Wydział Automatyki, Elektroniki i Informatyki

RAPORT Z PROJEKTU

Przemysłowe Bazy Danych

Temat:

Zapis raportów do plików
(nazwa raportu+data+godzina utworzenia; html/pdf/jpg)

Autorzy:
Jacek Dolniak 5TI
Kacper Kosek 5TI

Gliwice 2024

1. Aplikacja SCADA

Opis funkcjonalności

Aplikacja SCADA zrealizowana w Pythonie z wykorzystaniem biblioteki **Tkinter** jako interfejsu graficznego.
Kluczowe funkcjonalności:

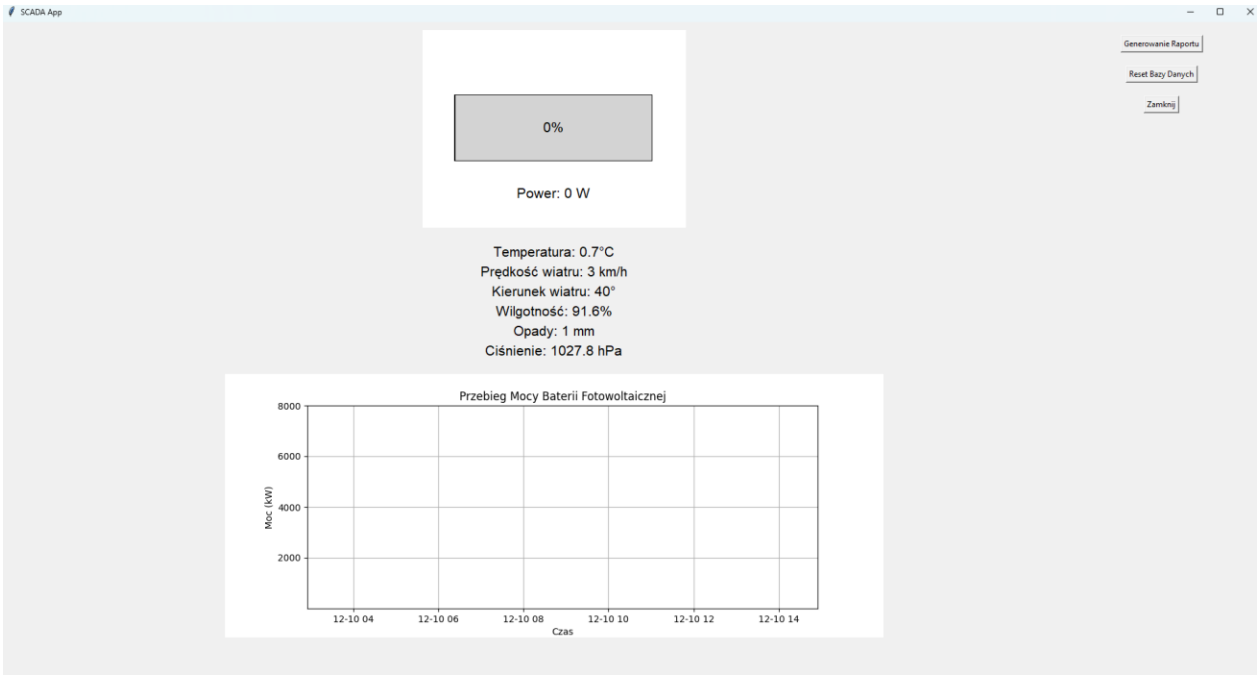
- **Wizualizacja procesu:** Wyświetlanie analogowego wskaźnika mocy (suwak procentowy), wykresu z danymi historycznymi oraz informacji pogodowych.
- **Automatyczny zapis danych:** Dane dotyczące mocy i warunków pogodowych są zapisywane co 3 sekundy do bazy danych MySQL.
- **Raporty:** Możliwość generowania raportów w formatach HTML, PDF i JPG.

Elementy aplikacji

a) **Interfejs użytkownika:** Tkinter, podzielony na sekcje (wskaźnik mocy, dane pogodowe, wykres, przyciski sterujące).

b) **Przechowywanie danych:** Baza danych MySQL z dwiema tabelami:
   - `power_data`: Dane o mocy paneli fotowoltaicznych.
   - `weather_data`: Dane pogodowe (temperatura, prędkość wiatru, wilgotność itp.).

c) **Integracja z zewnętrznymi źródłami danych:**
   - API danych pogodowych (IMGW).
   - Dane procesowe z interfejsu Modbus (symulacja w kodzie).



Rysunek 1 Wygląd aplikacji SCADA do obsługi procesu

1.1 Opis elementów SCADA przedstawionych na zrzucie ekranu

1. **Wskaźnik analogowy mocy** (górna część, prostokąt z wartością procentową):
    - Graficzna reprezentacja procentowego obciążenia systemu na podstawie wartości mocy.
    - Kolor wskaźnika zmienia się w zależności od poziomu obciążenia:
        - Zielony: niskie obciążenie.
        - Żółty: średnie obciążenie.
        - Czerwony: wysokie obciążenie.
    - Pod wskaźnikiem znajduje się bieżąca wartość mocy wyrażona w watach (W).

2. **Sekcja danych pogodowych** (środkowa część, lista tekstowa):
    - Prezentacja najnowszych odczytów pogodowych pobranych z API:
        - **Temperatura:** Obecna temperatura powietrza (°C).
        - **Prędkość wiatru:** Prędkość wiatru w km/h.
        - **Kierunek wiatru:** Kierunek wiatru w stopniach (°).
        - **Wilgotność:** Wilgotność względna powietrza (%).
        - **Opady:** Suma opadów w mm.
        - **Ciśnienie:** Ciśnienie atmosferyczne w hPa.

3. **Wykres mocy** (dolna część):
    - Wykres liniowy przedstawiający przebieg mocy paneli fotowoltaicznych w ciągu ostatnich 12 godzin.
    - Oś X: Czas (z oznaczeniem daty i godziny).
    - Oś Y: Moc wyrażona w kW.
    - Linie siatki umożliwiają lepszą analizę zmian wartości.

4. **Przyciski sterujące** (po prawej stronie):
    - **Generowanie Raportu:** Przyciski umożliwiające wygenerowanie raportu z danymi w formatach HTML, PDF i JPG.
    - **Reset Bazy Danych:** Usuwa wszystkie dane z tabel w bazie `scada_db`.
    - **Zamknij:** Zamyka aplikację SCADA.

2. Zapis danych do bazy MySQL

Opis bazy danych:
Baza danych scada_db zawiera dwie tabele:

a) **power_data**
    - Czas (timestamp): Czas pobrania danych.
    - Moc (float): Zmierzona moc (kW).

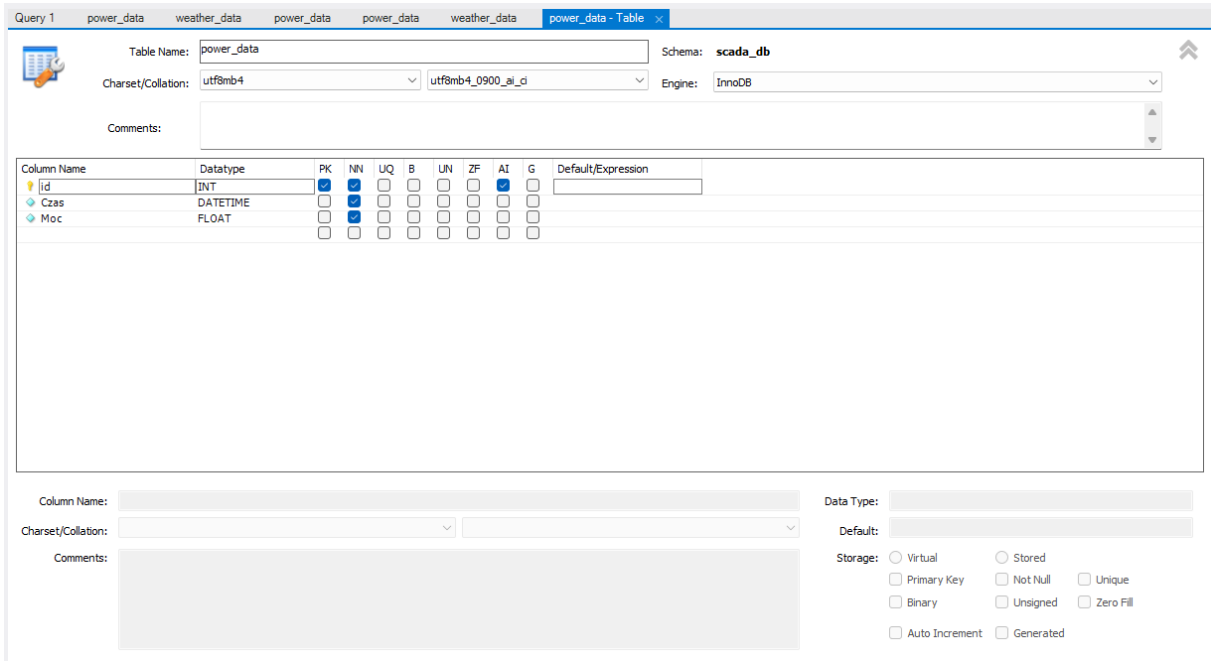
b) **weather_data**
    - Czas (timestamp): Czas pobrania danych.
    - temperatura (float): Temperatura (°C).
    - predkosc_wiatru (float): Prędkość wiatru (km/h).
    - kierunek_wiatru (int): Kierunek wiatru (°).
    - wilgotnosc (float): Wilgotność (%).

- opady (float): Suma opadów (mm).
- cisnienie (float): Ciśnienie (hPa).

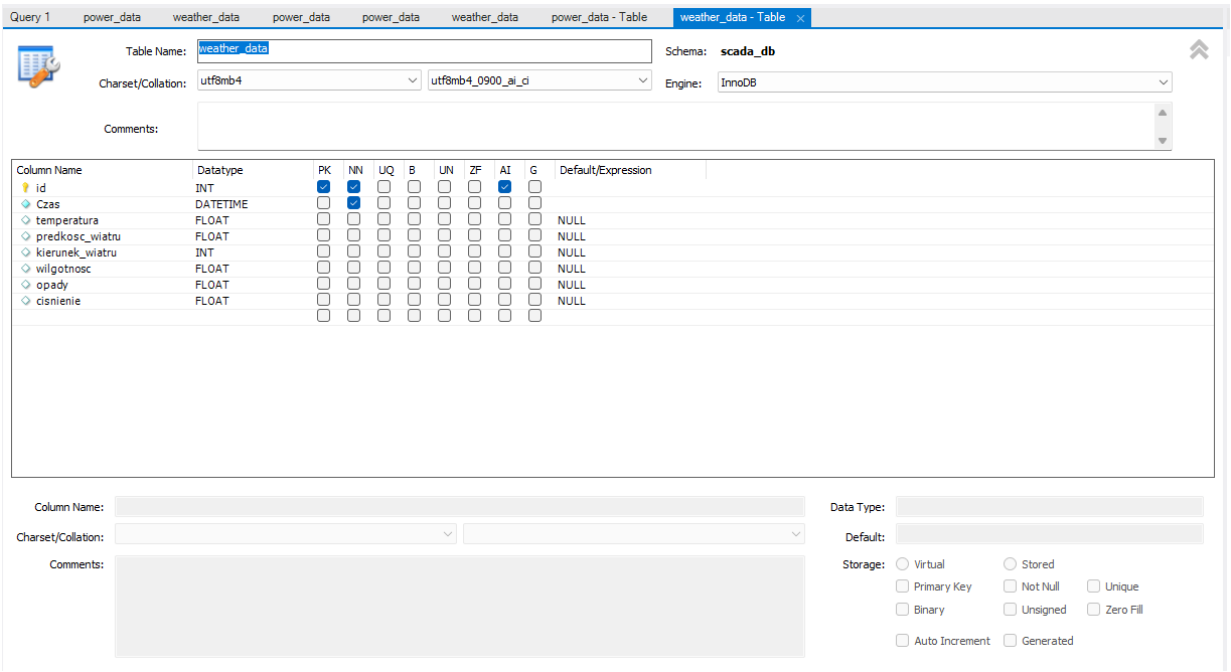
Mechanizm zapisu danych

- Dane o mocy i warunkach pogodowych są zapisywane do tabel przez funkcje get_power_data i get_weather_data:
    - Przykład zapytania SQL:
    INSERT INTO power_data (Czas, Moc) VALUES (%s, %s)

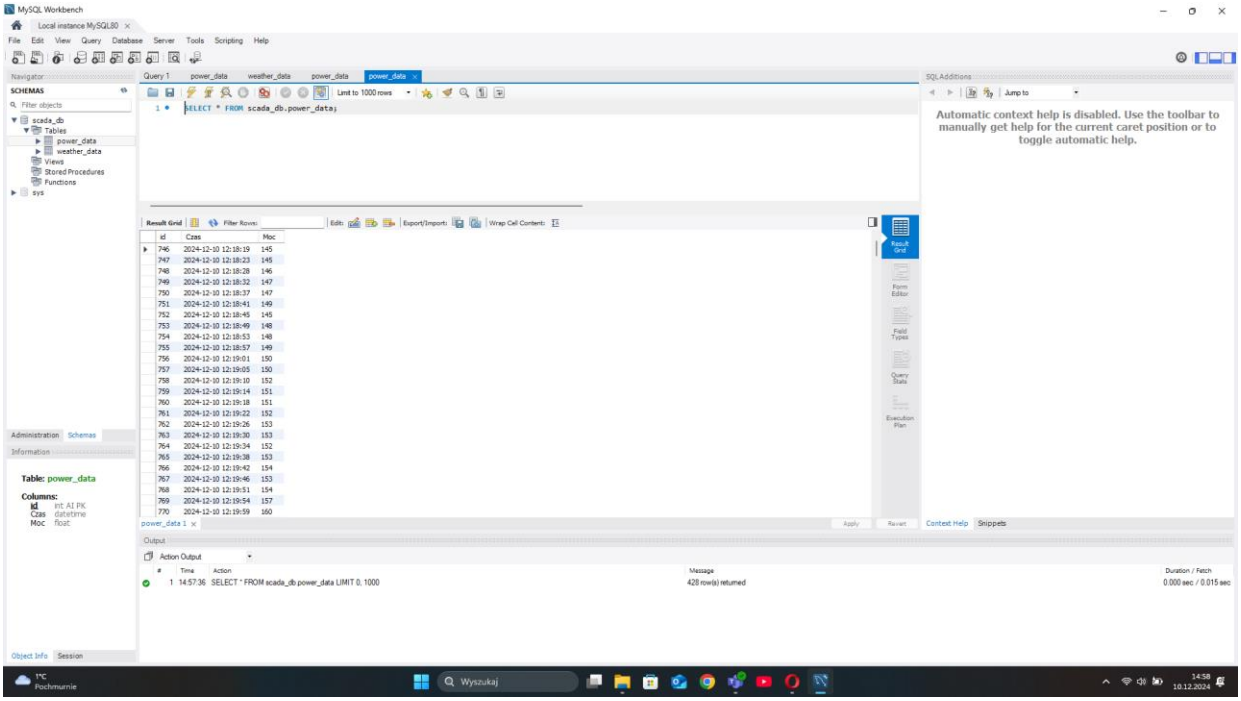
- Połączenie z bazą realizowane jest przy użyciu biblioteki **pymysql** oraz ORM **SQLAlchemy**.



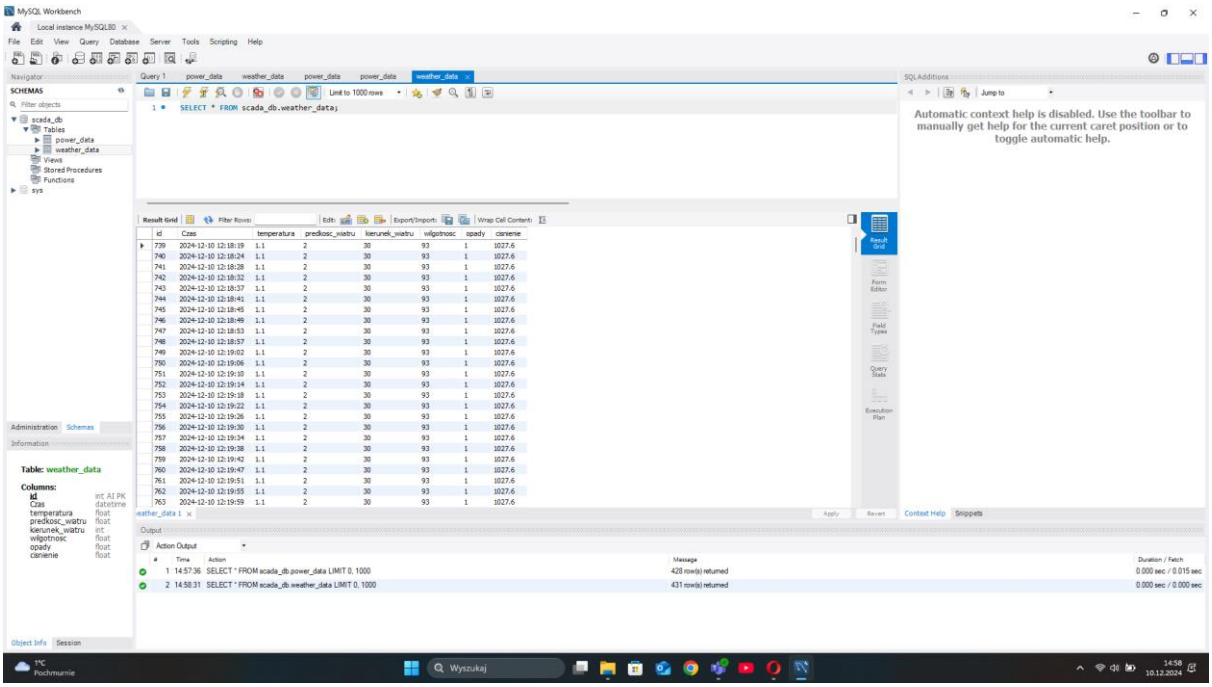
Rysunek 2 Rodzaje zmiennych dla bazy danych dotyczącej mocy falownika w MySQL Workbench



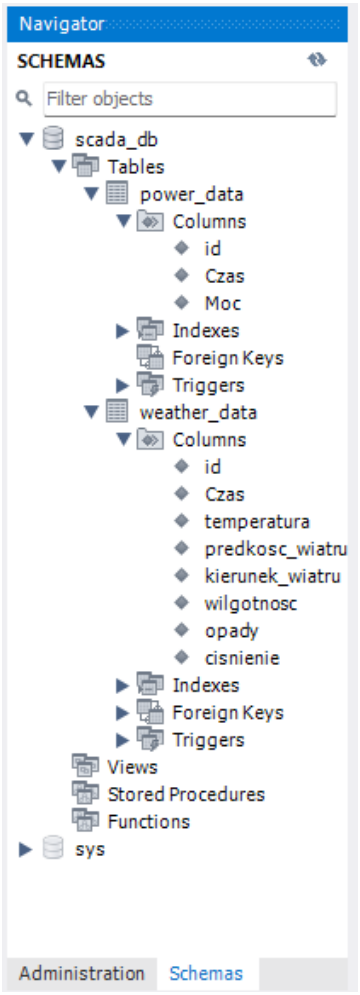
Rysunek 3 Rodzaje zmiennych dla bazy danych dotyczące parametrów meteorologicznych w MySQL Workbench



Rysunek 4 Rzeczywiste zmienne zarejestrowane w czasie pracy falownika



Rysunek 5 Rzeczywiste zmienne zarejestrowane dla warunków pogodowych



Rysunek 6 Schemat projektu bazy danych

3. Generowanie raportów

Zawartość raportów

Raport zawiera:

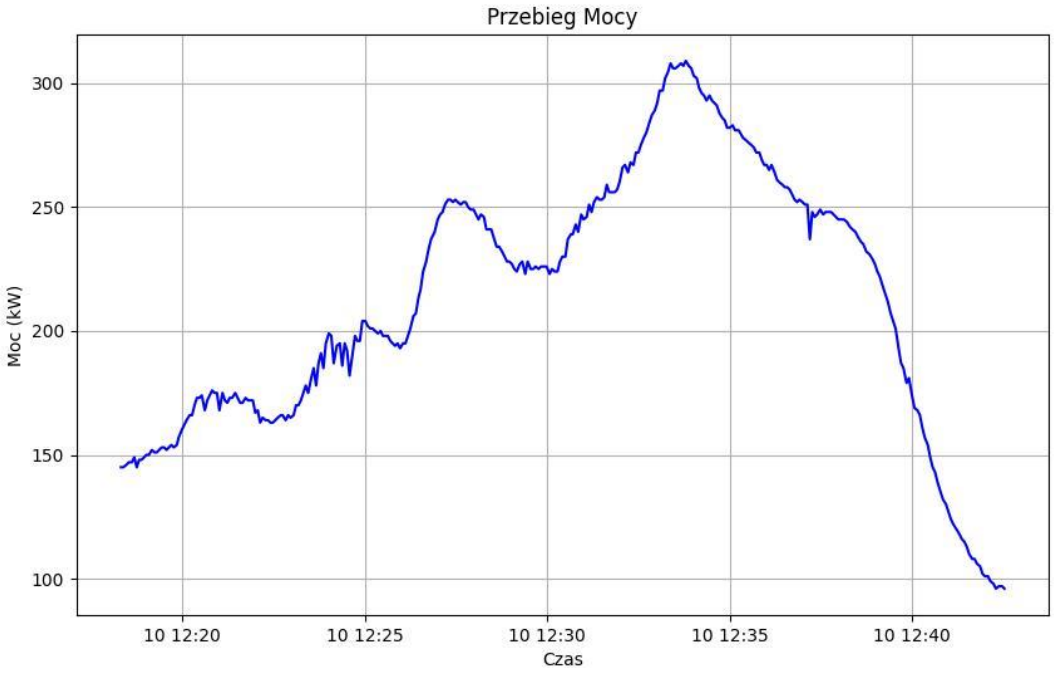
1. Dane o mocy pobrane z tabeli `power_data` w postaci tabeli i wykresu.
2. Dane pogodowe z tabeli `weather_data` w formie tabeli.

Generowane formaty

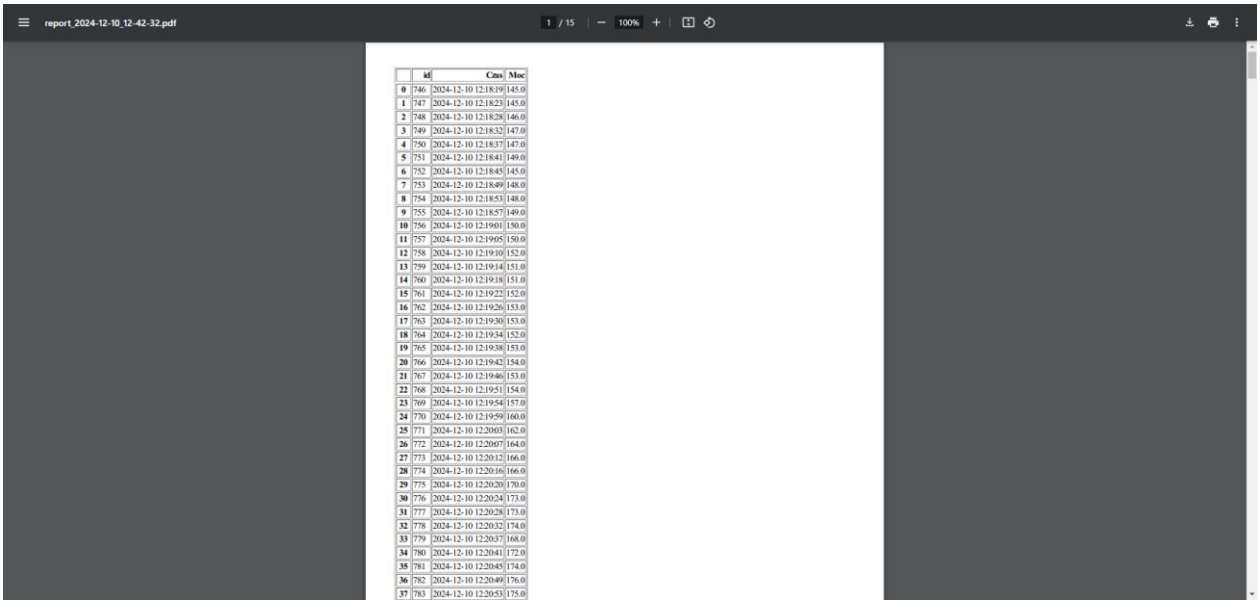
1. **HTML:**
   - Generowane za pomocą funkcji `to_html()` z biblioteki **pandas**.
2. **PDF:**
   - Konwersja z HTML do PDF za pomocą **pdfkit** i wkhtmltopdf.
3. **JPG:**
   - Wykresy zapisane w formacie graficznym za pomocą **matplotlib**.

Automatyczna nazwa pliku

Pliki są nazywane według wzorca:
`report_<data>_<godzina>.{format}`
np. `report_2024-12-10_14-30-45.pdf`.

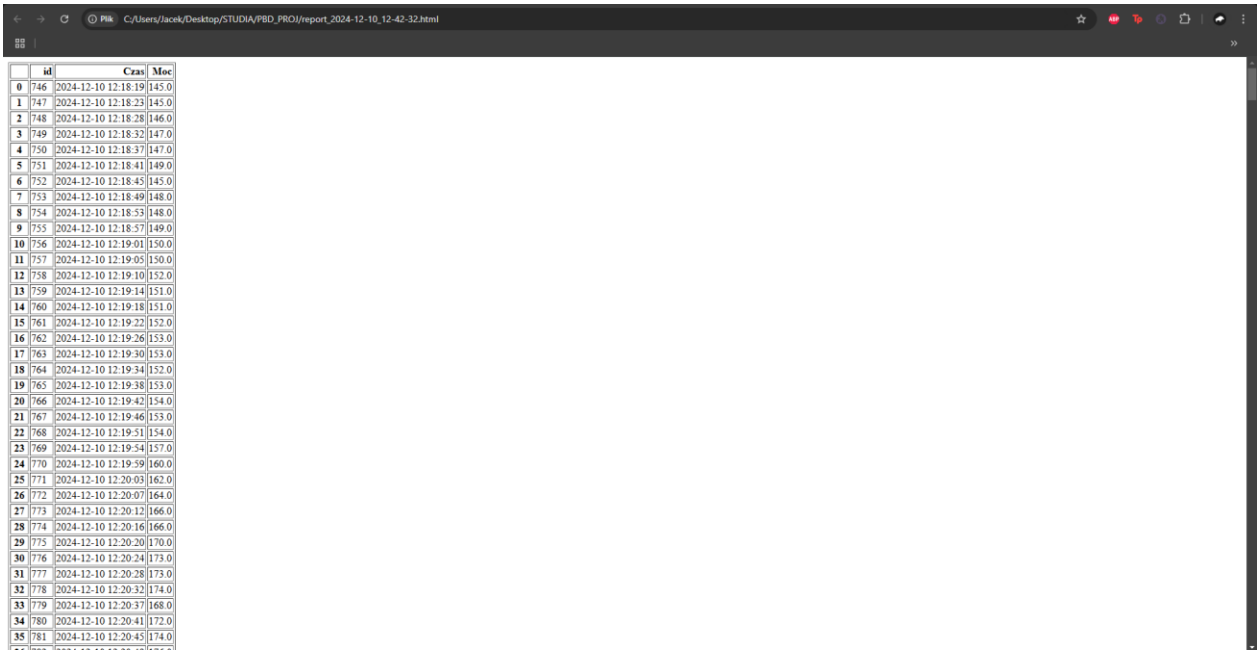


Rysunek 7 Generowany raport: Wersja jpg wykres mocy



	id	Czas	Moc
0	746	2024-12-10 12:18:19	145.0
1	747	2024-12-10 12:18:23	145.0
2	748	2024-12-10 12:18:28	146.0
3	749	2024-12-10 12:18:32	147.0
4	750	2024-12-10 12:18:37	147.0
5	751	2024-12-10 12:18:41	149.0
6	752	2024-12-10 12:18:45	145.0
7	753	2024-12-10 12:18:49	148.0
8	754	2024-12-10 12:18:53	148.0
9	755	2024-12-10 12:18:57	149.0
10	756	2024-12-10 12:19:01	150.0
11	757	2024-12-10 12:19:05	150.0
12	758	2024-12-10 12:19:10	152.0
13	759	2024-12-10 12:19:14	151.0
14	760	2024-12-10 12:19:18	151.0
15	761	2024-12-10 12:19:22	152.0
16	762	2024-12-10 12:19:26	153.0
17	763	2024-12-10 12:19:30	153.0
18	764	2024-12-10 12:19:34	152.0
19	765	2024-12-10 12:19:38	153.0
20	766	2024-12-10 12:19:42	154.0
21	767	2024-12-10 12:19:46	153.0
22	768	2024-12-10 12:19:51	154.0
23	769	2024-12-10 12:19:54	157.0
24	770	2024-12-10 12:19:59	160.0
25	771	2024-12-10 12:20:03	162.0
26	772	2024-12-10 12:20:07	164.0
27	773	2024-12-10 12:20:12	166.0
28	774	2024-12-10 12:20:16	166.0
29	775	2024-12-10 12:20:20	170.0
30	776	2024-12-10 12:20:24	173.0
31	777	2024-12-10 12:20:28	173.0
32	778	2024-12-10 12:20:32	174.0
33	779	2024-12-10 12:20:37	168.0
34	780	2024-12-10 12:20:41	172.0
35	781	2024-12-10 12:20:45	174.0
36	782	2024-12-10 12:20:49	176.0
37	783	2024-12-10 12:20:53	175.0

Rysunek 8 Generowany raport: Baza danych w formacie pdf



	id	Czas	Moc
0	746	2024-12-10 12:18:19	145.0
1	747	2024-12-10 12:18:23	145.0
2	748	2024-12-10 12:18:28	146.0
3	749	2024-12-10 12:18:32	147.0
4	750	2024-12-10 12:18:37	147.0
5	751	2024-12-10 12:18:41	149.0
6	752	2024-12-10 12:18:45	145.0
7	753	2024-12-10 12:18:49	148.0
8	754	2024-12-10 12:18:53	148.0
9	755	2024-12-10 12:18:57	149.0
10	756	2024-12-10 12:19:01	150.0
11	757	2024-12-10 12:19:05	150.0
12	758	2024-12-10 12:19:10	152.0
13	759	2024-12-10 12:19:14	151.0
14	760	2024-12-10 12:19:18	151.0
15	761	2024-12-10 12:19:22	152.0
16	762	2024-12-10 12:19:26	153.0
17	763	2024-12-10 12:19:30	153.0
18	764	2024-12-10 12:19:34	152.0
19	765	2024-12-10 12:19:38	153.0
20	766	2024-12-10 12:19:42	154.0
21	767	2024-12-10 12:19:46	153.0
22	768	2024-12-10 12:19:51	154.0
23	769	2024-12-10 12:19:54	157.0
24	770	2024-12-10 12:19:59	160.0
25	771	2024-12-10 12:20:03	162.0
26	772	2024-12-10 12:20:07	164.0
27	773	2024-12-10 12:20:12	166.0
28	774	2024-12-10 12:20:16	166.0
29	775	2024-12-10 12:20:20	170.0
30	776	2024-12-10 12:20:24	173.0
31	777	2024-12-10 12:20:28	173.0
32	778	2024-12-10 12:20:32	174.0
33	779	2024-12-10 12:20:37	168.0
34	780	2024-12-10 12:20:41	172.0
35	781	2024-12-10 12:20:45	174.0

Rysunek 9 Generowany raport: Baza danych w formacie HTML

3.1 Opis raportów generowanych w aplikacji SCADA

a) Zawartość raportów

Raporty zawierają:

- **Tabelaryczne dane historyczne o mocy (z tabeli `power_data` w bazie MySQL):**
  - Czas rejestracji (`Czas`).
  - Zarejestrowana moc w kW (`Moc`).
- **Tabelaryczne dane pogodowe (z tabeli `weather_data` w bazie MySQL):**

- Temperatura, wilgotność, prędkość i kierunek wiatru, opady oraz ciśnienie.

➢ **Wykres przebiegu mocy:**
  - Przedstawia zmiany mocy w czasie z ostatnich godzin (rysowany na podstawie danych historycznych).

b) Format raportów

Raporty są generowane w trzech formatach:

➢ **HTML:**

- Zawiera dane w formacie tabel HTML.
- Łatwy do podglądu w przeglądarce.

➢ **PDF:**

- Wysokiej jakości wydruk generowany z pliku HTML przy użyciu biblioteki `pdfkit`.
- Gotowy do archiwizacji lub wysyłki.

➢ **JPG:**

- Wykres zapisany jako obraz graficzny za pomocą `matplotlib`.
- Umożliwia szybkie dzielenie się wizualizacją.

c) Przykłady

- **HTML:** Tabela wygenerowana w raporcie zawiera dane historyczne, np.:
  - Czas: `2024-12-10 12:18:19`
  - Moc: `145.0 kW`.
- **PDF:** Raport w formacie PDF zawiera te same dane co HTML, w eleganckim, ustrukturyzowanym układzie.
- **JPG:** Wykres ilustruje zmienność mocy w określonym przedziale czasowym (widoczny na wcześniejszym przykładzie).

d) Automatyczne nazwy plików

Każdy raport ma nazwę zawierającą datę i godzinę utworzenia, np.:

- `report_2024-12-10_12-42-32.html`
- `report_2024-12-10_12-42-32.pdf`
- `report_2024-12-10_12-42-32.jpg`

4. Opis kodu aplikacji SCADA

Kod aplikacji SCADA został zrealizowany w języku Python z wykorzystaniem następujących technologii i bibliotek:

1. **Tkinter:** Do budowy interfejsu użytkownika, umożliwiającego wizualizację danych i interakcję z użytkownikiem.
2. **Matplotlib:** Do generowania wykresów przedstawiających dane o mocy.
3. **Pandas:** Do przetwarzania i formatowania danych pobranych z bazy danych MySQL.
4. **SQLAlchemy i PyMySQL:** Do zarządzania połączeniem z bazą danych MySQL.
5. **Pdfkit:** Do generowania raportów w formacie PDF.

Główne funkcje i ich opis:

1. **Funkcja `update_data`:**
   - Harmonogramuje okresową aktualizację danych w aplikacji (co 3 sekundy).
   - Wywołuje metody odpowiedzialne za pobieranie danych (`get_power_data`, `get_weather_data`), aktualizację wskaźnika mocy (`update_meter`) oraz odświeżanie wykresu (`update_power_chart`).
2. **Funkcja `get_power_data`:**
   - Pobiera dane o mocy z zewnętrznego API.
   - Przechowuje dane w zmiennej globalnej `power_value` oraz dodaje je do bazy danych MySQL w tabeli `power_data`.
3. **Funkcja `get_weather_data`:**
   - Pobiera dane pogodowe z API IMGW.
   - Wyświetla dane pogodowe w odpowiednich etykietach Tkinter.
   - Zapisuje dane pogodowe do bazy danych w tabeli `weather_data`.
4. **Funkcja `update_power_chart`:**
   - Rysuje wykres przedstawiający zmiany mocy w ciągu ostatnich 12 godzin.
   - Wykorzystuje dane przechowywane w strukturze `deque`.
5. **Funkcja `update_meter`:**
   - Aktualizuje wskaźnik analogowy mocy na podstawie wartości z `power_value`.
   - Wskaźnik zmienia kolor w zależności od wartości procentowej obciążenia:
     - Zielony: Niskie obciążenie.
     - Żółty: Średnie obciążenie.
     - Czerwony: Wysokie obciążenie.
6. **Funkcja `generate_report`:**
   - Pobiera dane historyczne z tabel `power_data` i `weather_data`.
   - Generuje raport w trzech formatach: HTML, PDF i JPG:
     - **HTML:** Formatowanie tabel danych przy użyciu `Pandas`.
     - **PDF:** Konwersja pliku HTML na PDF przy użyciu `Pdfkit`.
     - **JPG:** Wykres zapisany jako obraz za pomocą `Matplotlib`.

7. **Funkcja `clear_database`:**
    - Usuwa dane z tabel `power_data` i `weather_data` w celu resetowania bazy danych.
8. **Funkcja `exit_application`:**
    - Zamyka aplikację SCADA i zwalnia zasoby.

Struktura kodu:

- Kod jest podzielony na moduły odpowiedzialne za różne funkcjonalności:
    - **Interfejs użytkownika:** Użycie Tkinter do wizualizacji wskaźnika mocy, danych pogodowych oraz wykresu.
    - **Zarządzanie bazą danych:** Korzystanie z SQLAlchemy i PyMySQL do zapisu i odczytu danych.
    - **Raportowanie:** Generowanie raportów z użyciem Pandas, Pdfkit i Matplotlib.

Przykład użycia:

1. Aplikacja uruchamia się za pomocą:

```
python
Skopiuj kod
if __name__ == "__main__":
    root = tk.Tk()
    app = ScadaApp(root)
    root.mainloop()
```

2. Po uruchomieniu użytkownik może:
    - Obserwować dane w czasie rzeczywistym (wskaźnik mocy, dane pogodowe).
    - Generować raporty klikając przycisk "Generowanie Raportu".
    - Resetować bazę danych lub zamknąć aplikację za pomocą odpowiednich przycisków.

Kod został zaprojektowany w sposób modularny, co umożliwia łatwe rozszerzenie funkcjonalności o dodatkowe źródła danych lub nowe elementy interfejsu.

4.Podsumowanie i wnioski

Podsumowanie

Projekt aplikacji SCADA został zrealizowany w sposób kompleksowy, integrując zaawansowane technologie i narzędzia programistyczne w celu monitorowania procesów przemysłowych i generowania raportów. System obejmuje kilka kluczowych komponentów:

1. **Interfejs użytkownika:**
   - Wykorzystanie biblioteki Tkinter umożliwiło stworzenie intuicyjnego i przejrzystego panelu wizualizacji procesu. Aplikacja oferuje funkcje monitorowania mocy w czasie rzeczywistym oraz wyświetlanie aktualnych danych pogodowych.
   - Interfejs jest responsywny, umożliwiając użytkownikowi dynamiczne przeglądanie danych, generowanie raportów oraz zarządzanie bazą danych.
2. **Zarządzanie danymi:**
   - Dane o mocy oraz warunkach pogodowych są zapisywane w bazie MySQL, co umożliwia ich późniejszą analizę i raportowanie.
   - Użycie bibliotek PyMySQL i SQLAlchemy zapewniło stabilne i efektywne połączenie z bazą danych, a struktura tabel `power_data` oraz `weather_data` umożliwia przejrzystą organizację danych.
3. **Raportowanie:**
   - Aplikacja generuje raporty w formatach HTML, PDF i JPG, co pozwala na ich wszechstronne zastosowanie w analizach i dokumentacji.
   - Generowane raporty są precyzyjne i czytelne, zawierając zarówno dane tabelaryczne, jak i wizualizacje w postaci wykresów.
4. **Automatyzacja i elastyczność:**
   - Funkcja cyklicznego pobierania danych i aktualizacji wykresów (co 3 sekundy) zapewnia bieżącą aktualność monitorowanego procesu.
   - Kod został zaprojektowany w sposób modularny, co ułatwia jego modyfikację i rozszerzanie o nowe funkcje lub źródła danych.

Wnioski

1. **Skalowalność aplikacji**
   - Projekt może być łatwo rozwijany o dodatkowe moduły, takie jak integracja z innymi systemami SCADA (np. Ignition czy Wonderware) lub wprowadzenie bardziej zaawansowanej analizy danych, np. predykcji awarii systemu na podstawie zgromadzonych danych.
2. **Zastosowanie w przemyśle**
   - Aplikacja jest szczególnie przydatna w środowiskach przemysłowych, gdzie kluczowe jest monitorowanie parametrów procesów w czasie rzeczywistym, takich jak wydajność systemów energetycznych czy warunki pogodowe wpływające na działanie urządzeń.
3. **Elastyczność raportowania**

- Wieloformatowe raporty (HTML, PDF, JPG) pozwalają na ich szerokie zastosowanie, od analizy danych w przeglądarkach internetowych po wysyłkę raportów w formie wydruków czy prezentacji graficznych.

4. **Wysoka użyteczność**
   - Intuicyjny interfejs i zautomatyzowane funkcje (np. harmonogram aktualizacji danych) sprawiają, że aplikacja jest przyjazna dla użytkowników, nawet tych bez zaawansowanej wiedzy technicznej.
5. **Możliwości rozwoju**
   - Integracja z innymi systemami, wprowadzenie alertów w czasie rzeczywistym (np. SMS lub e-mail) czy rozszerzenie o analizę danych historycznych w czasie dłuższym niż 12 godzin mogą zwiększyć użyteczność i zakres aplikacji.
   - Dodanie funkcji eksportu do formatów takich jak Excel czy CSV umożliwi łatwiejszą integrację z innymi systemami analitycznymi.
6. **Efektywność i niezawodność**
   - Pomimo dużej złożoności, aplikacja wykazuje stabilność i efektywność w działaniu. Cykliczne pobieranie i wizualizacja danych, w połączeniu z możliwością ich archiwizacji, zapewniają niezawodność systemu.

Ostateczny wniosek

Aplikacja SCADA jest nowoczesnym, funkcjonalnym i skalowalnym rozwiązaniem, które łączy w sobie prostotę użytkowania z zaawansowanymi funkcjami analizy danych. Może znaleźć zastosowanie w szerokim spektrum procesów przemysłowych, od zarządzania energetyką po monitorowanie warunków środowiskowych. Dzięki elastycznej strukturze kodu i bogatym możliwościom raportowania, system jest gotowy do dalszego rozwoju i adaptacji do specyficznych wymagań użytkowników.

5. Kod Aplikacji SCADA w Python

```
import tkinter as tk
from tkinter import Canvas
import requests
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import urllib3
import pymysql
from collections import deque
import pandas as pd
import pdfkit
import os
from sqlalchemy import create_engine

# Disable InsecureRequestWarning
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

class ScadaApp:
    def __init__(self, root):
        self.root = root
        self.root.title("SCADA App")
        self.root.geometry("1920x1080")

        # Database connection
        self.db_connection = pymysql.connect(
            host="localhost",
            user="root",
```

```
            password="Razer1035.,",
            database="scada_db"
        )

        # Create SQLAlchemy engine
        self.engine = create_engine("mysql+pymysql://root:Razer1035.,@localhost/scada_db")

        # Frame for analog meter
        self.meter_frame = tk.Frame(self.root, width=400, height=300)
        self.meter_frame.grid(row=0, column=0, padx=340, pady=10, sticky='n')

        # Create canvas for analog meter (progress bar-like indicator)
        self.canvas = Canvas(self.meter_frame, width=400, height=300, bg='white')
        self.canvas.pack(expand=True)

        # Initial value for power
        self.power_value = 0.0

        # Frame for weather information
        self.weather_frame = tk.Frame(self.root, width=400, height=300)
        self.weather_frame.grid(row=1, column=0, padx=340, pady=10, sticky='n')

        # Labels to display weather data
        self.temperature_label = tk.Label(self.weather_frame, text="Temperatura: N/A",
font=("Arial", 16))
        self.temperature_label.pack(anchor='center')
        self.wind_speed_label = tk.Label(self.weather_frame, text="Prędkość wiatru: N/A",
font=("Arial", 16))
        self.wind_speed_label.pack(anchor='center')
        self.wind_direction_label = tk.Label(self.weather_frame, text="Kierunek wiatru: N/A",
font=("Arial", 16))
        self.wind_direction_label.pack(anchor='center')
        self.humidity_label = tk.Label(self.weather_frame, text="Wilgotność: N/A", font=("Arial",
16))
        self.humidity_label.pack(anchor='center')
        self.precipitation_label = tk.Label(self.weather_frame, text="Opady: N/A", font=("Arial",
16))
        self.precipitation_label.pack(anchor='center')
        self.pressure_label = tk.Label(self.weather_frame, text="Ciśnienie: N/A", font=("Arial",
16))
        self.pressure_label.pack(anchor='center')

        # Create frame for power chart
        self.plot_frame = tk.Frame(self.root, width=400, height=300)
        self.plot_frame.grid(row=2, column=0, padx=340, pady=10, sticky='n')

        # Create matplotlib figure for power chart
        self.figure = Figure(figsize=(10, 4), dpi=100)
        self.ax = self.figure.add_subplot(111)
        self.ax.set_title("Moc w ciągu ostatnich 12 godzin")
        self.ax.set_xlabel("Czas")
        self.ax.set_ylabel("Moc (kW)")
        self.ax.set_yticks([2000, 4000, 6000, 8000])  # Set constant y-axis values
        self.ax.set_ylim(0, 8000)  # Fix the y-axis limits to prevent scaling
        self.ax.grid(True)  # Add grid to the chart
        self.power_data = deque(maxlen=480)  # Initialize deque with a maximum length
        self.time_data = deque(maxlen=480)  # Store corresponding time values
        self.chart = FigureCanvasTkAgg(self.figure, master=self.plot_frame)
        self.chart.get_tk_widget().pack(expand=True)

        # Create frame for control buttons
        self.control_frame = tk.Frame(self.root, width=200, height=300)
        self.control_frame.grid(row=0, column=1, rowspan=6, padx=20, pady=10, sticky='n')

        # Button to generate report
        self.report_button = tk.Button(self.control_frame, text="Generowanie Raportu",
command=self.generate_report)
        self.report_button.pack(pady=10)

        # Button to clear database
        self.clear_db_button = tk.Button(self.control_frame, text="Reset Bazy Danych",
command=self.clear_database)
        self.clear_db_button.pack(pady=10)
```

```
        # Button to exit the application
        self.exit_button = tk.Button(self.control_frame, text="Zamknij",
command=self.exit_application)
        self.exit_button.pack(pady=10)

        # Start the data update loop
        self.update_data()

    def update_data(self):
        # Fetch data and update charts and UI
        self.get_power_data()
        self.get_weather_data()
        self.update_meter()
        self.update_power_chart()

        # Schedule the next update (every 3 seconds)
        self.root.after(3000, self.update_data)

    def update_power_chart(self):
        self.ax.clear()
        self.ax.set_title("Przebieg Mocy Baterii Fotowoltaicznej")
        self.ax.set_xlabel("Czas")
        self.ax.set_ylabel("Moc (kW)")
        self.ax.set_yticks([2000, 4000, 6000, 8000])  # Set constant y-axis values
        self.ax.set_ylim(0, 8000)  # Fix the y-axis limits to prevent scaling
        self.ax.grid(True)  # Add grid to the chart

        # Plot the filtered data
        if self.power_data:
            self.ax.plot(self.time_data, self.power_data, color='Slateblue', alpha=0.6)
            self.ax.set_xlim([datetime.now() - timedelta(hours=12), datetime.now()])
        self.chart.draw()

    def update_meter(self):
        self.canvas.delete("all")
        percentage = min(100, max(0, (self.power_value / 8000) * 100))
        color = "red" if percentage >= 75 else "yellow" if percentage >= 50 else "green" if
percentage >= 25 else "grey"
        self.canvas.create_rectangle(50, 100, 350, 200, fill="lightgrey", outline="black")
        self.canvas.create_rectangle(50, 100, 50 + 3 * percentage, 200, fill=color,
outline="black")
        self.canvas.create_text(200, 150, text=f"{int(percentage)}%", font=("Arial", 16))
        self.canvas.create_text(200, 250, text=f"Power: {int(self.power_value)} W",
font=("Arial", 16))

    def get_power_data(self):
        try:
            response = requests.get("https://91.233.250.151:5555/dyn/getDashValues.json",
verify=False, timeout=5)
            if response.status_code == 200:
                data = response.json()
                power_data = data.get("result", {}).get("0198-xxxxx100", {}).get("6100_40263F00",
{}).get("1", [])[0]
                if "val" in power_data:
                    self.power_value = power_data["val"]
                    now = datetime.now()
                    self.power_data.append(self.power_value)
                    self.time_data.append(now)

                    # Insert data into the database
                    with self.db_connection.cursor() as cursor:
                        sql = "INSERT INTO power_data (Czas, Moc) VALUES (%s, %s)"
                        cursor.execute(sql, (now, self.power_value))
                        self.db_connection.commit()
        except Exception as e:
            print(f"Error fetching power data: {e}")

    def get_weather_data(self):
        try:
            url = "https://danepubliczne.imgw.pl/api/data/synop"
            response = requests.get(url)
            if response.status_code == 200:
                data = response.json()
                for station_data in data:
```

```
                    if station_data['stacja'] == 'Katowice':
                        temperature = station_data.get('temperatura', None)
                        wind_speed = station_data.get('predkosc_wiatru', None)
                        wind_direction = station_data.get('kierunek_wiatru', None)
                        humidity = station_data.get('wilgotnosc_wzgledna', None)
                        precipitation = station_data.get('suma_opadu', None)
                        pressure = station_data.get('cisnienie', None)

                        self.temperature_label.config(text=f"Temperatura: {temperature}°C")
                        self.wind_speed_label.config(text=f"Prędkość wiatru: {wind_speed} km/h")
                        self.wind_direction_label.config(text=f"Kierunek wiatru:
{wind_direction}°")
                        self.humidity_label.config(text=f"Wilgotność: {humidity}%")
                        self.precipitation_label.config(text=f"Opady: {precipitation} mm")
                        self.pressure_label.config(text=f"Ciśnienie: {pressure} hPa")

                        # Insert data into the database
                        now = datetime.now()
                        with self.db_connection.cursor() as cursor:
                            sql = """
                                INSERT INTO weather_data (
                                    Czas, temperatura, predkosc_wiatru, kierunek_wiatru,
                                    wilgotnosc, opady, cisnienie
                                ) VALUES (%s, %s, %s, %s, %s, %s, %s)
                            """
                            cursor.execute(sql, (now, temperature, wind_speed, wind_direction,
                                                 humidity, precipitation, pressure))
                            self.db_connection.commit()
                        break
            except Exception as e:
                print(f"Error fetching weather data: {e}")


    def generate_report(self):
        try:
            # Get the current date and time for the filename
            timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

            # Connect to the database using SQLAlchemy
            df_power = pd.read_sql("SELECT * FROM power_data", self.engine)
            df_weather = pd.read_sql("SELECT * FROM weather_data", self.engine)

            # Generate HTML report
            html_content = df_power.to_html() + "<br><br>" + df_weather.to_html()
            html_filename = f"report_{timestamp}.html"
            with open(html_filename, "w") as html_file:
                html_file.write(html_content)

            # Path to wkhtmltopdf
            path_to_wkhtmltopdf = r"C:\Program Files\wkhtmltopdf\bin\wkhtmltopdf.exe"
            pdfkit_config = pdfkit.configuration(wkhtmltopdf=path_to_wkhtmltopdf)

            # Convert HTML to PDF
            pdf_filename = f"report_{timestamp}.pdf"
            pdfkit.from_file(html_filename, pdf_filename, configuration=pdfkit_config)

            # Create and save chart as JPG
            jpg_filename = f"report_{timestamp}.jpg"
            plt.figure(figsize=(10, 6))
            plt.plot(df_power['Czas'], df_power['Moc'], label="Moc (kW)", color='blue')
            plt.xlabel('Czas')
            plt.ylabel('Moc (kW)')
            plt.title('Przebieg Mocy')
            plt.grid(True)
            plt.savefig(jpg_filename)
            plt.close()

            print(f"Report generated successfully:\nHTML: {html_filename}\nPDF:
{pdf_filename}\nJPG: {jpg_filename}")

        except Exception as e:
            print(f"Error generating report: {e}")
```

```
    def clear_database(self):
        try:
            with self.db_connection.cursor() as cursor:
                cursor.execute("DELETE FROM power_data")
                cursor.execute("DELETE FROM weather_data")
                self.db_connection.commit()
            print("Database cleared successfully.")
        except Exception as e:
            print(f"Error clearing database: {e}")

    def exit_application(self):
        self.root.destroy()
        print("Application closed.")

if __name__ == "__main__":
    root = tk.Tk()
    app = ScadaApp(root)
    root.mainloop()
```