect All together

WiFi





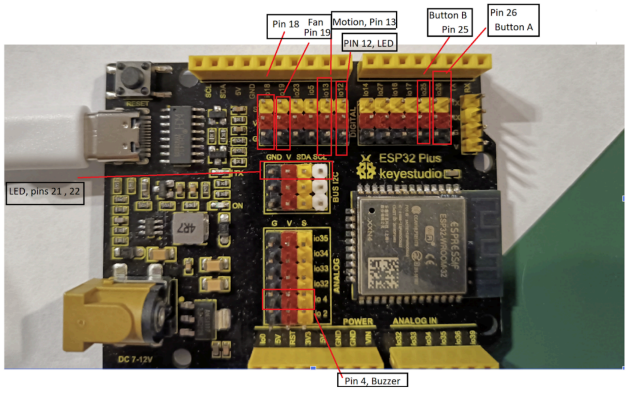
3g:

- Relies on large cell towers spaced far apart.
- Uses frequencies in the 800 MHz to 2.1 GHz range.
- Core network based on circuit-switched technology (for voice) and packet-switched technology (for data).

5g:

- Requires a dense network of small cells due to higher frequencies (up to 100 GHz), which have shorter ranges.
- Utilizes millimeter waves, which provide higher speeds but are easily obstructed by buildings and trees.
- Core network is entirely packet-switched and IP-based, integrating with cloud technologies and edge computing.

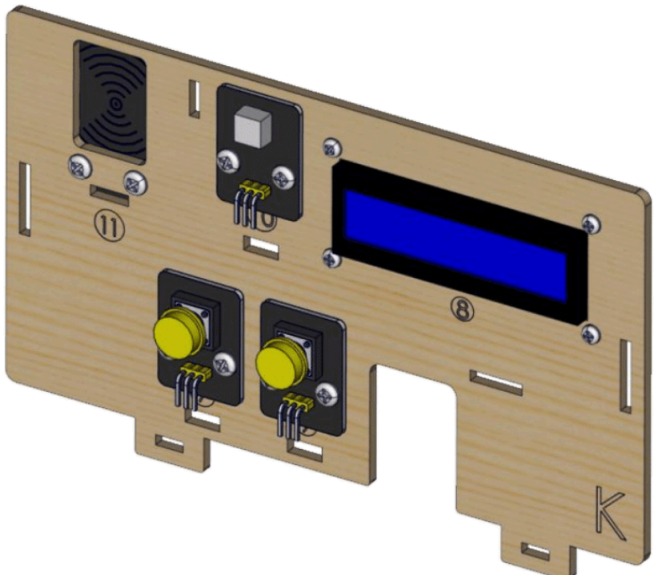
Hello world: server / app.py



ESP32 microcontroller and a set of sensors, actuators and other cool peripherals like buzzers, LCD displays, buttons and LED lights

Day 2

Micropython examples: buttons, polling for button(s) state(s), LED, fading LED (PWM)

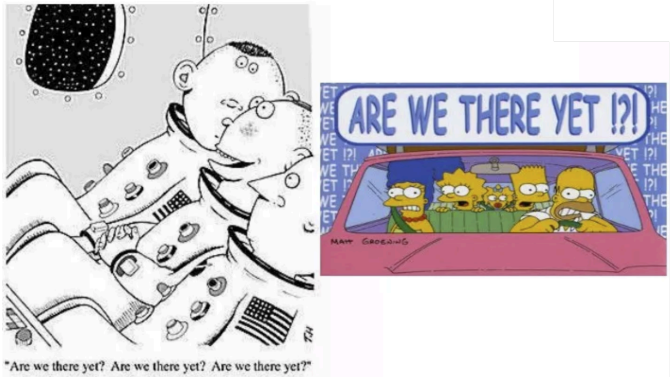


What is a button (sensor? actuator?)

`button1 = Pin(26, Pin.IN, Pin.PULL_UP)`: This line sets up a pin on the microcontroller to read input from a button.

- `26` is the pin number on the microcontroller where the button is connected.
- `Pin.IN` configures the pin as an input.
- `Pin.PULL_UP` enables the internal pull-up resistor, which ensures that the pin reads as `1` (high) when the button is not pressed.

Polling vs. Interrupts



Polling: continuously checks the state of the button in a “while(true) loop.

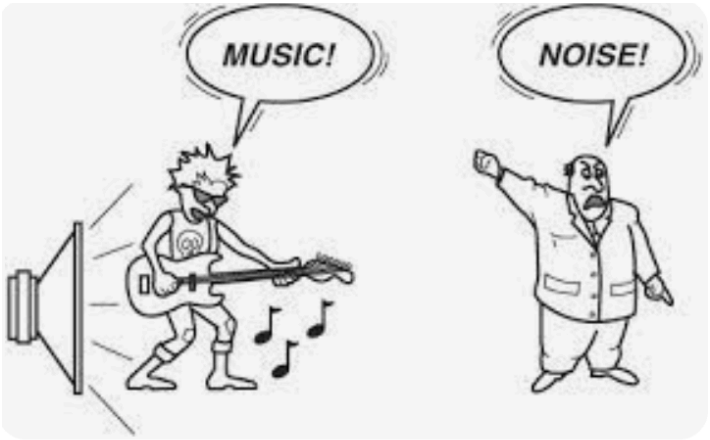
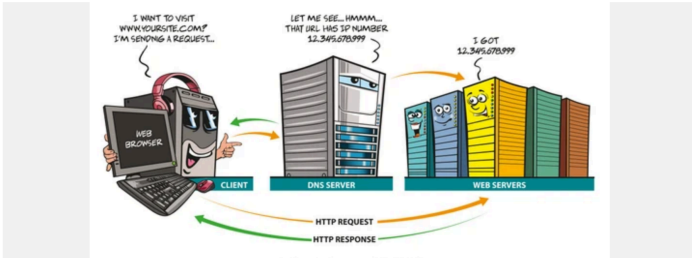
Polling: The Overly Attached Processor (The CPU is constantly interrupting its own work to ask, "Do you need me now?" It's like a clingy partner who can't let you have a moment of peace. This method can be a bit annoying and inefficient because the CPU spends a lot of time asking questions instead of doing its own work.)

CPU: "Are you ok?" **Device:** "I am fine"
CPU: "Are you ok?" **Device:** "I am fine"
…
CPU: "Are you ok?" **Device:** "Button pressed .."

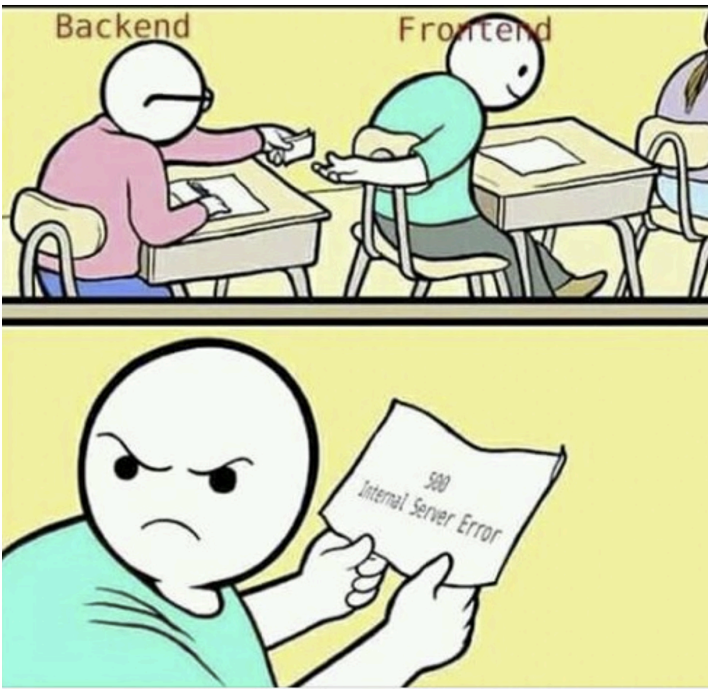
Interrupts: In more advanced IoT applications, interrupts can be used instead of polling. Interrupts are a mechanism where the hardware notifies the microcontroller immediately when an event (like a button press) occurs, allowing the program to respond quickly without constantly checking the state.

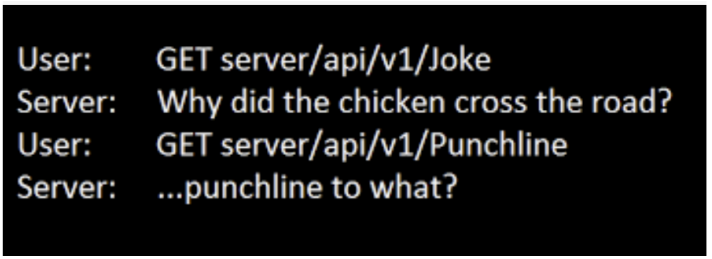
let's imagine the CPU as a chill and responsible partner. Interrupts are like this partner giving you space and only stepping in when you actually need something.

CPU: "I'll be working over here. Just call me if you need anything." **Device:** "Button pressed!"

	How interrupts are supported on the hardware level?
Day 3. Music vs Noise  	Music is typically composed of organized sound patterns, with elements like melody, harmony, rhythm, and timbre. These patterns follow specific rules or structures, such as scales, chords, and time signatures.
 	

Rest API





A REST API (Representational State Transfer Application Programming Interface) is a set of rules and conventions for building and interacting with web services. It allows different software applications to communicate with each other over the internet using standard HTTP methods.

RESTful APIs use standard HTTP methods to perform actions on resources. The four most commonly used methods are:

GET: Retrieve a resource or a list of resources.

POST: Create a new resource.

PUT: Update an existing resource or create one if it doesn't exist.

DELETE: Remove a resource

RESTful APIs use standard HTTP status codes to indicate the result of the API request:

200 OK: The request was successful.

201 Created: A new resource was successfully created.

204 No Content: The request was successful, but there is no content to return.

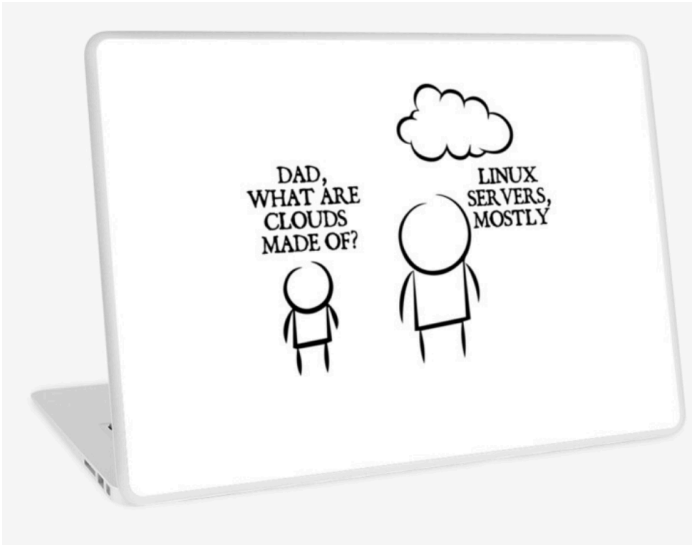
400 Bad Request: The request was malformed or invalid.

401 Unauthorized: Authentication is required and has failed or has not been provided.

404 Not Found: The requested resource does not exist.

500 Internal Server Error: The server encountered an error processing the request.

Web Cloud



Definitions:

Sensor vs. Actuator

- **Sensor**: A sensor is a device that detects and responds to some type of input from the physical environment. The input could be light, heat, motion, moisture, pressure, or any other environmental phenomena. Sensors convert these inputs into signals that can be read and processed by electronic devices.
- **Actuator**: An actuator is a device that takes electrical signals and converts them into physical action or movement. Examples of actuators include motors, servos, and solenoids, which can cause movement or control a mechanism or system.

Button as a Sensor

A button, also known as a pushbutton switch, is an **input device** that detects physical pressure when it is pressed by a user. Here's how it fits the definition of a sensor:

1. **Detection**: A button detects the presence of a user's press. When you press a button, it changes its state from open (not pressed) to closed (pressed), or vice versa.
2. **Signal Conversion**: This physical change in state is converted into an electrical signal that can be read by a microcontroller or other electronic device. When the button is pressed, it changes the voltage level on the pin it is connected to, which the microcontroller can then interpret.

How we started? (hello world GET method to register)

http://192.168.1.4/smart?serial_number=111111&user_name=Lilia&ip_address=11221212

What are we up to now?

http://192.168.1.112/smarthouse/v1/houses

Houses:app.py

Init:

```
self._log("* Event Loop")
self.loop = get_event_loop()

self._log("* IoT Hub Update Timer")
self._iot_hub_update_flag = False
self._iot_hub_timer = Timer(0)
self._iot_hub_timer.init(
   period=self.config["update_interval_ms"],
   mode=Timer.PERIODIC,
   callback=self._iot_hub_timer_callback,
)
….
if self.exit_code == 0:
   self._iot_hub_register()

   self._lcd_out(self.menu.get_current_content())

   self.loop.create_task(self.event_consumer())

   self._log("Enter event loop")
   try:
       self.loop.run_forever()
   except KeyboardInterrupt:
       self._log("Keyboard interrupt detected. Stopping...")

self._log("Exit event loop")
….
```

```
def _iot_hub_timer_callback(self, t):
    """Raise update flag."""
    self._iot_hub_update_flag = True


self._log("Setting up peripheral devices")
```

Device

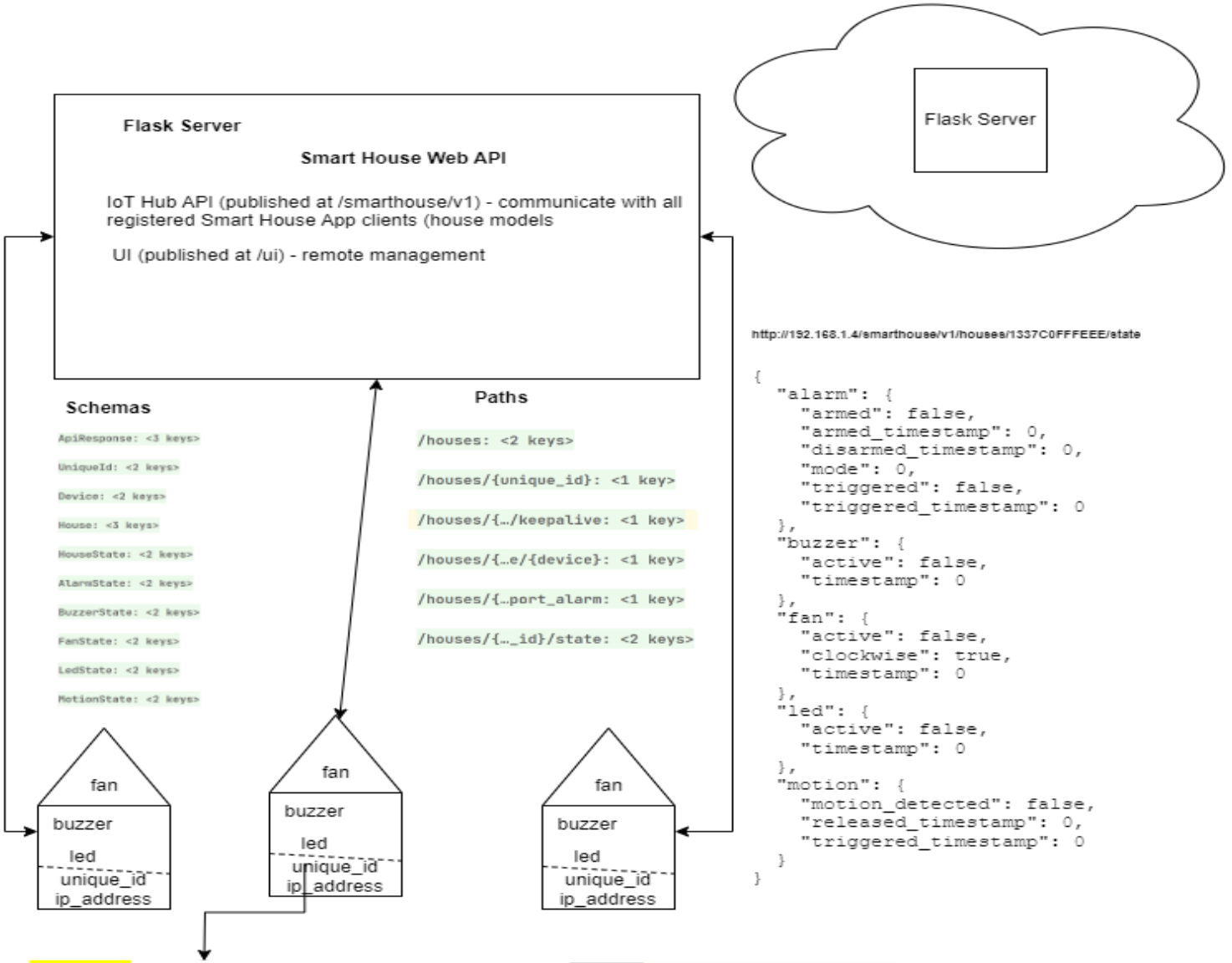
```
self.button_a = Button(
    name="/in/button_a",
    pin_num=26,
    event_queue=self.event_queue,
    debug=self._DEBUG,
)
...

def _iot_hub_register(self): // step 1
    """Check-in with IoT Hub and provide initial state."""
    self._log("Check-in with IoT Hub")
    self._iot_hub_call(
        call_method="POST",
        call_url=f"{self.config.get('api_endpoint')}/houses",
        call_json={
            "unique_id": self.unique_id,
            "ip_address": self.wlan.ip_address,
            "state": self._state,
        },
    )

def _iot_hub_keepalive(self):
    """Send keepalive and get latest state from IoT Hub if available."""
    self._log("Send keepalive to IoT Hub")
    response = self._iot_hub_call(
        call_method="PUT",
        call_url=f"{self.config.get('api_endpoint')}/houses/{self.unique_id}/keepalive",  # noqa: E501
        call_json={
            "unique_id": self.unique_id,
            "ip_address": self.wlan.ip_address,
        },
    )
```

```
    if response.status_code == 202:
        self.alarm.set_trigger(triggered=True, period_ms=4000)

    if response.status_code == 205:
        self._iot_hub_get_state()

def _iot_hub_set_state(self):
    """Send latest state to IoT Hub."""
    self._log("PUSH state to IoT Hub")
    self._iot_hub_call(
        call_method="PUT",
        call_url=f"{self.config.get('api_endpoint')}/houses/{self.unique_id}/state",
        call_json={
            "unique_id": self.unique_id,
            "ip_address": self.wlan.ip_address,
            "state": self._state,
        },
    )

def _iot_hub_get_state(self):
    """Get latest state from IoT Hub."""
    self._log("PULL state from IoT Hub")
    response = self._iot_hub_call(
        call_method="GET",
        call_url=f"{self.config.get('api_endpoint')}/houses/{self.unique_id}/state",
    )

    try:
        json_response = response.json()
    except ValueError:
        self._log("ERROR: State response not in JSON")
    finally:
        wall_msg_ui = json_response["wall_msg"]
        if self._state["wall_msg"] != wall_msg_ui:
            self._log("* WALL MSG: CHANGED")
            self._state["wall_msg"] = wall_msg_ui
            self._lcd_out(self.menu.get_current_content(), show_wall_msg=True)

        buzzer_active_ui = json_response["buzzer"]["active"]
        if self.buzzer._state["active"] != buzzer_active_ui:
            if buzzer_active_ui:
                self._log("* BUZZER: PLAY")
```

```
                self.buzzer.start_melody()
            else:
                self._log("* BUZZER: STOP")
                self.buzzer.stop_melody()
        else:
            self._log("* BUZZER: UNCHANGED")

        fan_active_ui = json_response["fan"]["active"]
        if self.fan._state["active"] != fan_active_ui:
            if fan_active_ui:
                self._log("* FAN: ON")
                self.fan.turn_on(clockwise=True)
            else:
                self._log("* FAN: OFF")
                self.fan.turn_off()
        else:
            self._log("* FAN: UNCHANGED")

        led_active_ui = json_response["led"]["active"]
        if self.led._state["active"] != led_active_ui:
            if led_active_ui:
                self._log("* LED: ON")
                self.led.turn_on()
            else:
                self._log("* LED: OFF")
                self.led.turn_off()
        else:
            self._log("* LED: UNCHANGED")

def _iot_hub_report_alarm(self):
    """Send alarm report to trigger other global alarms through IoT Hub."""
    self._log("Send alarm report to IoT Hub")
    self._iot_hub_call(
        call_method="PUT",
        call_url=f"{self.config.get('api_endpoint')}/houses/{self.unique_id}/report_alarm",  # noqa: E501
    )
```

Events:

```
async def event_consumer(self):
    """Process events asynchronously."""
    while True:
        if self.event_queue:
            event = self.event_queue.popleft()
            self.event_processor(event)

        if self._iot_hub_update_flag:
            self._iot_hub_update_flag = False
            self._iot_hub_keepalive()

        if self._state_change_remote:
            self._state_change_remote = False
            self._iot_hub_get_state()

        if self._state_change_local:
            self._state_change_local = False
            self._iot_hub_set_state()

        await sleep_ms(100)

def event_processor(self, event):
    """Process event."""
    self._log(f"Got event: {event}")

    if event["source"] == "/in/button_a" and event["state"]["pressed"]:
        self.menu.move_next()
        self._lcd_out(self.menu.get_current_content(), show_wall_msg=True)
```



http://192.168.1.4/smarthouse/v1/houses/1337C0FFFEEE/state


```
{
  "alarm": {
    "armed": false,
    "armed_timestamp": 0,
    "disarmed_timestamp": 0,
    "mode": 0,
    "triggered": false,
    "triggered_timestamp": 0
  },
  "buzzer": {
    "active": false,
    "timestamp": 0
  },
  "fan": {
    "active": false,
    "clockwise": true,
    "timestamp": 0
  },
  "led": {
    "active": false,
    "timestamp": 0
  },
  "motion": {
    "motion_detected": false,
    "released_timestamp": 0,
    "triggered_timestamp": 0
  }
}
```


```yaml
HouseState:
  type: object
  properties:
    alarm:
      $ref: "#/components/schemas/AlarmState"
    buzzer:
      $ref: "#/components/schemas/BuzzerState"
    fan:
      $ref: "#/components/schemas/FanState"
    led:
      $ref: "#/components/schemas/LedState"
    motion:
      $ref: "#/components/schemas/MotionState"
    wall_msg:
      type: string
```

```yaml
Device:
  type: string
  enum: [buzzer, fan, led]
```

```
<!DOCTYPE html>
<html>
 <head> ... </head>
  <!-- <body onload="fetchHouses();"> -->
 <body onload="fetchHouses();setInterval(fetchHouses, 1000);">
    <h2>Smart Houses</h2>
   <table>
     <thead> ... </thead>
     <tbody id="house_container"> == $0
       <tr id="1337CAFEC0DE"> ... </tr>
       <tr id="1337C0FFFEEE"> ... </tr>
     </tbody>
   </table>
  </body>
</html>
```

Not secure 192.168.1.112/ui

Paused in debugger

Smart Houses

Unique ID	IP Address	Status	Alarm	Buzzer	Fan	LED	Motion	Last Seen	Time Created	Time Modified	Time Deleted	Wall Message
1337CAFEC0DE	192.168.1.42	Lost	Disarmed	Off	Off	Off	No	2023-07-13 00:01:02	2023-07-13 00:01:02			
1337C0FFFEEE	192.168.1.24	Lost	Disarmed	Off	Off	Off	No	2023-07-13 00:01:02	2023-07-13 00:01:02			

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

Page Workspace

top
192.168.1.112
static
houses_dashboard.js
ui

houses_dashboard.js

```
1  function fetchHouses() {
2    const api_url = "/smarthouse/v1/houses"
3
4    fetch(api_url)
5      .then(response => response.json())
6      .then(data => {  data = (2) [{…}, {…}]
7        const tableBody = document.getElementById("house_container");
8
9        tableBody.innerHTML = "";
10
11       data.forEach(house => {
12         const row = document.createElement("tr");
13         row.setAttribute("id", house.unique_id)
14
15         const unique_id = document.createElement("td");
16         unique_id.textContent = house.unique_id;
17         row.appendChild(unique_id);
18
19         const ip_address = document.createElement("td");
20         ip_address.textContent = house.ip_address;
21         row.appendChild(ip_address);
```

Paused on breakpoint

Watch

```
data: Array(2)
  0: {global_alarm: false, ip_address: '192.168.1.42', state: {…}, status: 'Lost', timestamp_created:
  1: {global_alarm: false, ip_address: '192.168.1.24', state: {…}, status: 'Lost', timestamp_created:
  length: 2
  [[Prototype]]: Array(0)
```

Breakpoints

Pause on uncaught exceptions
Pause on caught exceptions

houses_dashboard.js

```
const tableBody = document.getElementById("house_container");
```

Scope

Local

```
this: undefined
data: (2) [{…}, {…}]
tableBody: <value unavailable>
```

Global

http://192.168.1.112/smarthouse/v1/houses


```
[
 {
  "global_alarm": false,
  "ip_address": "192.168.1.42",
  "state": {
   "alarm": {
    "armed": false,
    "armed_timestamp": 0,
    "disarmed_timestamp": 0,
    "mode": 0,
    "triggered": false,
    "triggered_timestamp": 0
   },
   "buzzer": {
    "active": false,
    "timestamp": 0
   },
   "fan": {
    "active": false,
    "clockwise": true,
    "timestamp": 0
   },
   "led": {
    "active": false,
    "timestamp": 0
   },
   "motion": {
    "motion_detected": false,
    "released_timestamp": 0,
    "triggered_timestamp": 0
   }
  },
  "status": "Lost",
  "timestamp_created": "2023-07-13 00:01:02",
  "timestamp_deleted": "",
  "timestamp_keepalive": "2023-07-13 00:01:02",
  "timestamp_modified": "",
  "unique_id": "1337CAFEC0DE",
  "update_from_ui": false
 },
 {
  "global_alarm": false,
  "ip_address": "192.168.1.24",
```

```
  "state": {
    "alarm": {
      "armed": false,
      "armed_timestamp": 0,
      "disarmed_timestamp": 0,
      "mode": 0,
      "triggered": false,
      "triggered_timestamp": 0
    },
    "buzzer": {
      "active": false,
      "timestamp": 0
    },
    "fan": {
      "active": false,
      "clockwise": true,
      "timestamp": 0
    },
    "led": {
      "active": false,
      "timestamp": 0
    },
    "motion": {
      "motion_detected": false,
      "released_timestamp": 0,
      "triggered_timestamp": 0
    }
  },
  "status": "Lost",
  "timestamp_created": "2023-07-13 00:01:02",
  "timestamp_deleted": "",
  "timestamp_keepalive": "2023-07-13 00:01:02",
  "timestamp_modified": "",
  "unique_id": "1337C0FFFEEE",
  "update_from_ui": false
 }
]
```

Data Flow:

http://192.168.1.112/

Request routed to flask server that has defined path for “/”:

```
@app.route("/")
def hello_world():
    """Serve test page."""
    return render_template("hello_world.html")
```

hello_world.html

```
<body>
    <h1>Hello, World!</h1>

    <a href="/ui">Click here for the IoT Hub Interface</a>
    <br>
    <a href="/static/list_houses">Or, go here to see static list of houses for testing</a>
</body>
```

```
@app.route("/ui")
def ui_index():
    """Serve dashboard page."""
    return render_template("ui_index.html", houses=HOUSES)
```

ui_index.html

```
<body onload="fetchHouses();setInterval(fetchHouses, 1000);">
```