



Nordpool Scraper Mini Project

Electricity Hourly Tariff Retrieval and Display Tool

Ance Strazdina

CMP408: IoT and Cloud Secure Development

2023/24

Contents

1	Introduction	1
1.1	Background.....	1
1.1.1	The area of IoT and Cloud Secure Development.....	1
1.2	Objectives	1
2	Procedure.....	3
2.1	Hardware	3
2.2	Software	3
2.2.1	Nordpool Scraper Main Functionality	4
2.2.2	Button functionality.....	5
2.3	Cloud	6
2.3.1	Webapp	6
2.3.2	MQTT broker	7
2.3.3	mqtt_subscriber.py	8
3	Conclusion.....	9
3.1	General Discussion.....	9
3.2	Future work	11
	References	12
	Appendices.....	13
	Appendix A - RPi Source Code.....	13
3.2.1	elspot.py	13
3.2.2	hourlyTariff.py.....	15
3.2.3	mqtt_publisher.py.....	16
3.2.4	refresher.py.....	17
3.2.5	button_module.c.....	18
3.2.6	Makefile	21
	Appendix B - Server Source Code	21
3.2.7	mqtt_subscriber.py	21
3.2.8	flask_app.py	22
3.2.9	index.html	22
3.2.10	styles.css	23

1 INTRODUCTION

1.1 BACKGROUND

Saving electricity has both environmental and financial benefits and Internet of Things (IoT) plays a role in this by monitoring energy networks. It can reduce electricity consumption and costs by automatically disabling unused devices and observing energy prices and usage (Pourbeik, 2021).

Nordpool is a Northern European power exchange group that publishes its hourly electricity tariffs. For Nordpool customers, knowing these is helpful and several projects, such as `nordpool.didnt.work` (n.d), which was inspired this project, assist consumers with price monitoring by utilising the Nordpool data which is what this project aims to accomplish.

1.1.1 The area of IoT and Cloud Secure Development

This project addresses several areas of IoT and cloud secure development by utilising a loadable kernel module (LKM) to interact with the Linux kernel, automated data retrieval, and hardware and cloud infrastructure for displaying it. This requires several security measures. LKM's proximity to the kernel demands functionality and security testing to not harm the system. The cloud must also address this with security rules and data protection. Lastly, data transfer must be secure, and all code must have robust error handling to ensure seamless function.

A lot of IoT manufacturers overlook security for profit (Neshenko, et al., 2019) so this project implements the appropriate security measures where possible and, if not accomplished, it discusses the reasons, impacts, and countermeasures for this.

1.2 OBJECTIVES

Successful implementation of the Nordpool Scraper must meet the following objectives:

1. Integrating the following **hardware**:
 - Raspberry Pi Zero W (RPi),
 - Three LEDs for displaying the current tariff colour,
 - A button for manually triggering the scripts.
2. **Software.**
 - Ensuring the main functionality with the following:
 - i. A daily script that retrieves data from the Nordpool API, labels, and saves it,
 - ii. A Message Queuing Telemetry Transport (MQTT) publisher that sends the tariff data to a webserver,
 - iii. An hourly script that retrieves the tariff label for the specific time and lights the corresponding LED.
 - Ensuring the button functionality:
 - i. LKM to handle a button press and notify user space,

- ii. A user space background service that waits for the notification and triggers the main scripts.
- 3. Integrating the following **cloud** components:
 - Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instance that:
 - i. Acts as an MQTT broker,
 - ii. Receives data from RPi over MQTT,
 - iii. Acts as a webserver and hosts a website displaying the received tariff data.

2 PROCEDURE

2.1 HARDWARE

This phase was completed by wiring 3 LEDs (Green, Yellow, Red), 1 two-legged button, 4 resistors, 8 male/female wires, and 1 male/male wire to the RPi. Figure 2-1 displays these components wired to the RPi and Figure 2-2 documents the used pins.

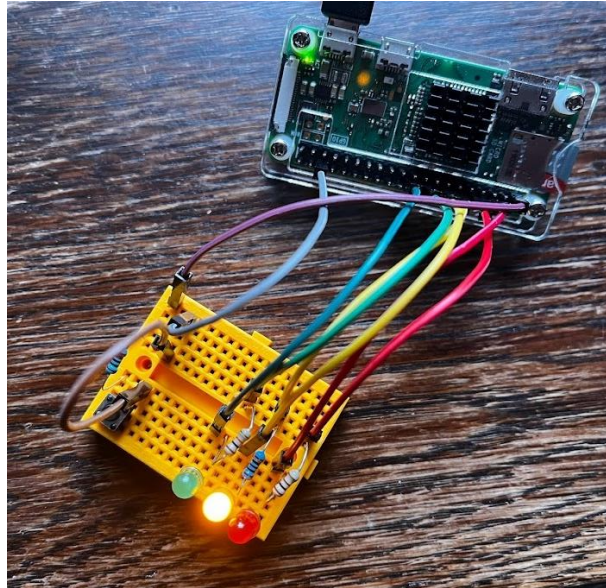


Figure 2-1 Hardware wired to the RPi.

Button - GPIO 16 and 3V3

LEDs:

Green - GPIO 24 and GND

Yellow - GPIO 18 and GND

Red - GPIO 14 and GND

Figure 2-2 Used RPi pins.

2.2 SOFTWARE

This phase was conducted on an Ubuntu system with later stages performed over an SSH connection to the RPi to integrate the software and hardware. The default password of the RPi was changed to ensure the security of the environment. The development was done in Python and C and the following Python libraries were installed:

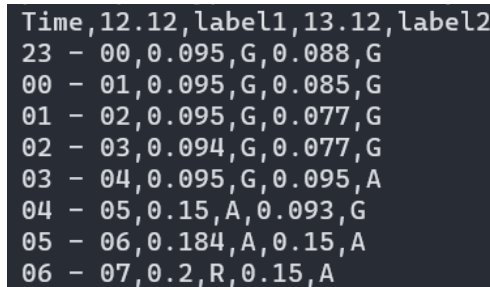
- pandas (pandas-dev, 2023) – data analysis and manipulation package.
- paho-mqtt (Eclipse, 2021) – enables applications to connect to an MQTT broker to publish and receive messages.
- nordpool (Huoman, 2022) – fetches data from the Nordpool API.

The developed code is available in Appendix A - RPi Source Code.

2.2.1 Nordpool Scraper Main Functionality

2.2.1.1 *elspot.py*

This script fetches hourly electricity tariff prices from Nordpool and writes them to a CSV file. Error handling is performed to ensure the reliability of the data. The data from the resulting elspot.csv file is then labelled in the *label* function. This function uses panda's *quintile* and *cut* utilities to label tariffs based on their value distribution – the top 25% values are labelled 'R', the bottom 25% - 'G', and the rest – 'A' (Figure 2-3). The data is rearranged for increased readability and rewritten to elspot.csv.



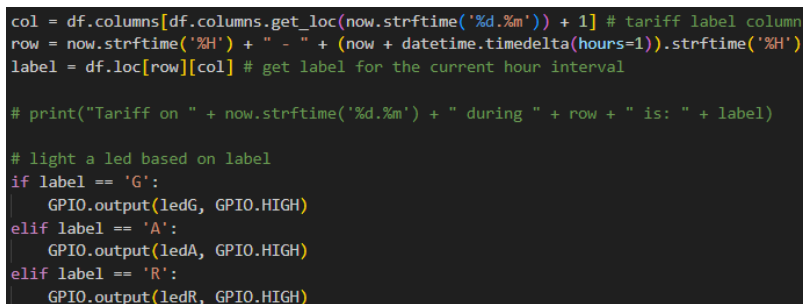
Time	12.12	label1	13.12	label2
23 - 00	0.095	G	0.088	G
00 - 01	0.095	G	0.085	G
01 - 02	0.095	G	0.077	G
02 - 03	0.094	G	0.077	G
03 - 04	0.095	G	0.095	A
04 - 05	0.15	A	0.093	G
05 - 06	0.184	A	0.15	A
06 - 07	0.2	R	0.15	A

Figure 2-3 Fragment from elspot.csv with labelled tariffs.

Lastly, this script calls triggers mqtt_publisher.py (2.2.1.3)

2.2.1.2 *hourlyTariff.py*

This script retrieves the tariff value corresponding to the current time from elspot.csv. Using RPi.GPIO, it changes the state of the LEDs, and lights the LED that corresponds to the retrieved label (red LED for 'R', yellow for 'A', and green for 'G') (Figure 2-4).



```
col = df.columns[df.columns.get_loc(now.strftime('%d.%m')) + 1] # tariff label column
row = now.strftime('%H') + " - " + (now + datetime.timedelta(hours=1)).strftime('%H')
label = df.loc[row][col] # get label for the current hour interval

# print("Tariff on " + now.strftime('%d.%m') + " during " + row + " is: " + label)

# light a led based on label
if label == 'G':
    GPIO.output(ledG, GPIO.HIGH)
elif label == 'A':
    GPIO.output(ledA, GPIO.HIGH)
elif label == 'R':
    GPIO.output(ledR, GPIO.HIGH)
```

Figure 2-4 Retrieving the label for the current time and changing the LED state.

2.2.1.3 *mqtt_publisher.py*

This script publishes elspot.csv to the MQTT broker (the AWS EC2 instance) with the topic *rpi/elspot*.

Both elspot.py and hourlyTariff.py are inserted into the crontab (Figure 2-5) with elspot.py executing once daily after the day-ahead data is released and hourlyTariff.py executing every hour.

```
# m h dom mon dow   command
50 11 * * * cd /home/pi/nordpool_scrape && /usr/bin/python3 /home/pi/nordpool_scrape/elspot.py
0 * * * * cd /home/pi/nordpool_scrape && /usr/bin/python3 /home/pi/nordpool_scrape/hourlyTariff.py
```

Figure 2-5 Crontab of the RPi.

2.2.2 Button functionality

2.2.2.1 *button_module.c*

To ensure that the tariff data and LED state could be manually refreshed with a button press if desired, an LKM was written. This module, based on kernel programming guides by Salzman, et al. (2023) and Corbet, et al. (2005), utilises interrupt handling and character devices. A button press interrupt is handled with *copy_to_user* to write a signal message 'B' to a button character device. A background user space service (2.2.2.2) immediately receives this if reading from the device file (Figure 2-6). A *button_read* function also provides this functionality if the character device read operation is performed after.

```
// button interrupt handler
static irqreturn_t button_irq(int irq, void *dev_id) {
    pr_info("Button pressed!\n");
    button_pressed = 1; // button is pressed flag

    // notify user space by writing to the char device
    if (button_device) {
        size_t ret = copy_to_user(button_device, &message, sizeof(char)); // write

        // return will be empty if successful, but if return is not empty catch it
        if (ret != 0) {
            pr_err("Failed to copy data to userspace\n");
            return -EFAULT;
        }
    }

    return IRQ_HANDLED;
}
```

Figure 2-6 Button press interrupt handled by notifying user space with a signal message.

This LKM was cross-compiled for the RPi on the CMP408 Fedora VM which had the required toolset (Figure 2-7) and transferred to the RPi where it was inserted into the kernel.

```
[cmp408@localhost button_module]$ make KERNEL=/home/cmp408/rpisc/linux CROSS=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabi/bin/arm-linux-gnueabi-
make ARCH=arm CROSS_COMPILE=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabi/bin/arm-linux-gnueabi- -C /home/cmp408/rpisc/linux M=/home/cmp408/button_module modules
make[1]: Entering directory '/home/cmp408/rpisc/linux'
  CC [M] /home/cmp408/button_module/button_module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M] /home/cmp408/button_module/button_module.mod.o
  LD [M] /home/cmp408/button_module/button_module.ko
make[1]: Leaving directory '/home/cmp408/rpisc/linux'
```

Figure 2-7 Cross-compiling the button module.

2.2.2.2 *refresher.py*

This script performs read operations on the button character device file that the button press signal messages are copied to. When the message 'B' is received from the kernel, this script runs *elspot.py* (which also triggers *mqtt_publish.py*) and *hourlyTariff.py* in that order (Figure 2-8) resulting in data and LED refresh.

```

BUTTON_DEV_PATH = "/dev/button_dev" # button character device path

def main():
    with open(BUTTON_DEV_PATH, "rb") as char_device:
        # print("Button app started. Waiting for button press...")
        while True:
            message = char_device.read(1)
            if message == b'B': # 'B' is the signal message for button press in the button module LKM
                # print("Button pressed!")
                subprocess.run(['python3', '/home/pi/nordpool_scrape/elspot.py'])
                subprocess.run(['python3', '/home/pi/nordpool_scrape/hourlyTariff.py'])

```

Figure 2-8 elspot.py and hourlyTariff.py run by refresher.py.

This script was configured to run as a daemon on boot (Figure 2-9).

```

[Unit]
Description=Resource refresh for nordpi project
After=network.target

[Service]
WorkingDirectory=/home/pi/nordpool_scrape
ExecStart=/usr/bin/python3 /home/pi/nordpool_scrape/refresher.py
Restart=always
User=pi

[Install]
WantedBy=multi-user.target

```

Figure 2-9 systemd service unit file for refresher.service.

2.3 CLOUD

This part of this project was addressed with an AWS EC2 instance. An Ubuntu Server 22.04 LTS t2.micro instance with 16GiB storage was created. It was configured to only allow inbound SSH connections from the developer's IP address, and allow all connections to ports 80 and 1883 (Figure 2-10).

Type	Protocol	Port range	Source	Description
Custom TCP	TCP	1883	0.0.0.0/0	MQTT
SSH	TCP	22	94.2.70.74/32	management access
HTTP	TCP	80	0.0.0.0/0	–

Figure 2-10 Nordpool Server security group inbound rules.

This configuration aligned with AWS Free Tier; however, an Elastic IP was configured to retain the same public IP address upon instance reboot which is a paid service. All code is available in Appendix B - Server Source Code.

2.3.1 Webapp

By referencing the guide by (Huiyeon, 2020) a lightweight Flask (pallets, 2023) webapp was deployed on port 80 of the server (Figure 2-11). This resulted in a production-ready server for the application and was configured to start on boot.

Time	12.12	13.12
23 - 00	0.095	0.088
00 - 01	0.095	0.085
01 - 02	0.095	0.077
02 - 03	0.094	0.077
03 - 04	0.095	0.095
04 - 05	0.15	0.093
05 - 06	0.184	0.15
06 - 07	0.2	0.15
07 - 08	0.2	0.15
08 - 09	0.2	0.15
09 - 10	0.184	0.148
10 - 11	0.184	0.15
11 - 12	0.155	0.15
12 - 13	0.15	0.149
13 - 14	0.171	0.149
14 - 15	0.195	0.146
15 - 16	0.181	0.154
16 - 17	0.172	0.148
17 - 18	0.155	0.15
18 - 19	0.152	0.13
19 - 20	0.147	0.113
20 - 21	0.128	0.101
21 - 22	0.111	0.104
22 - 23	0.098	0.091

Nordpool Hourly Spot Prices for LV (and Baltics).

Prices are in EUR/Kwh. Timezone - UTC. VAT is not included.

Red = highest 25% tariffs of the day, avoid heavy electricity usage; Green = lowest 25% tariffs of the day.

Author: Ance Strazdina

Figure 2-11 Webapp index page with colour-coded tariff values.

2.3.1.1 flask_app.py

This script manages the Flask app. It reads data from the elspot.csv file received over MQTT (2.3.3) and passes it to a function which renders the site based on a template. This template defines how the tariff data should be displayed, and implements some logic to colour the tariff value cells with CSS based on their label (Figure 2-12).

```
<!-- DATA ROWS -->
{% for row in rows %}
<tr>
  {% for col in row %}
    {% if loop.nextitem == "G" %}
    <td class="Green">{{ col }}</td>
    {% elif loop.nextitem == "A" %}
    <td class="Amber">{{ col }}</td>
    {% elif loop.nextitem == "R" %}
    <td class="Red">{{ col }}</td>
    {% else %}
    <td>{{ col }}</td>
    {% endif %}
  {% endfor %}
</tr>
{% endfor %}
```

Figure 2-12 Tariff cell colours determined based on their label.

2.3.2 MQTT broker

The Mosquitto (Eclipse, 2023) MQTT broker was configured to run on the server. Port 1883 was configured with an MQTT listener and for security anonymous connections were disabled (Figure 2-13). A *passwd* file

with usernames and hashed passwords was generated with the `mqtt_passwd` utility. This allowed authentication with the broker. An SSL configuration for Mosquitto was attempted but led to the service failure (likely due to certificate file permissions) and was not pursued further.

```
listener 1883
protocol mqtt
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Figure 2-13 Mosquitto configuration.

2.3.3 mqtt_subscriber.py

This script was configured as a daemon that runs on boot (Figure 2-14). It connects to the MQTT broker (the server itself), and receives data published with the *pi/elspot* topic. This data is saved to `elspot.csv` which is used to render the site with tariff data.

```
[Unit]
Description=MQTT subscriber service
After=network.target

[Service]
ExecStart=/usr/bin/python3 /home/ubuntu/nordpool_scrape/mqtt_subscriber.py
Restart=always
User=ubuntu

[Install]
WantedBy=multi-user.target
```

Figure 2-14 systemd service unit file for mqtt_sub.service.

3 CONCLUSION

3.1 GENERAL DISCUSSION

This project successfully implemented the desired functionality. It utilises scheduled scripts to get electricity tariff data from Nordpool and incorporates this with the colour-coded LEDs to display the status of the current tariff. An LKM notifies user space from the kernel about button press interrupts so scripts can also be triggered manually. Lastly, it utilises the lightweight MQTT to transfer data to an EC2 instance webserver to concisely present it in a Flask webapp. Figure 3-1 and Figure 3-2 demonstrate the flow of this project.

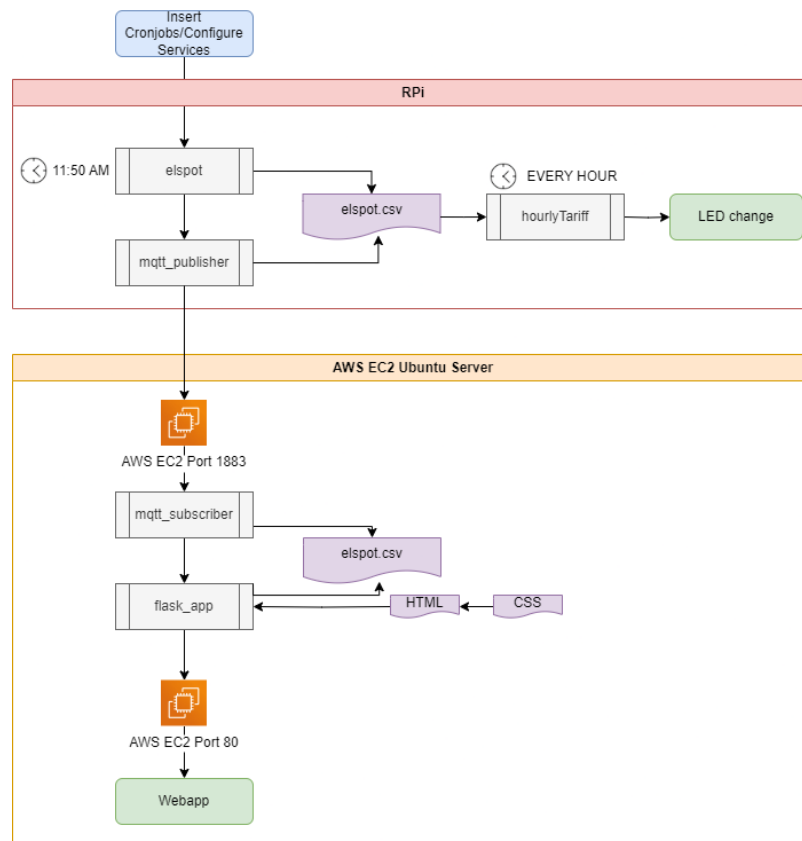


Figure 3-1 Scheduled Nordpool Scraper functionality.

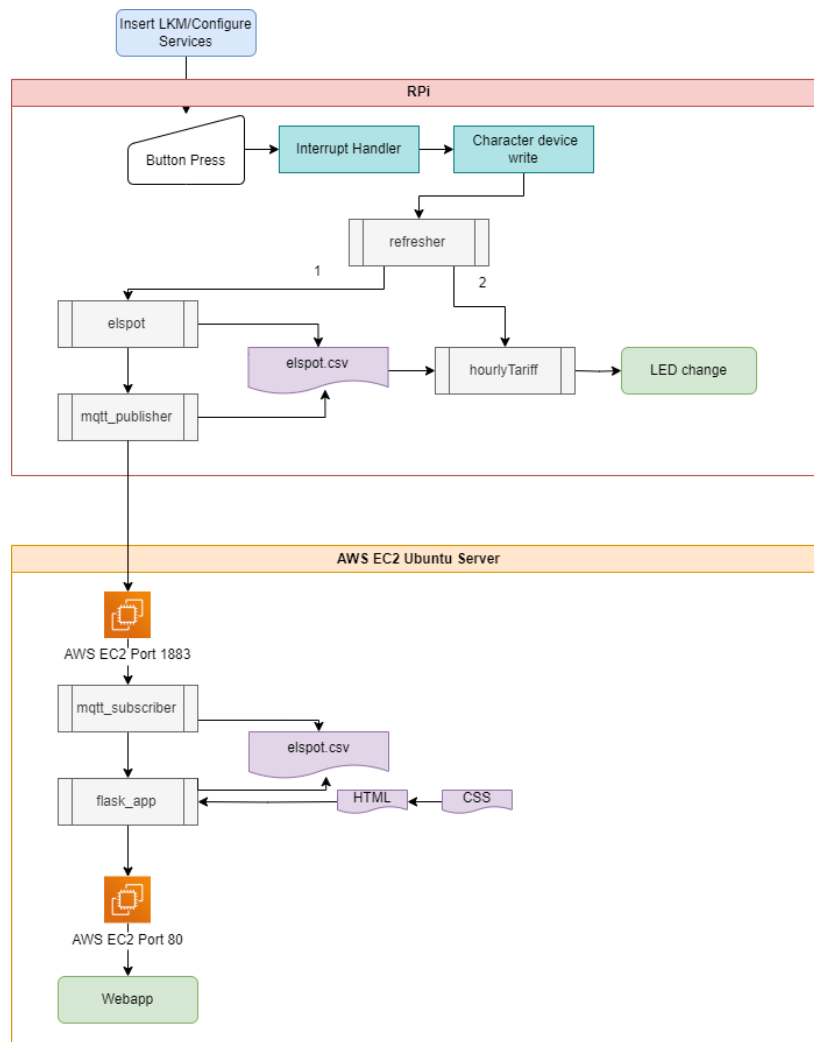


Figure 3-2 Manual Nordpool Scraper functionality.

Nonetheless, several improvements could be made. The LKM currently provides basic functionality but could benefit from extensive error handling, and addressing challenges such as interrupt sharing issues, that may arise in more complex deployments.

Permission issues arose when the user space script tried to read the character device file. The same file permission issue arose with the *passwd* file for Mosquitto and is likely the cause of unsuccessful MQTT+SSL implementation by this issue persisting for the generated certificates. The permissions for the button device and *passwd* files were changed. This was not critical in this case, but, in larger systems, user groups may need to be configured to address this with a better practice.

Lastly, the website could be served over HTTPS. This was initially planned but no domain to point the IP to was reserved as AWS Route 53 had restricted functionality on the provided account and it was decided against other domain registration approaches at this time. SSL is not crucial as no sensitive data is worked with; however, this is a good future step.

3.2 FUTURE WORK

Future development of this project would include addressing the issues identified above. Additionally, timezone support and the functionality to choose from Nordpool regions on the website would be beneficial. Lastly, the setup for this project could be automated with scripts for dependency, service, and cronjob setup.

REFERENCES

Corbet, J., Rubini, A. & Kroah-Hartman, G., 2005. Chapter 3. Char Drivers. In: *Linux Device Drivers, 3rd Edition*. Sebastopol: O'Reilly, pp. 42-72.

Eclipse, 2021. *paho.mqtt.python*. [Online]
Available at: <https://github.com/eclipse/paho.mqtt.python>
[Accessed 12 December 2023].

Eclipse, 2023. *mosquitto*. [Online]
Available at: <https://github.com/eclipse/mosquitto>
[Accessed 12 December 2023].

Huiyeon, K., 2020. *Step-by-step visual guide on deploying a Flask application on AWS EC2*. [Online]
Available at: <https://medium.com/techfront/step-by-step-visual-guide-on-deploying-a-flask-application-on-aws-ec2-8e3e8b82c4f7>
[Accessed 12 December 2023].

Huoman, K., 2022. *nordpool*. [Online]
Available at: <https://github.com/kiipe/nordpool>
[Accessed 12 December 2023].

Neshenko, N. et al., 2019. Demystifying IoT Security: An Exhaustive Survey. *IEEE Communications Surveys & Tutorials*, 21(3), pp. 2702 - 2733.

nordpool.didnt.work, n.d. *Nordpool spot prices (day-ahead, hourly, LV)*. [Online]
Available at: <https://nordpool.didnt.work/>
[Accessed 13 December 2023].

pallets, 2023. *flask*. [Online]
Available at: <https://github.com/pallets/flask>
[Accessed 12 December 2023].

pandas-dev, 2023. *pandas*. [Online]
Available at: <https://github.com/pandas-dev/pandas>
[Accessed 6 December 2023].

Pourbeik, P., 2021. *How to use IoT for energy efficiency and sustainability*. [Online]
Available at: <https://www.techtarget.com/iotagenda/feature/How-to-use-IoT-for-energy-efficiency-and-sustainability>
[Accessed 13 Decmeber 2023].

Salzman, P. J. et al., 2023. *The Linux Kernel Module Programming Guide*. [Online]
Available at: <https://sysprog21.github.io/lkmpg/>
[Accessed 12 December 2023].

APPENDIX A - RPI SOURCE CODE

3.2.1 elspot.py

```
"""
Name: elspot.py
Desc: gets Nordpool hourly spot prices for today and tomorrow in Latvia, saves to a CSV, and
labels hourly tariffs based on their value.
Auth: Ance Strazdina
Date: 10/12/2023
"""

# imports
from nordpool import elspot
from requests.exceptions import ConnectionError
import datetime, os.path, csv, sys, subprocess
import pandas as pd

area = 'LV' # area to get spot prices for
# this script gets data for Latvia, however, the rest of the Baltic States have the same
tariffs

# label tariffs as affordable, OK, expensive based on price distribution
def label(c, n):
    df = pd.read_csv('elspot.csv')

    # read in todays prices, calculate quartiles, and label hourly tariffs
    prices = df[c['end']].strftime('%d.%m')
    q1, q3 = prices.quantile([0.25, 0.75])
    df['label' + str(len(df.columns)-2)] = pd.cut(prices, bins=[float('-inf'), q1, q3,
float('inf')], labels=['G', 'A', 'R'])

    # read in tomorrows prices, calculate quartiles, and label hourly tariffs
    prices = df[n['end']].strftime('%d.%m')
    q1, q3 = prices.quantile([0.25, 0.75])
    df['label' + str(len(df.columns)-2)] = pd.cut(prices, bins=[float('-inf'), q1, q3,
float('inf')], labels=['G', 'A', 'R'])

    """
    R(ed) - Expensive
    A(mber) - OK
    G(reen) - Affordable
```

```

"""

# reorder columns and save
col = df.pop('label1')
df.insert(2, col.name, col)
df.to_csv('elspot.csv', index=False)

# write spot prices to csv
def writedata(c, n):
    # overwrite the existing file (if one exists)
    if os.path.isfile('elspot.csv'):
        f = open('elspot.csv', "w+")
        f.close()

    # add column labels
    with open('elspot.csv', 'a', newline='') as file:
        writer = csv.writer(file)
        row = ['Time', c['end'].strftime('%d.%m'), n['end'].strftime('%d.%m')] # Time,
[today's date], [tomorrow's date]
        writer.writerow(row)

    # write spot prices to csv
    for x in c['areas']['LV']['values']:
        with open('elspot.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            # [current hour] - [next hour], today's tariff, tomorrow's tariff. tariffs in kWh
            rounded to 3 decimal points.
            row = [x['start'].strftime('%H') + " - " + x['end'].strftime('%H'),
round(x['value']/1000, 3),
round(n['areas']['LV']['values'][c['areas']['LV']['values'].index(x)]['value']/1000, 3)]
            writer.writerow(row)

# main function
def main():
    # get spot prices for today and tomorrow
    # dates in 'yyyy-mm-dd' format
    try:
        if datetime.datetime.now(datetime.timezone.utc).time() >= datetime.time(11, 50):
            current_day =
elspot.Prices().hourly(end_date=datetime.datetime.now(datetime.timezone.utc).date(),
areas=[area])
            next_day =
elspot.Prices().hourly(end_date=datetime.datetime.now(datetime.timezone.utc).date() +
datetime.timedelta(days=1), areas=[area])
        else:

```



```

        # if its too early to get data for tomorrow: current day = previous day; next day
= current day
        current_day =
elspot.Prices().hourly(end_date=datetime.datetime.now(datetime.timezone.utc).date() +
datetime.timedelta(days=-1), areas=[area])
        next_day =
elspot.Prices().hourly(end_date=datetime.datetime.now(datetime.timezone.utc).date(),
areas=[area])
    except ConnectionError:
        sys.exit("No connection.")

    if current_day['areas']['LV']['Average'] == float('inf') or
next_day['areas']['LV']['Average'] == float('inf'):
        sys.exit("Cannot get data right now.")

    writedata(current_day, next_day) # write data to a csv
    label(current_day, next_day) # label tariffs based on their value

    # trigger mqtt
    subprocess.run(['python3', 'mqtt_publisher.py'])

# call main
if __name__ == '__main__':
    main()

```

3.2.2 hourlyTariff.py

```

"""
Name: hourlyTariff.py
Desc: changes LED hourly based on tariff label
Auth: Ance Strazdina
Date: 11/12/2023
"""

# imports
import sys, datetime
import pandas as pd
import RPi.GPIO as GPIO

# led pins
ledG = 24
ledA = 18
ledR = 14

# gpio setup
GPIO.setwarnings(False)

```

```

GPIO.setmode(GPIO.BCM)
GPIO.setup(ledG, GPIO.OUT, initial = GPIO.LOW)
GPIO.setup(ledA, GPIO.OUT, initial = GPIO.LOW)
GPIO.setup(ledR, GPIO.OUT, initial = GPIO.LOW)

def main():
    try:
        # read in the csv with daily tariffs, set Time column as index
        df = pd.read_csv('elspot.csv') # or tariff.csv
    except FileNotFoundError:
        sys.exit("Couldn't find elspot.csv.")

    df.set_index('Time', inplace=True)

    now = datetime.datetime.now(datetime.timezone.utc) # get current date and time

    # get row and column to use based on current date and time
    col = df.columns[df.columns.get_loc(now.strftime('%d.%m')) + 1] # tariff label column to
    use (the column next to date column)
    row = now.strftime('%H') + " - " + (now + datetime.timedelta(hours=1)).strftime('%H') #
    represents current hour interval as "[current hour] - [next hour]"
    label = df.loc[row][col] # get label for the current hour interval

    # print("Tariff on " + now.strftime('%d.%m') + " during " + row + " is: " + label)

    # light a led based on label
    if label == 'G':
        GPIO.output(ledG, GPIO.HIGH)
    elif label == 'A':
        GPIO.output(ledA, GPIO.HIGH)
    elif label == 'R':
        GPIO.output(ledR, GPIO.HIGH)

# call main
if __name__ == '__main__':
    main()

```

3.2.3 mqtt_publisher.py

```

"""
Name: mqtt_publisher.py
Desc: send CSV to server via MQTT
Auth: Ance Strazdina
Date: 11/12/2023
"""

```

```

# imports
import paho.mqtt.client as mqtt
import socket, sys

# MQTT details
broker = '' # Elastic IP of the EC2 instance
user = '' # username here
passwd = '' # password here
topic = 'pi/elspot'

# read in the CSV file content
try:
    with open('elspot.csv', 'r') as file:
        csv_content = file.read()
except FileNotFoundError::
    sys.exit("Couldn't find elspot.csv.")

# create client and connect
client = mqtt.Client()
client.username_pw_set(user, passwd)
try:
    client.connect(broker)
except socket.timeout:
    sys.exit("Couldn't establish connection to the broker.")

# publish data
client.publish(topic, csv_content)
client.disconnect()

```

3.2.4 refresher.py

```

"""
Name: refresh.py
Desc: run scripts after a button press is detected to refresh data
Auth: Ance Strazdina
Date: 11/12/2023
"""

# imports
import subprocess

BUTTON_DEV_PATH = "/dev/button_dev" # button character device path

def main():
    with open(BUTTON_DEV_PATH, "rb") as char_device:
        # print("Button app started. Waiting for button press...")

```

```

        while True:
            message = char_device.read(1)
            if message == b'B': # 'B' is the signal message for button press in the button
module LKM
                # print("Button pressed!")
                subprocess.run(['python3', '/home/pi/nordpool_scrape/elspot.py'])
                subprocess.run(['python3', '/home/pi/nordpool_scrape/hourlyTariff.py'])

if __name__ == "__main__":
    main()

```

3.2.5 button_module.c

```

#include <linux/init.h>
#include <linux/module.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/fs.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Ance Strazdina");
MODULE_DESCRIPTION("Button Interrupt Userspace Notify Module");

#define BUTTON_GPIO_PIN 16 // button pin, code assumes the button is placed correctly
#define GPIO_DESC "Button Interrupt"

static int major_number;
static struct class *button_class;
static struct device *button_device;
static int button_pressed;
static char message = 'B'; // button press signal message

// button interrupt handler
static irqreturn_t button_irq(int irq, void *dev_id) {
    pr_info("Button pressed!\n");
    button_pressed = 1; // button is pressed flag

    // notify user space by writing to the char device
    if (button_device) {
        size_t ret = copy_to_user(button_device, &message, sizeof(char)); // write button
press signal message to char device (immediate userspace notify)

        // return will be empty if successful, but if return is not empty catch it
        if (ret != 0) {
            pr_err("Failed to copy data to userspace\n");

```

```

        return -EFAULT;
    }
}

return IRQ_HANDLED;
}

// callback for button char device being opened by the userspace app
static int button_open(struct inode *inode, struct file *file) {
    pr_info("Button char device opened\n");
    return 0;
}

// callback for button char device being read by the userspace app
static ssize_t button_read(struct file *file, char *buffer, size_t len, loff_t *offset) {
    if (button_pressed) {
        button_pressed = 0; // reset the button press flag
        return copy_to_user(buffer, &message, sizeof(char)) ? -EFAULT : 1; // write button
        // press signal message to char device (userspace notify if char dev file is read after button
        // press)
    } else {
        return 0; // button has not been pressed, no data available
    }
}

// callback for button charr device being closed by the userspace app
static int button_release(struct inode *inode, struct file *file) {
    pr_info("Button char device closed\n");
    return 0;
}

// functions to handle the userspace app opening, reading, and closing the button char device
static struct file_operations fops = {
    .open = button_open,
    .read = button_read,
    .release = button_release,
};

// initialise function
static int __init button_init(void) {
    // request gpio
    if (gpio_request(BUTTON_GPIO_PIN, GPIO_DESC) < 0) {
        pr_err("Failed to request GPIO %d\n", BUTTON_GPIO_PIN);
        return -ENODEV;
    }
}

```

```

// configure gpio
if (gpio_direction_input(BUTTON_GPIO_PIN) < 0) {
    pr_err("Failed to set GPIO %d as input\n", BUTTON_GPIO_PIN);
    gpio_free(BUTTON_GPIO_PIN);
    return -ENODEV;
}

// request irq for the gpio pin
if (request_irq(gpio_to_irq(BUTTON_GPIO_PIN), button_irq, IRQF_TRIGGER_RISING, GPIO_DESC,
NULL) < 0) {
    pr_err("Failed to request IRQ for GPIO %d\n", BUTTON_GPIO_PIN);
    gpio_free(BUTTON_GPIO_PIN);
    return -ENODEV;
}

major_number = register_chrdev(0, "button_char", &fops); // register button character
device

// create button class and character device: /dev/button_dev
button_class = class_create(THIS_MODULE, "button_class");
button_device = device_create(button_class, NULL, MKDEV(major_number, 0), NULL,
"button_dev");

pr_info("Button interrupt module loaded\n");
return 0;
}

static void __exit button_exit(void) {
    // destroy device and class
    device_destroy(button_class, MKDEV(major_number, 0));
    class_unregister(button_class);
    class_destroy(button_class);

    unregister_chrdev(major_number, "button_char"); // unregister character device

    // free irq and gpio
    free_irq(gpio_to_irq(BUTTON_GPIO_PIN), NULL);
    gpio_free(BUTTON_GPIO_PIN);

    pr_info("Button interrupt module unloaded\n");
}

module_init(button_init);
module_exit(button_exit);

```

3.2.6 Makefile

```
KERNEL := /home/cmp408/rpisc/linux
PWD := $(shell pwd)
obj-m += button_module.o

all:
    make ARCH=arm CROSS_COMPILE=$(CROSS) -C $(KERNEL) M=$(PWD) modules
clean:
    make -C $(KERNEL) M=$(PWD) clean
```

APPENDIX B - SERVER SOURCE CODE

3.2.7 mqtt_subscriber.py

```
"""
Name: mqtt_subscriber.py
Desc: susbscribe to the elspot topic and get data sent from RPi via MQTT
Auth: Ance Strazdina
Date: 11/12/2023
"""

# imports
import paho.mqtt.client as mqtt

# MQTT details
broker = '' # Elastic IP of the EC2 instance
user = '' # username here
passwd = '' # password here
topic = 'pi/elspot'

# callback function when a message is received
def on_message(client, userdata, msg):
    csv_content = msg.payload.decode()
    # save the CSV content to a file
    with open('/home/ubuntu/nordpool_scrape/elspot.csv', 'w') as file:
        file.write(csv_content)

client = mqtt.Client() # create client
client.on_message = on_message # callback function on message

# connect
client.username_pw_set(user, passwd)
client.connect(broker)
```

```
client.subscribe(topic) # subscribe to the topic
client.loop_forever()
```

3.2.8 flask_app.py

```
"""
Name: flask_app.py
Desc: read in CSV data and render index page with a html table
Auth: Ance Strazdina
Date: 11/12/2023
"""

# imports
import csv
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    # read in csv with data and render site
    with open('elspot.csv') as file:
        reader = csv.reader(file)
        header = next(reader)
        return render_template('index.html', header=header, rows=reader)

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

3.2.9 index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title> Nordpool Hourly Spot Prices </title>
    <link rel="stylesheet" type="text/css" href="{{url_for('static',
filename='css/styles.css')}}">
  </head>
  <body>
    <div align="center">
      <table>
        <!-- HEADER ROW -->
        <tr>
          {% for col in header %}
            <th>{{ col }}</th>
          {% endfor %}
        </tr>
```



```

<!-- DATA ROWS -->
{% for row in rows %}
<tr>
  {% for col in row %}
    {% if loop.nextitem == "G" %}
<td class="Green">{{ col }}</td>
    {% elif loop.nextitem == "A" %}
<td class="Amber">{{ col }}</td>
    {% elif loop.nextitem == "R" %}
<td class="Red">{{ col }}</td>
    {% else %}
<td>{{ col }}</td>
    {% endif %}
  {% endfor %}
</tr>
{% endfor %}
</table>
</div>
<div align="center">
  <h3>Nordpool Hourly Spot Prices for LV (and Baltics).</h3>
  <h4>Prices are in EUR/kWh. Timezone - UTC. VAT is not included.</h4>
  <h5>Red = highest 25% tariffs of the day, avoid heavy electricity usage; Green =
lowest 25% tariffs of the day. </h5>
  <footer>
    <p>Author: Ance Strazdina</p>
  </footer>
</div>
</body>
</html>

```

3.2.10 styles.css

```

body {
  font-family: Arial, sans-serif;
}

table, th, td{
  border: 1px solid black;
  border-collapse: collapse;
}

th, td {
  text-align: center;
  padding: 3px;
}

```

```
td:first-child {  
    font-weight: bold;  
}  
  
th:nth-child(3), td:nth-child(3), th:nth-child(5), td:nth-child(5) {  
    display: none;  
}  
  
.Green {  
    background-color: #008450;  
}  
  
.Amber {  
    background-color: #EFB700;  
}  
  
.Red {  
    background-color: #B81D13;  
}
```