

POLITECHNIKA ŚWIĘTOKRZYSKA
Wydział Elektrotechniki, Automatyki i Informatyki

Laboratorium: Języki skryptowe	
Temat: Aplikacja do zarządzania zużyciem energii w domu	
Autor: Mikołaj Pacierz	Grupa: 3ID11A
Prowadzący: dr. inż. Dariusz Michalski	Data oddania: 27.06.2025

1. Cel projektu

Celem projektu było stworzenie aplikacji umożliwiającej zarządzaniem zużyciem energii w domu.

2. Zakres funkcjonalny

- Dodawanie urządzeń przez interfejs graficzny
- Obliczanie zużycia energii na podstawie poboru mocy, oraz dziennego czasu pracy urządzenia.
- Obliczanie kosztów urządzeń na podstawie zużycia energii przez urządzenie.
- Obliczenie całkowitego zużycia energii.
- Obsługa błędów (np. przekroczenie maksymalnego poboru mocy, lub czasu pracy).
- Odczytywanie informacji o urządzeniach z pliku CSV.
- Zapisywanie informacji o urządzeniach do pliku CSV.

3. Struktura projektu

energy_manager_app/	
├── main.py	# Główny moduł uruchamiający aplikację (GUI)
├── models/	
│ ├── device.py	# Klasa Device
│ ├── energy_manager.py	# Klasa EnergyManager
│ └── __init__.py	# Inicjalizacja pakietu
├── charts/	
│ ├── energy_chart.py	# Funkcja tworząca wykres energii
│ ├── price_chart.py	# Funkcja tworząca wykres cen
│ └── __init__.py	# Inicjalizacja pakietu
├── gui/	
│ ├── add_window.py	# Okno dodawania urządzeń
│ ├── utils.py	# Funkcje pomocnicze (np. zapis/odczyt pliku CSV)
│ └── __init__.py	# Inicjalizacja pakietu
├── data/	
│ └── devices.csv	# Przykładowe dane urządzeń
├── output/	
│ └── devices_output.csv	# Wygenerowane dane urządzeń
├── requirements.txt	# Informacja o używanych pakietach
├── setup.cfg	# Plik konfiguracyjny dla flake8
└── README.md	# Opis działania aplikacji

4. Technologie i biblioteki

Aplikacja została napisana w środowisku programistycznym PyCharm Professional Edition na systemie Linux. W aplikacji wykorzystano następujące technologie i biblioteki:

- Python 3.11
- Tkinter
- os, csv, functools, matplotlib
- unittest, flake8, memory_profiler

5. Sposób uruchomienia programu

5.1 Instrukcja uruchomienia

Windows

- `git clone https://github.com/mikolaj-pacierz-psk/energy-manager.git`
- `cd energy-manager`
- `python -m venv .venv`
- `venv\Scripts\activate`
- `pip install -r requirements.txt`
- `python main.py`

Linux

- `git clone https://github.com/mikolaj-pacierz-psk/energy-manager.git`
- `cd energy-manager`
- `python -m venv .venv`
- `source .venv/bin/activate`
- `pip install -r requirements.txt`
- `python main.py`

5.2 Przykładowe dane wejściowe/wyjściowe

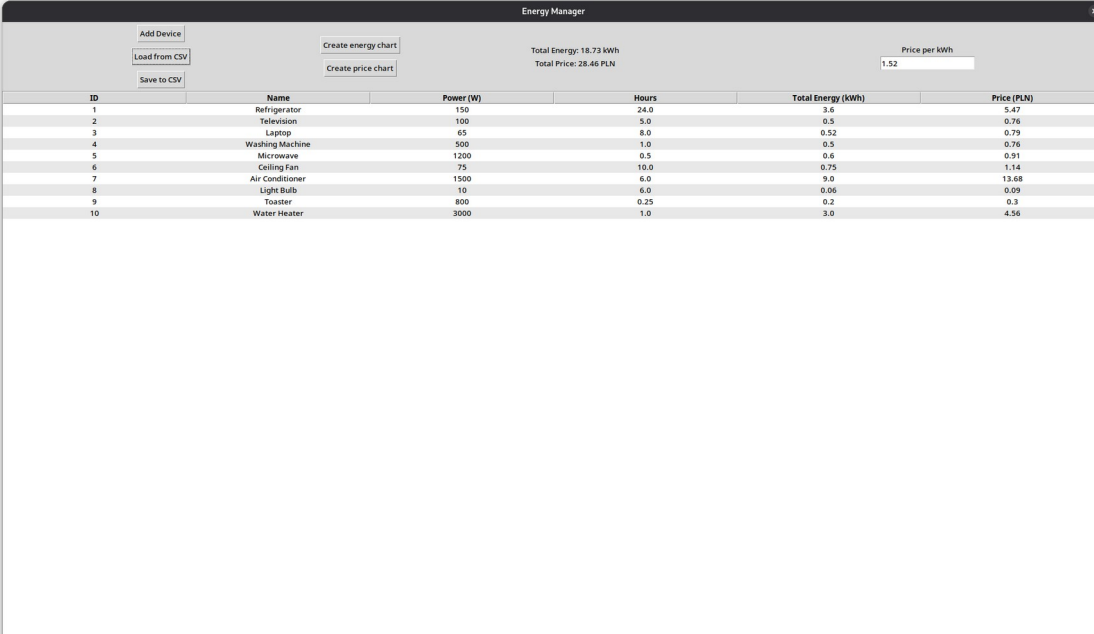
Zawartość pliku wejściowego *devices.csv*:

```
id,name,power,hours_per_day
1,Refrigerator,150,24
2,Television,100,5
3,Laptop,65,8
4,Washing Machine,500,1
5,Microwave,1200,0.5
6,Ceiling Fan,75,10
7,Air Conditioner,1500,6
8,Light Bulb,10,6
9,Toaster,800,0.25
10,Water Heater,3000,1
```

Zawartość pliku wyjściowego *devices_output.csv*:

```
id,name,power,hours_per_day,total_energy,price
1,Refrigerator,150,24.0,3.6,5.47
2,Television,100,5.0,0.5,0.76
3,Laptop,65,8.0,0.52,0.79
4,Washing Machine,500,1.0,0.5,0.76
5,Microwave,1200,0.5,0.6,0.91
6,Ceiling Fan,75,10.0,0.75,1.14
7,Air Conditioner,1500,6.0,9.0,13.68
8,Light Bulb,10,6.0,0.06,0.09
9,Toaster,800,0.25,0.2,0.3
10,Water Heater,3000,1.0,3.0,4.56
```

5.3 Zrzuty ekranu

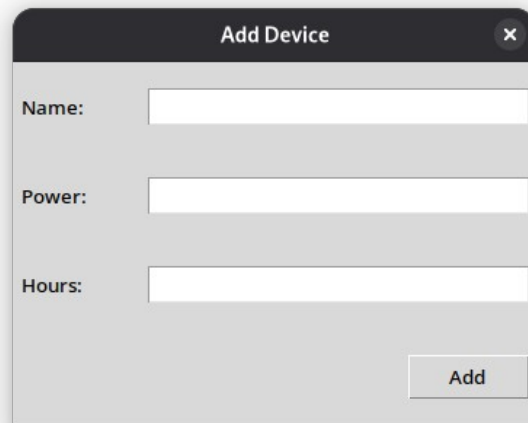


The screenshot shows the 'Energy Manager' application window. At the top, there are buttons for 'Add Device', 'Load from CSV', 'Save to CSV', 'Create energy chart', and 'Create price chart'. Below these buttons, the application displays a table with the following data:

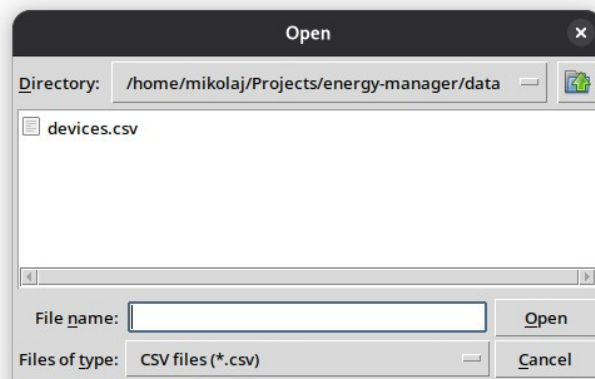
ID	Name	Power (W)	Hours	Total Energy (kWh)	Price (PLN)
1	Refrigerator	150	24.0	3.6	5.47
2	Television	100	5.0	0.5	0.76
3	Laptop	65	8.0	0.52	0.79
4	Washing Machine	500	1.0	0.5	0.76
5	Microwave	1200	0.5	0.6	0.91
6	Ceiling Fan	75	10.0	0.75	1.14
7	Air Conditioner	1500	6.0	9.0	13.68
8	Light Bulb	10	6.0	0.06	0.09
9	Toaster	800	0.25	0.2	0.3
10	Water Heater	3000	1.0	3.0	4.56

Below the table, there are summary statistics: 'Total Energy: 18.73 kWh' and 'Total Price: 28.46 PLN'. On the right side, there is a 'Price per kWh' input field with the value '1.52'.

Rysunek 1: Główny ekran aplikacji



Rysunek 2: Okno dodawania urządzenia



Rysunek 3: Okno wyboru wejściowego pliku

6. Przykłady kodu

6.1 Klasy

```
class Device:
    def __init__(self, name, power, hours_per_day):
        self.name = name
        self.power = int(power)
        self.hours_per_day = float(hours_per_day)
        assert 0 < self.hours_per_day <= 24, f"Hours must be a float value between 0 and 24"
        assert 0 < self.power <= 10000, f"Power must be a float value between 0 and 10000"

    def calculate_energy(self):
        return round(self.power * self.hours_per_day / 1000, 2)

    def calculate_price(self, price):
        return round(self.calculate_energy() * price, 2)
```

Listing 1: Klasa Device

```

class EnergyManager:
    def __init__(self, price):
        self.devices = []
        self.price = price

    def add_device(self, name, power, hours_per_day):
        self.devices.append(Device(name, power, hours_per_day))

    def load_from_csv(self, filename):
        with open(filename, 'r', newline='') as csvfile:
            reader = csv.DictReader(csvfile, delimiter=',')
            for row in reader:
                self.devices.append(Device(row['name'], row['power'], row['hours_per_day']))

    def save_to_csv(self, filename):
        with open(filename, 'w', newline='') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=['id', 'name', 'power', 'hours_per_day',
'total_energy', 'price'])
            writer.writeheader()
            for device in self.devices:
                writer.writerow({
                    'id': self.devices.index(device) + 1,
                    'name': device.name,
                    'power': device.power,
                    'hours_per_day': device.hours_per_day,
                    'total_energy': device.calculate_energy(),
                    'price': device.calculate_price(self.price)
                })

    def calculate_total_energy(self):
        return reduce(lambda energy, device: energy + device.calculate_energy(), self.devices,
0)

    def calculate_total_price(self):
        return reduce(lambda price, device: price + device.calculate_price(self.price),
self.devices, 0)

```

Listing 2: Klasa EnergyManager

6.2 Fragment funkcji funkcyjnej

```
def calculate_total_energy(self):
    return reduce(lambda energy, device: energy + device.calculate_energy(), self.devices,
0)

def calculate_total_price(self):
    return reduce(lambda price, device: price + device.calculate_price(self.price),
self.devices, 0)
```

Listing 3: Metody liczące całkowite zużycie energii oraz całkowity koszt

6.3 Obsługa wyjątków

```
def on_add():
    try:
        name = name_entry.get()
        power = power_entry.get()
        hours = hours_entry.get()
        energy_manager.add_device(name, power, hours)
        reload_table(energy_manager, table)
        window.destroy()
    except AssertionError as e:
        messagebox.showerror("Error", str(e), parent=window)
    except ValueError:
        messagebox.showerror("Error", "Power and hours must be a float value",
parent=window)
```

Listing 4: Metoda on_add()

W metodzie `on_add()` zostały obsługowane dwa rodzaje wyjątków tj. `AssertionError` i `ValueError`. `AssertionError` może zostać wywołany jeżeli moc urządzenia przekroczy 10000W lub gdy ilość godzin pracy urządzenia będzie większa niż 24. `AssertionError` jest podnoszony w klasie `Device`, gdzie asercje zostały użyte w konstruktorze klasy, aby działały jako limity.

7. Testowanie

7.1 Opis sposobu testowania

W aplikacji zastosowano testy jednostkowe napisane przy użyciu biblioteki *unittest*. Testy obejmują główne klasy aplikacji tj. *Device* i *EnergyManager*. Do analizy kodu wykorzystano również biblioteki *flake8* i *memory_profiler*.

7.2 Klasy testujące

```
class TestDevice(unittest.TestCase):
    def test_calculate_energy(self):
        device = Device("Fridge", 100, 24)
        self.assertEqual(device.calculate_energy(), 2.4)

    def test_calculate_price(self):
        device = Device("Fridge", 100, 24)
        self.assertEqual(device.calculate_price(1.52), 3.65)

    def test_invalid_hours(self):
        self.assertRaises(AssertionError, Device, "Fridge", 100, 25)

    def test_invalid_power(self):
        self.assertRaises(AssertionError, Device, "Fridge", 10001, 24)

if __name__ == '__main__':
    unittest.main()
```

Listing 5: Klasa TestDevice

```

class TestEnergyManager(unittest.TestCase):
    def setUp(self):
        self.energy_manager = EnergyManager(1.52)

    def test_add_device(self):
        self.energy_manager.add_device("Fridge", 100, 24)
        self.assertEqual(len(self.energy_manager.devices), 1)
        self.assertEqual(self.energy_manager.devices[0].name, "Fridge")
        self.assertEqual(self.energy_manager.devices[0].power, 100)
        self.assertEqual(self.energy_manager.devices[0].hours_per_day, 24)

    def test_save_to_csv(self):
        path_output = "./output/test_devices_output.csv"
        self.energy_manager.save_to_csv(path_output)
        self.assertTrue(os.path.exists(path_output))
        os.remove(path_output)

    def test_calculate_total_energy(self):
        self.energy_manager.add_device("Fridge", 100, 10)
        self.energy_manager.add_device("TV", 200, 5)
        self.assertEqual(self.energy_manager.calculate_total_energy(), 2)

    def test_calculate_total_price(self):
        self.energy_manager.add_device("Fridge", 100, 10)
        self.energy_manager.add_device("TV", 200, 5)
        self.assertEqual(self.energy_manager.calculate_total_price(), 3.04)

if __name__ == '__main__':
    unittest.main()

```

Listing 6: Klasa TestEnergyManager

```

@profile
def main():
    energy_manager = EnergyManager(1.52)

    root = tk.Tk()
    root.title("Energy Manager")
    root.geometry("1920x1080")

```

Listing 7: Fragment funkcji main()

7.3 Wyniki analizy kodu

```
~/Projects/energy-manager main
}
~/Projects/energy-manager main
} flake8 .

~/Projects/energy-manager main
} python -m memory_profiler main.py
Filename: main.py
```

Line #	Mem usage	Increment	Occurrences	Line Contents
12	74.2 MiB	74.2 MiB	1	@profile
13				def main():
14	74.2 MiB	0.0 MiB	1	energy_manager = EnergyManager(1.52)
15				
16	79.6 MiB	5.4 MiB	1	root = tk.Tk()
17	79.6 MiB	0.0 MiB	1	root.title("Energy Manager")
18	79.6 MiB	0.0 MiB	1	root.geometry("1920x1080")
19				
20	79.6 MiB	0.0 MiB	1	control_frame = tk.Frame(root)
21	79.6 MiB	0.0 MiB	1	control_frame.pack(fill="both")
22				

Rysunek 4: Wyniki analizy przy użyciu *flake8* i *memory_profiler*

8. Wnioski

8.1 Co się udało

- Stworzenie aplikacji do zarządzania zużyciem energii z interfejsem graficznym
- Stworzenie wykresów przedstawiających zużycie energii i koszty
- Zaimplementowane wczytywanie i zapisywanie do plików CSV
- Zaimplementowanie obsługi błędów, w celu zapobiegania wprowadzaniu nieprawidłowych danych (przekroczenie maksymalnego poboru mocy i czasu pracy)
- Automatyczne testy jednostkowe
- Statyczna analiza kodu przy pomocy *flake8*
- Dynamiczna analiza kodu przy pomocy *memory_profiler*

8.2 Co można było zrobić lepiej

- Dodanie integracji dat do aplikacji
- Zaimplementowanie obsługi innych formatów plików np. JSON

8.3 Jakie kompetencje zostały rozwinięte

- Projektowanie aplikacji z wykorzystaniem Tkinter
- Umiejętność korzystania z programowania funkcyjnego
- Zastosowanie testów i obsługi wyjątków
- Statyczna i dynamiczna analiza kodu

9. Repozytorium

Link do publicznego repozytorium w serwisie Github:

<https://github.com/mikolaj-pacierz-psk/energy-manager>