Final Project - Thermostat Web Interface

CIS 129 - Robert Zuchowski - 2024-12-18

Introduction

My project is a simple web interface for the thermostat I use in my home.

I chose this topic for my final project because it is practical and something that I can realistically implement. It explores using python to interact with REST APIs, use SSDP protocol to discover devices on a network, and implement an HTTP web server.

Summary of Findings

Currently available solutions depend on the vendor of the thermostat. I have a Venstar smart thermostat for which there are available web and mobile apps. They are all proprietary and require both access to the internet and creating an account to use. Additionally they do not support older hardware and OS versions. For example, their Mac app requires an M1 chip and does not support Intel.

Real-World Applications

The real world application of this software is simple, it is a self hosted web app alternative to the proprietary apps offered by the thermostat vendor. The advantage it provides over existing solutions are privacy and that it can be operated over wi-fi with no internet connection required and will work with any device that can connect to a local network and run a web browser.

Design Approach

I am taking the approach of listing the actions the application should be able to do to establish the requirements of the project and then mapping those to one or more functions. I am also taking an inventory of the end points of the Venstar API to make sure I have a method for each of them in my wrapper class.

Requirements

Basic Functionality

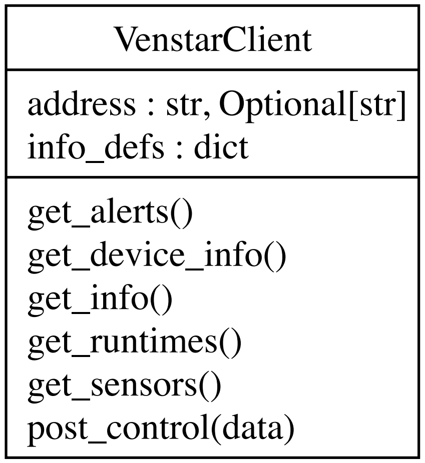
- Automatically discover the address of the thermostat
- Display the current temperature reading of the thermostat
- Display the current mode of the thermostat (heat, cool, auto, off)
- Display any active alerts, such as for filter replacement
- Toggle the fan setting of my thermostat between on and auto
- Change the mode of the thermostat between heat, cool, auto, and off
- Change the temperature thresholds for heating and cooling

Solution Design Proposal

My solution will involve these main components:

API Wrapper class

This will simplify interacting with the thermostats RESTful API by encapsualting its functionality into a class. It provides methods for all of the end points as well as a dictionary for mapping numeric codes to descriptions for sue in the interface.



```python
import json, requests

import ssdp

class VenstarClient:
    """API wrapper class for Venstar thermostats"""

    def __init__(self, location: str = None):
        if location:
            self.address: str = location
        else:
            self.address: str = ssdp.get_address()


    info_defs = {
    'mode' : {0: 'Off', 1 : 'Heat', 2 : 'Cool', 3 : 'Auto'},
    'state' : {0: 'Idle', 1: 'Heating', 2: 'Cooling', 3: 'Lockout', 4: 'Error'},
    'fan' : {0: 'Auto', 1 : 'On'},
    'fanstate': {0: 'Off', 1 : 'On'},
    'tempunits' : {0: 'Fahrenheit', 1: 'Celsius'},
    'schedule': {0: 'Disabled', 1: 'Enabled'},
    'schedulepart' : {0: 'morning', 1: 'day', 2: 'evening', 3: 'night', 255: 'inactive'},
    'away' : {0: 'Home', 1: 'Away'},
    'holiday': {0: 'not observing holiday', 1: 'observing holiday'},
    'override': {0 : 'Off', 1 : 'On'},
    'forceunocc': {0 : 'Off', 1 : 'On'},
    'availablemodes' : {0 : 'all modes', 1: 'heat/cool only', 2: 'heat only'}
    }

    def get_device_info(self):
        return json.loads(requests.get(self.address).text)

    def get_info(self):
        url = f'{self.address}/query/info'
        return json.loads(requests.get(url).text)
    def get_sensors(self):
        url = f'{self.address}/query/sensors'
        return json.loads(requests.get(url).text)

    def get_runtimes(self):
        url = f'{self.address}/query/runtimes'
        return json.loads(requests.get(url).text)

    def get_alerts(self):
        url = f'{self.address}/query/alerts'
        return json.loads(requests.get(url).text)

    def post_control(self, data):
        url = f'{self.address}/control'
        return json.loads(requests.post(url, data).text)
```

SSDP code:

```python
import socket

SSDP_ADDR = "239.255.255.250"
SSDP_PORT = 1900
SSDP_MX = 1
SSDP_ST = "venstar:thermostat:ecp"


ssdp_request_string = f"""M-SEARCH * HTTP/1.1
HOST: {SSDP_ADDR}:{SSDP_PORT}
MAN: "ssdp:discover"
MX: {SSDP_MX}
ST: {SSDP_ST}
"""

def parse_ssdp_response(response: bytes) -> dict:
    response_str = response.decode('utf-8')
    response_lines = response_str.split('\r\n')

    response_dict = {}
    for line in response_lines:
        if ': ' in line:
            key, value = line.split(': ', 1)
            response_dict[key] = value

    return response_dict


def get_address() -> str:

    ssdp_request = bytes(ssdp_request_string.replace('\n', '\r\n'), encoding='utf-8')

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(ssdp_request, (SSDP_ADDR, SSDP_PORT))
    response = sock.recv(1000)

    return parse_ssdp_response(response)['Location']
```

Flask web server

This will create and instance of and interact with the API wrapper class as well as serve the web interface.

Flask server code

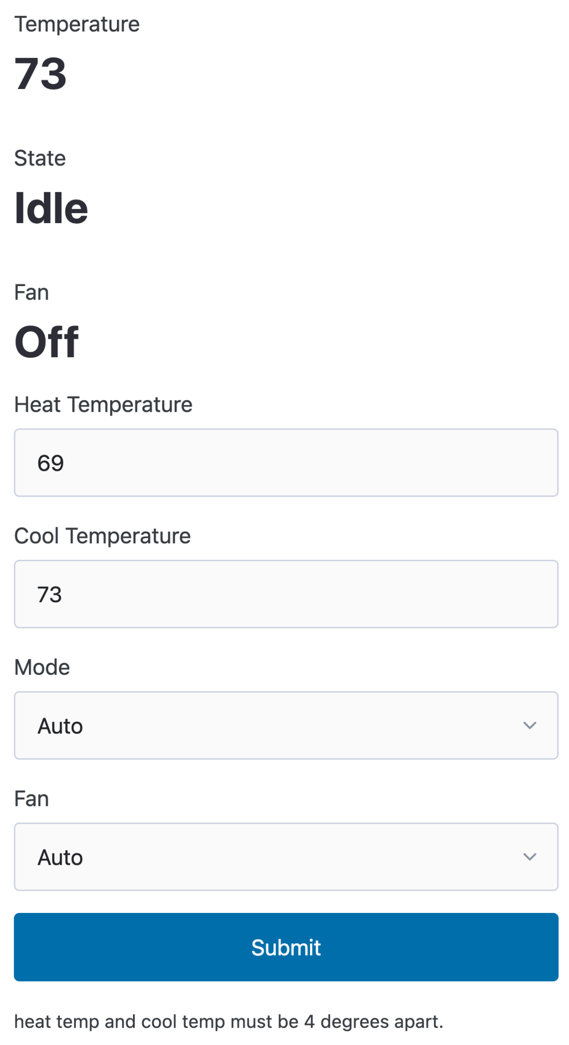
```
from flask import Flask, render_template, request, redirect

from wrapper import VenstarClient
```

```
venstar = VenstarClient()

app = Flask(__name__)

@app.route('/', methods=('GET', 'POST'))
def home():
    if request.method == 'POST':
        print(str(request.form))
        venstar.post_control(request.form)
        return redirect(request.referrer)
    return render_template('index.html', info=venstar.get_info(), info_defs=venstar.info_defs)


if __name__ == "__main__":
    app.run()
```

HTML interface

This is how the user will interact with the application. It will be composed of a simple form that when submitted will send requests to the flask web server which will then use an instance of the Venstar API wrapper class to interact with the thermostat. The HTML will be created using Jinja/Flask render templates.

Interface Image


Temperature

73

State

Idle

Fan

Off

Heat Temperature

69

Cool Temperature

73

Mode

Auto

Fan

Auto

heat temp and cool temp must be 4 degrees apart.

Open Questions

- How could this application be packaged and deployed?
- As a part of deployment, what kind of security considerations must be made if the application were accessible from the internet?
- How can I implement a database in the application for logging and reporting purposes?

Citations

Venstar. (n.d.). https://venstar.com/apps/

Discover. Code. Control. developer.venstar.com. (n.d.). https://developer.venstar.com

HTTP for humans. Requests. (n.d.). https://requests.readthedocs.io/en/latest/

Welcome to flask. Welcome to Flask - Flask Documentation (3.1.x). (n.d.). https://flask.palletsprojects.com/en/stable/

Wickramarachchi, A. (2020, October 8). *How to discover your network-attached devices using python SSDP*. Medium. https://python.plainenglish.io/how-iot-devices-are-discovered-for-communication-ac6a33a27ff0