

# AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych  
Katedra Informatyki

## DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

### **Implementacja wczytywania danych z pliku CSV w C++**

Autor:

Kamil Gruca  
Mateusz Basiaga

Prowadzący:

mgr inż. Dawid Kotlarski

Nowy Sącz 2024

# Spis treści

<b>1. Ogólne określenie wymagań</b>	<b>3</b>
1.1. Cel pracy . . . . .	3
1.2. Zakładane wyniki . . . . .	4
<b>2. Analiza problemu</b>	<b>5</b>
2.1. Parsowanie i przygotowanie danych . . . . .	5
2.2. Sposób na rozwiązanie problemu . . . . .	6
2.3. Ograniczenia i wyzwania . . . . .	8
2.4. Wykorzystanie aplikacji w praktyce . . . . .	9
2.5. Możliwości rozwoju i przyszłe rozszerzenia . . . . .	10
<b>3. Projektowanie</b>	<b>11</b>
3.1. Język programowania wykorzystany w projekcie . . . . .	11
3.2. Narzędzia i technologie wykorzystane w projekcie . . . . .	11
3.3. Zarządzanie projektem i współpraca w zespole . . . . .	12
<b>4. Implementacja</b>	<b>13</b>
4.1. Struktura programu . . . . .	13
4.2. Rozłożenie metody <code>LineData::serialize(ofstream&amp; out) const</code> . . . . .	14
4.3. Rozłożenie metody <code>Logger::Logger(const std::string&amp; filename)</code> . . . . .	15
4.4. Rozłożenie metody <code>TreeData::addData(const LineData&amp; lineData)</code> . . . . .	16
4.5. Działanie programu . . . . .	17
4.6. Testy programu . . . . .	20
4.7. Pomoc AI w projekcie . . . . .	21
<b>5. Wnioski</b>	<b>22</b>
<b>Literatura</b>	<b>23</b>
<b>Spis rysunków</b>	<b>24</b>
<b>Spis listingów</b>	<b>25</b>

# 1. Ogólne określenie wymagań

Celem projektu jest implementacja i przetestowanie programu w języku C++ umożliwiającego analizę danych zawartych w pliku CSV. Projekt obejmuje stworzenie modułowego systemu, który umożliwia wczytywanie danych do pamięci komputera, ich przetwarzanie oraz analizę według określonych kryteriów. Głównym założeniem jest zaimplementowanie mechanizmów pozwalających na grupowanie danych w strukturze drzewa, gdzie poszczególne węzły odpowiadają hierarchii czasowej (rok, miesiąc, dzień, ćwiartka doby).

## 1.1. Cel pracy

Głównym celem projektu jest:

- Implementacja programu do analizy danych z pliku CSV w języku C++ z wykorzystaniem struktury modularnej, gdzie każda klasa i funkcjonalność są umieszczone w oddzielnych plikach źródłowych.
- Zastosowanie struktury drzewa hierarchicznego do organizacji danych według jednostek czasowych (rok, miesiąc, dzień, ćwiartka doby) oraz zaimplementowanie wzorca projektowego iterator do poruszania się po drzewie.
- Opracowanie mechanizmów obliczeniowych umożliwiających użytkownikowi wykonywanie operacji takich jak obliczanie sum, średnich, porównań oraz wyszukiwanie danych w określonych przedziałach czasowych.
- Zabezpieczenie programu przed błędami w plikach CSV poprzez bieżącą analizę danych, odrzucanie niepoprawnych rekordów i tworzenie logów z procesu wczytywania.
- Przeprowadzenie testów jednostkowych za pomocą frameworka GoogleTest w celu zapewnienia poprawności działania programu w różnych scenariuszach.
- Opracowanie szczegółowej dokumentacji technicznej projektu przy użyciu narzędzi LaTeX i Doxygen oraz publikacja kodu źródłowego w serwisie GitHub.
- Stworzenie interaktywnego interfejsu użytkownika, który pozwala na wybór operacji z poziomu menu, w tym wczytywanie danych, analizę oraz ich zapis i odczyt w formacie binarnym.

## 1.2. Zakładane wyniki

Oczekiwane efekty programu to:

- Poprawna i wydajna implementacja programu do analizy danych z pliku CSV, który umożliwia wczytywanie, organizowanie i przetwarzanie danych w hierarchicznej strukturze drzewa z uwzględnieniem jednostek czasowych.
- Mechanizmy obliczeniowe pozwalające na wykonywanie operacji takich jak obliczanie sum, średnich, porównań oraz wyszukiwanie rekordów zgodnych z podanymi kryteriami, w tym tolerancji błędu i zakresu czasowego.
- Zbiór testów jednostkowych pokrywających kluczowe funkcjonalności programu, w tym obsługę poprawnych i niepoprawnych danych, dokładność wyników analiz oraz zachowanie programu w nietypowych scenariuszach, takich jak brak ciągłości czasowej w danych.
- Interaktywne menu użytkownika, które pozwala na wybór i realizację operacji, takich jak analiza danych, zapis i odczyt w formacie binarnym oraz wczytywanie danych z plików CSV.
- Szczegółowa dokumentacja techniczna i użytkownika opracowana w narzędziach LaTeX i Doxygen, obejmująca opis struktury programu, założeń projektowych i instrukcji obsługi.
- Repozytorium projektu w serwisie GitHub, zawierające kod źródłowy, testy jednostkowe oraz konfigurację do automatyzacji budowania i testowania za pomocą CMake i GoogleTest.

## 2. Analiza problemu

Analizowany problem dotyczy przetwarzania, organizacji i analizy danych energetycznych zapisanych w formacie CSV. Dane te zawierają kluczowe informacje dotyczące autokonsumpcji, eksportu, importu, poboru oraz produkcji energii w wybranych przedziałach czasowych. Celem projektu jest stworzenie wszechstronnej aplikacji, która umożliwi efektywne zarządzanie danymi, organizując je w sposób hierarchiczny, a także zapewni użytkownikowi narzędzia do ich analizy i raportowania. W centrum problemu leży potrzeba radzenia sobie z dużymi zbiorami danych o różnorodnej jakości i strukturze. Program ma być odporny na typowe problemy związane z nieprzewidywalnością danych wejściowych, takie jak brak ciągłości czasowej, powtarzające się rekordy, puste linie czy brakujące wartości. Dodatkowo, projekt wymaga implementacji struktur danych, które umożliwią szybkie i elastyczne przetwarzanie danych w wybranych przedziałach czasowych, zgodnie z założeniami organizacji w hierarchiczne drzewo.

### 2.1. Parsowanie i przygotowanie danych

Prasowanie danych (ang. data wrangling) jest kluczowym etapem w procesie przygotowania danych do analizy, szczególnie w przypadku dużych zbiorów danych, takich jak te, które są przechowywane w formacie CSV. W kontekście tego projektu, parsowanie danych obejmuje szereg czynności mających na celu oczyszczenie, przekształcenie i zorganizowanie surowych danych energetycznych w sposób umożliwiający ich efektywne przetwarzanie oraz analizę. Dla tego projektu kluczowe aspekty prasowania danych to:

- **Obsługa brakujących danych** - Pliki CSV mogą zawierać rekordy, w których brakuje niektóre wartości, np. dane o autokonsumpcji, eksporcie czy produkcji energii. W takich przypadkach konieczne jest wdrożenie metod ich obsługi, takich jak usuwanie wierszy z brakującymi danymi lub ich uzupełnianie na podstawie dostępnych informacji, np. przy użyciu interpolacji.
- **Eliminowanie duplikatów** - W procesie wczytywania danych mogą występować powtarzające się wiersze, które prowadzą do błędnych obliczeń lub zafałszowanej analizy. Dlatego ważne jest, by aplikacja automatycznie identyfikowała i usuwała duplikaty z pliku CSV, aby zapewnić spójność i dokładność danych.

- **Walidacja danych** - Wiele plików CSV zawiera dane, które mogą być zapisane w nieodpowiednim formacie lub zawierać błędne informacje, np. błędne daty, godziny spoza dozwolonych przedziałów czy wartości liczbowe w złym formacie. Program musi sprawdzać poprawność danych i odrzucać te, które nie spełniają wymagań określonych w specyfikacji. Przykładem może być sprawdzenie, czy godziny pomiaru są w poprawnym formacie 24-godzinnym lub upewnienie się, że wartości dotyczące autokonsumpcji nie są ujemne.
- **Logowanie i raportowanie błędów** - Ważnym elementem prasowania danych jest także tworzenie raportów o błędach, które umożliwiają analizę nieprawidłowości w plikach CSV. Przykładowo, system powinien zapisywać informacje o odrzuconych rekordach, wskazując przyczyny ich odrzucenia (np. brakujące dane, błędne formaty). Taki raport może być pomocny zarówno w diagnostyce problemów, jak i w przyszłym doskonaleniu procesu przetwarzania danych.

Parsowanie danych jest więc kluczowym etapem w całym procesie analizy, który umożliwia dalsze, zaawansowane operacje na danych. Dzięki odpowiedniemu oczyszczeniu i organizacji danych, aplikacja będzie w stanie działać szybko, precyzyjnie i bezbłędnie, oferując użytkownikowi wysoką jakość wyników analizy.

## 2.2. Sposób na rozwiązanie problemu

W celu rozwiązania problemu przyjęto następujące założenia:

### Format danych wejściowych

Dane w pliku CSV składają się z kolumn:

- Data i godzina pomiaru,
- Autokonsumpcja (energia zużywana na bieżąco),
- Eksport (energia wysyłana do sieci),
- Import (energia pobierana z sieci),
- Pobór (całkowita energia zużyta),
- Produkcja (energia wytworzona przez falownik).

### **Struktura organizacji danych**

Dane mają być przechowywane w strukturze drzewa, której poziomy odpowiadają kolejno:

- Rokowi,
- Miesiącowi,
- Dniowi,
- Ćwiartkom dnia (00:00–5:45, 6:00–11:45, 12:00–17:45, 18:00–23:45).

Drzewo powinno umożliwiać dostęp do danych w dowolnym przedziale czasowym.

### **Obsługa danych wejściowych**

Program musi obsługiwać dane o nieprzewidywalnej strukturze, w tym:

- Puste linie,
- Niekompletne rekordy,
- Powtarzające się wiersze.

Wszystkie błędne dane powinny być odrzucane, a ich analiza powinna być logowana w dwóch plikach: jeden dla pełnego raportu procesu, a drugi dla szczegółowego opisu błędów.

### **Funkcjonalność aplikacji**

Program ma oferować użytkownikowi możliwość interakcji za pomocą menu, które obejmuje:

- Wczytywanie danych z pliku CSV,
- Zapisywanie i odczytywanie danych w formacie binarnym,
- Analizę danych, w tym obliczanie sum, średnich i porównania wartości w różnych przedziałach czasowych,
- Filtrowanie rekordów na podstawie określonych kryteriów.

## 2.3. Ograniczenia i wyzwania

W trakcie analizy zidentyfikowano potencjalne ograniczenia i trudności:

### Wydajność:

- Przetwarzanie dużych plików CSV może wymagać znacznych zasobów pamięci i mocy obliczeniowej.
- Konieczne jest zaimplementowanie wydajnych algorytmów oraz odpowiedniego zarządzania pamięcią, aby uniknąć spowolnień.

### Nieprzewidywalność danych:

- Dane mogą być niekompletne, powtarzające się lub zapisane w różnej kolejności, co wymaga dokładnej walidacji i przetwarzania.

### Złożoność struktury drzewa:

- Implementacja drzewa z wieloma poziomami wymaga precyzyjnego projektowania, szczególnie w przypadku braku ciągłości czasowej w danych.

### Synchronizacja modułów:

- Zapewnienie, że różne moduły programu (wczytywanie danych, operacje na drzewie, zapisywanie do plików) działają spójnie, może być wyzwaniem.

### Obsługa błędów:

- Program musi być odporny na awarie wynikające z nieprawidłowych danych, jednocześnie generując szczegółowe raporty o błędach.



## 2.4. Wykorzystanie aplikacji w praktyce

### **Zarządzanie energią w budynkach komercyjnych i mieszkalnych:**

Aplikacja może być wykorzystana w inteligentnych budynkach (tzw. smart buildings) do monitorowania i optymalizacji zużycia energii.

- Monitorowanie autokonsumpcji - Analiza, ile energii zużywa budynek w czasie rzeczywistym, np. dla budynków wyposażonych w panele fotowoltaiczne.
- Optymalizacja zużycia - Identyfikacja okresów szczytowego poboru energii, co pozwala na dostosowanie harmonogramu pracy urządzeń elektrycznych.
- Raportowanie kosztów - Generowanie raportów zużycia i oszczędności, co pomaga użytkownikom indywidualnym i przedsiębiorstwom kontrolować koszty energii.

### **Zarządzanie farmami fotowoltaicznymi i wiatrowymi:**

Farmy produkujące energię odnawialną wymagają precyzyjnych narzędzi do monitorowania i analizy danych.

- Monitorowanie wydajności paneli fotowoltaicznych - Aplikacja może pomóc w analizie, jak skutecznie działają panele w różnych warunkach pogodowych.
- Zarządzanie eksportem energii do sieci - Umożliwia optymalne zarządzanie przesyłem energii do sieci energetycznej i minimalizację strat.
- Identyfikacja usterek - Szybkie wykrywanie problemów, takich jak niesprawne panele czy turbiny, na podstawie analizy odchyleń w danych energetycznych.

### **Energetyka i dostawcy energii:**

Dostawcy energii mogą wykorzystać aplikację do monitorowania przepływów energii i optymalizacji jej dystrybucji.

- Analiza eksportu i importu energii - Umożliwia ocenę, ile energii jest przesyłane między różnymi punktami sieci, oraz identyfikację miejsc przeciążenia sieci.
- Prognozowanie zapotrzebowania na energię - Dzięki analizie historycznych danych dostawcy mogą przewidywać zapotrzebowanie na energię w różnych porach dnia i roku.

## 2.5. Możliwości rozwoju i przyszłe rozszerzenia

Projekt ma potencjał do dalszego rozwoju i implementacji dodatkowych funkcji w przyszłości, które mogą przyczynić się do jeszcze lepszego zarządzania danymi energetycznymi. Potencjalne kierunki rozwoju obejmują:

- **Integracja z systemami IoT** - Możliwość łączenia aplikacji z urządzeniami IoT, takimi jak inteligentne liczniki energii, czujniki czy panele fotowoltaiczne, pozwoli na zbieranie danych w czasie rzeczywistym oraz ich natychmiastową analizę, co umożliwi szybszą reakcję na zmiany w zużyciu energii.
- **Predykcja i sztuczna inteligencja** - W przyszłości aplikacja może być rozbudowana o moduł predykcji zapotrzebowania na energię na podstawie analizy dużych zbiorów danych. Sztuczna inteligencja (AI) mogłaby przewidywać wzorce zużycia energii, identyfikować anomalie i proponować optymalizację w czasie rzeczywistym.
- **Zwiększenie wydajności algorytmów przetwarzania danych** - Ze względu na rosnącą ilość danych energetycznych, możliwe jest zaimplementowanie algorytmów opartych na technologiach rozproszonego przetwarzania, takich jak obliczenia w chmurze, co pozwoli na skalowanie aplikacji do obsługi jeszcze większych zbiorów danych.
- **Rozszerzenie formatu wejściowego** - Rozważenie dodania obsługi innych formatów danych wejściowych (np. JSON, XML), co pozwoli na szersze wykorzystanie aplikacji w różnych środowiskach i z różnymi źródłami danych.

## 3. Projektowanie

### 3.1. Język programowania wykorzystany w projekcie

W projekcie, zgodnie z wymaganiami, zostanie wykorzystany język C++. Język ten jest idealnym wyborem do implementacji złożonych aplikacji przetwarzających duże zbiory danych, takich jak hierarchiczne struktury drzewa przechowujące dane energetyczne. Dzięki możliwości manualnego zarządzania pamięcią, C++ pozwala na efektywne operowanie na danych o dużej objętości, co jest szczególnie istotne w przypadku analizy informacji zawartych w plikach CSV, które mogą obejmować tysiące lub miliony rekordów. C++ oferuje również zaawansowane wsparcie dla programowania obiektowego, co umożliwi stworzenie klas odpowiadających za węzły drzewa, operacje na danych oraz obsługę plików wejściowych i wyjściowych. Dzięki temu kod będzie modularny, przejrzysty i łatwiejszy do rozbudowy w przyszłości.

### 3.2. Narzędzia i technologie wykorzystane w projekcie

W projekcie wykorzystano następujące narzędzia i technologie, które wspierają zarówno rozwój aplikacji, jak i zapewniają jej odpowiednią jakość oraz funkcjonalność:

- **Google Test** - Framework do testowania jednostkowego, który umożliwia automatyczne testowanie poprawności działania aplikacji w różnych scenariuszach. Google Test pozwala na łatwą integrację z systemami Continuous Integration (CI), co zapewnia automatyczne uruchamianie testów na każdym etapie rozwoju projektu. W projekcie służy do weryfikacji poprawności przetwarzania danych oraz testowania funkcji operujących na strukturze drzewa.
- **CMake** - Narzędzie do konfiguracji procesu budowania projektu. CMake generuje odpowiednie pliki konfiguracyjne dla różnych systemów operacyjnych i kompilatorów, co zapewnia elastyczność i łatwość w budowaniu aplikacji na różnych platformach, takich jak Windows i Linux. Dzięki CMake możliwe jest utrzymanie jednej, spójnej konfiguracji procesu budowy, co upraszcza zarządzanie projektem.
- **Ninja** - Lekki system budowania, który współpracuje z CMake i zapewnia szybkie kompilacje. Ninja został wybrany ze względu na mniejsze zużycie zasobów systemowych w porównaniu do tradycyjnych systemów budowania, co znacząco przyspiesza proces kompilacji i umożliwia efektywne zarządzanie czasem w trakcie rozwijania projektu.

- **Doxygen** - Narzędzie do generowania dokumentacji na podstawie komentarzy w kodzie źródłowym. Doxygen automatycznie tworzy dokumentację w formacie HTML, co pozwala na łatwiejsze zrozumienie struktury i działania kodu. Dokumentacja generowana przez Doxygen jest dostępna na stronie projektu, co umożliwia łatwiejsze przeglądanie i korzystanie z niej przez przyszłych programistów oraz użytkowników.
- **GitHub Actions** - Platforma CI/CD, która umożliwia automatyczne uruchamianie testów, generowanie dokumentacji oraz wdrażanie aplikacji na repozytorium GitHub. GitHub Actions w tym projekcie służy do uruchamiania testów jednostkowych, weryfikacji konwencji commitów, generowania dokumentacji oraz publikacji wersji na GitHub Pages. Dzięki tej platformie cały proces rozwoju aplikacji jest zautomatyzowany, co znacząco zwiększa efektywność i zapewnia jakość kodu.
- **Github Copilot** - to narzędzie oparte na sztucznej inteligencji, które zostało stworzone przez GitHub we współpracy z OpenAI. Jego głównym celem jest wspomaganie programistów w codziennej pracy poprzez automatyczne sugerowanie fragmentów kodu, funkcji czy całych bloków logicznych na podstawie kontekstu i opisów wprowadzonych przez użytkownika. Copilot działa jako rozszerzenie do popularnych edytorów kodu, takich jak Visual Studio Code, i jest w stanie uczyć się na podstawie istniejącego kodu, co pozwala na dostosowywanie sugestii do bieżącego projektu.

### 3.3. Zarządzanie projektem i współpraca w zespole

W projekcie szczególną uwagę poświęcono efektywnemu zarządzaniu procesem programowania oraz współpracy w zespole. Dzięki zastosowaniu systemu kontroli wersji Git oraz platformy GitHub, udało się zapewnić pełną śledzenie zmian w kodzie oraz umożliwić współpracę wielu programistów nad różnymi elementami aplikacji w sposób zorganizowany i bezpieczny.

## 4. Implementacja

### 4.1. Struktura programu

Program, jest narzędziem do zarządzania danymi w strukturze drzewa, które może być ładowane, analizowane, przetwarzane i zapisywane w różnych formatach. Służy do analizy danych związanych z różnymi parametrami (takimi jak autokonsumpcja, eksport, import, produkcja), przechowywanych w plikach CSV i plikach binarnych. Aplikacja pozwala użytkownikowi na wykonywanie różnych operacji na tych danych, takich jak obliczenia, wyszukiwanie, porównywanie oraz zapisywanie wyników w formie plików. Pliki które odpowiadają za działanie programu:

#### **Plik app.cpp**

Odpowiada za interakcję z użytkownikiem i wywoływanie odpowiednich funkcji. Użytkownik wybiera opcje z menu, a aplikacja odpowiada na te wybory, wykonując odpowiednie operacje na danych (np. ładowanie danych, obliczenia).

#### **Plik lineData.cpp**

Plik ten implementuje funkcje zadeklarowane w nagłówku lineData.hpp. Główne funkcje to konstrukcja obiektów LineData z danych wejściowych (zarówno z pliku tekstowego, jak i z pliku binarnego) oraz operacje na tych obiektach.

#### **Plik logger.cpp**

Plik ten zawiera klasę Logger, która jest odpowiedzialna za rejestrowanie logów działania programu (zarówno informacji ogólnych, jak i błędów). Zawiera dwie instancje loggera: logger (do logów ogólnych) oraz loggerError (do logów błędów).

#### **Plik treeData.cpp**

Plik ten zawiera klasę TreeData, która jest odpowiedzialna za zarządzanie strukturą danych w postaci drzewa. Drzewo to składa się z lat, miesięcy, dni i kwartałów, a każdy kwartał zawiera dane związane z pojedynczymi liniami danych. Plik treeData.cpp implementuje funkcje, które umożliwiają dodawanie danych, drukowanie ich, obliczanie sum, średnich, porównywanie danych pomiędzy okresami i inne operacje.

## 4.2. Rozłożenie metody `LineData::serialize(ofstream& out) const`

Metoda jest odpowiedzialna za zapisanie wszystkich danych obiektu `LineData` do pliku w formacie binarnym, w tym rozmiaru i zawartości daty oraz wartości zmiennoprzecinkowych (autokonsumpcji, eksportu, importu, poboru i produkcji), co umożliwia efektywne przechowywanie i późniejsze odczytywanie tych danych. Metoda została przedstawiona na Listingu nr. 1 (s. 14):

```
1 void LineData::serialize(ofstream& out) const {  
2     size_t dateSize = date.size();  
3     out.write(reinterpret_cast<const char*>(&dateSize), sizeof(  
4     dateSize));  
5     out.write(date.c_str(), dateSize);  
6     out.write(reinterpret_cast<const char*>(&autokonsumpcja),  
7     sizeof(autokonsumpcja));  
8     out.write(reinterpret_cast<const char*>(&eksport), sizeof(  
9     eksport));  
10    out.write(reinterpret_cast<const char*>(&import), sizeof(import));  
11    out.write(reinterpret_cast<const char*>(&pobor), sizeof(pobor));  
12    out.write(reinterpret_cast<const char*>(&produkcja), sizeof(  
13    produkcja));  
14 }
```

**Listing 1.** `LineData::serialize(ofstream& out) const`

Opis metody przedstawionej na listingu nr. 1 (s. 14):

- Kod oblicza długość ciągu `date` (czyli liczbę znaków w dacie) i zapisuje tę długość do pliku binarnego. Zmienna `dateSize` jest typu `size_t` i przechowuje liczbę znaków w zmiennej `date`. Funkcja `out.write()` zapisuje tę liczbę jako ciąg bajtów do pliku. (Linia 2,3)
- Wiersze kodu zapisują zmienne zmiennoprzecinkowe do pliku binarnego. Zmienna jest przekonwertowana na wskaźnik, aby można było zapisać jej zawartość jako ciąg bajtów w formacie binarnym. Każda zmienna jest zapisywana z użyciem `sizeof()`, co zapewnia, że odpowiednia ilość bajtów jest zapisywana do pliku. (Linia 4-9)

### 4.3. Rozłożenie metody `Logger::Logger(const std::string& filename)`

Metoda ta jest odpowiedzialna za inicjalizację obiektu klasy `Logger`, tworzenie pliku logu z nazwą zawierającą datę i czas oraz otwieranie tego pliku do zapisu. Jeżeli plik o nazwie wynikowej już istnieje, zostaje usunięty przed otwarciem nowego pliku. Jeśli nie uda się otworzyć pliku, metoda zgłasza wyjątek. Metoda ta została przedstawiona na Listingu nr. 2 (s. 15)

```

1 Logger::Logger(const std::string& filename) {
2     auto t = std::time(nullptr);
3     std::tm tm;
4     localtime_s(&tm, &t);
5     std::ostringstream oss;
6     oss << filename << "_" << std::put_time(&tm, "%d%m%Y_%H%M%S")
7     << ".txt";
8     std::string datedFilename = oss.str();
9     if (std::remove(datedFilename.c_str()) != 0) {
10    }
11    logfile.open(datedFilename, std::ios::out | std::ios::app);
12    if (!logfile.is_open()) {
13        throw std::runtime_error("Nie można otworzyć logów");
14    }
15 }
```

**Listing 2.** `Logger::Logger(const std::string& filename)`

#### Opis metody przedstawionej na Listingu nr. 2 (s. 15)

- W pierwszym kroku metoda uzyskuje aktualny czas systemowy, a następnie zamienia ten czas na strukturę `tm` reprezentującą lokalny czas. (Linia 2-4)
- Używając klasy `std::ostringstream`, generowana jest nazwa pliku, która składa się z podstawowej nazwy (przekazanej jako parametr `filename`), a następnie dodawany jest ciąg daty i czasu. Cała ta nazwa jest następnie konwertowana na ciąg znaków, który będzie używany do utworzenia pliku logu. (Linia 5-7)
- Przed otwarciem nowego pliku, metoda sprawdza, czy plik o wygenerowanej nazwie już istnieje. Jeśli tak, jest on usuwany. (Linia 8,9)
- Następnie, plik logu jest otwierany w trybie dopisania i zapisu. Jeśli otwarcie pliku się nie powiedzie, rzucany jest wyjątek z odpowiednią wiadomością błędu. (Linia 10-12)

#### 4.4. Rozłożenie metody `TreeData::addData(const LineData& lineData)`

Metoda ta jest odpowiedzialna za dodanie nowych danych (obiekt `LineData`) do odpowiedniej struktury danych w drzewie (struktura `years`), przechowując dane zorganizowane według roku, miesiąca, dnia i kwartału. Na podstawie daty zawartej w obiekcie `lineData` metoda dzieli datę na poszczególne elementy (rok, miesiąc, dzień, godzina, minuta) i przypisuje dane do odpowiednich węzłów drzewa. Metoda ta została przedstawiona na Listingu nr. 3 (s. 16)

```

1 void TreeData::addData(const LineData& lineData) {
2     stringstream ss(lineData.getDate());
3     string token;
4     vector<int> dateParts;
5
6     while (getline(ss, token, '.')) {
7         dateParts.push_back(stoi(token));
8     }
9
10    int year = dateParts[2];
11    int month = dateParts[1];
12    int day = dateParts[0];
13    int hour = stoi(lineData.getDate().substr(11, 2));
14    int minute = stoi(lineData.getDate().substr(14, 2));
15    int quarter = (hour * 60 + minute) / 360;
16
17    years[year].year = year;
18    years[year].months[month].month = month;
19    years[year].months[month].days[day].day = day;
20    years[year].months[month].days[day].quarters[quarter].quarter =
    quarter;
21    years[year].months[month].days[day].quarters[quarter].hour =
    hour;
22    years[year].months[month].days[day].quarters[quarter].minute =
    minute;
23    years[year].months[month].days[day].quarters[quarter].data.
    push_back(lineData);
24 }

```

**Listing 3.** Metoda `double oblicz_pi`

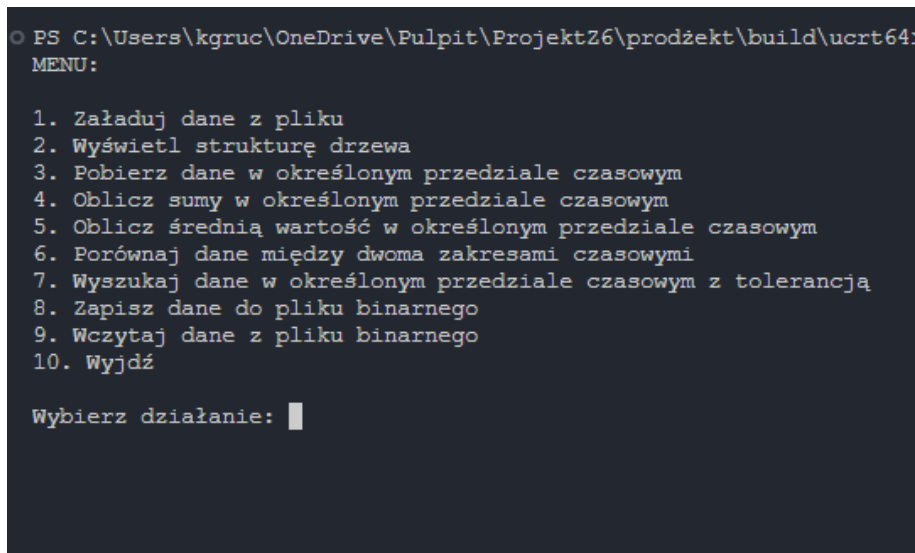


**Opis metody przedstawionej na Listingu nr. 3 (s. 16)**

- Kod rozpoczyna od przetworzenia daty zawartej w obiekcie `lineData`. Metoda `getDate()` zwraca datę w formacie `dd.mm.yyyy HH:MM`. Następnie, za pomocą `stringstream` i funkcji `getline()`, data jest rozdzielana na poszczególne składniki: dzień, miesiąc i rok, które są następnie zapisane w `dateParts`. (Linia 2-8)
- Na podstawie rozdzielonej daty, wyciągane są poszczególne elementy: rok, miesiąc, dzień, godzina i minuta. Następnie, godzina i minuta są używane do obliczenia kwartału, przy czym kwartały są ustalane na podstawie czasu (np. pierwszy kwartał to godziny 00:00–00:59). (Linia 10-15)
- Na podstawie wyodrębnionych danych (rok, miesiąc, dzień, godzina, minuta, kwartał), metoda wstawia nowy obiekt `lineData` do odpowiedniego węzła w drzewie. Węzeł drzewa jest hierarchicznie zorganizowany, zaczynając od roku, przez miesiąc, dzień, kwartał, a na końcu dodawana jest sama data (`lineData`) do listy `data` w odpowiednim kwartale. (Linia 17-24)

**4.5. Działanie programu**

Program posiada interaktywne menu w którym użytkownik może wybrać jedną z opcji do wykonania. Menu zostało przedstawione na rysunku nr. 4.1 (s. 17)



```
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ6\projekt\build\ucrt64\
MENU:

1. Załaduj dane z pliku
2. Wyświetl strukturę drzewa
3. Pobierz dane w określonym przedziale czasowym
4. Oblicz sumy w określonym przedziale czasowym
5. Oblicz średnią wartość w określonym przedziale czasowym
6. Porównaj dane między dwoma zakresami czasowymi
7. Wyszukaj dane w określonym przedziale czasowym z tolerancją
8. Zapisz dane do pliku binarnego
9. Wczytaj dane z pliku binarnego
10. Wyjdź

Wybierz działanie: █
```

Rys. 4.1. Menu programu

Teraz za pomocą 1 opcji użytkownik może wczytać do programu dane z pliku .csv, które można później poddać analizie, zostało to pokazane na rysunku nr. 4.2 (s. 18).

```
Wybierz działanie: 1

Dane zostały załadowane pomyślnie.
Załadowano 5953 linii
Znaleziono 7 niepoprawnych linii
Sprawdź pliki log i log_error, aby uzyskać więcej informacji

MENU:

1. Załaduj dane z pliku
2. Wyświetl strukturę drzewa
3. Pobierz dane w określonym przedziale czasowym
4. Oblicz sumy w określonym przedziale czasowym
5. Oblicz średnią wartość w określonym przedziale czasowym
6. Porównaj dane między dwoma zakresami czasowymi
7. Wyszukaj dane w określonym przedziale czasowym z tolerancją
8. Zapisz dane do pliku binarnego
9. Wczytaj dane z pliku binarnego
10. Wyjdź

Wybierz działanie: █
```

Rys. 4.2. Załadowanie pliku .csv


Jak widzimy na rysunku nr. 4.2 (s. 18) z pliku .csv zostało załadowane poprawnie 5953 linii, oraz znaleziono 7 niepoprawnych linii, aby uzyskać informacje na temat tego błędu, użytkownik może zobaczyć pliki z logami, które program generuje. Teraz możemy przejść do testu np. 3 opcji, wybieramy ją i wpisujemy ramy czasowe tak jak na rysunku nr. 4.3 (s. 18)

```
Wybierz działanie: 3

Podaj datę początkową (dd.mm.yyyy hh:mm): 01.10.2020 0:00
Podaj datę końcową (dd.mm.yyyy hh:mm): 01.10.2020 3:00
Dane pomiędzy 01.10.2020 0:00 a 01.10.2020 3:00:
01.10.2020 0:00 0 0 406.832 406.832 0
01.10.2020 0:15 0 0 403.566 403.566 0
01.10.2020 0:30 0 0 336.733 336.733 0
01.10.2020 0:45 0 0 306.529 306.529 0
01.10.2020 1:00 0 0 1397.27 1397.27 0
01.10.2020 1:15 0 0 267.803 267.803 0
01.10.2020 1:30 0 0 283.664 283.664 0
01.10.2020 1:45 0 0 391.34 391.34 0
01.10.2020 2:00 0 0 408.269 408.269 0
```

Rys. 4.3. Testowanie opcji

Jak widzimy na rysunku nr. 4.3 (s. 18), dane zostały prawidłowo wypisane z pliku .csv zgodnie z wybranym przedziałem czasowym. Teraz możemy sobie zapisać nasze dane do pliku binarnego, tak jak zostało to pokazane na rysunku nr. 4.4 (s. 19).

```
5. Oblicz średnią wartość w określonym przedziale czas  
6. Porównaj dane między dwoma zakresami czasowymi  
7. Wyszukaj dane w określonym przedziale czasowym z to  
8. Zapisz dane do pliku binarnego  
9. Wczytaj dane z pliku binarnego  
10. Wyjdź  
  
Wybierz działanie: 8  
  
Dane zostały pomyślnie zapisane.   
  
MENU:  
  
1. Załaduj dane z pliku  
2. Wyświetl strukturę drzewa  
3. Pobierz dane w określonym przedziale czasowym
```

Rys. 4.4. Zapis do pliku binarnego

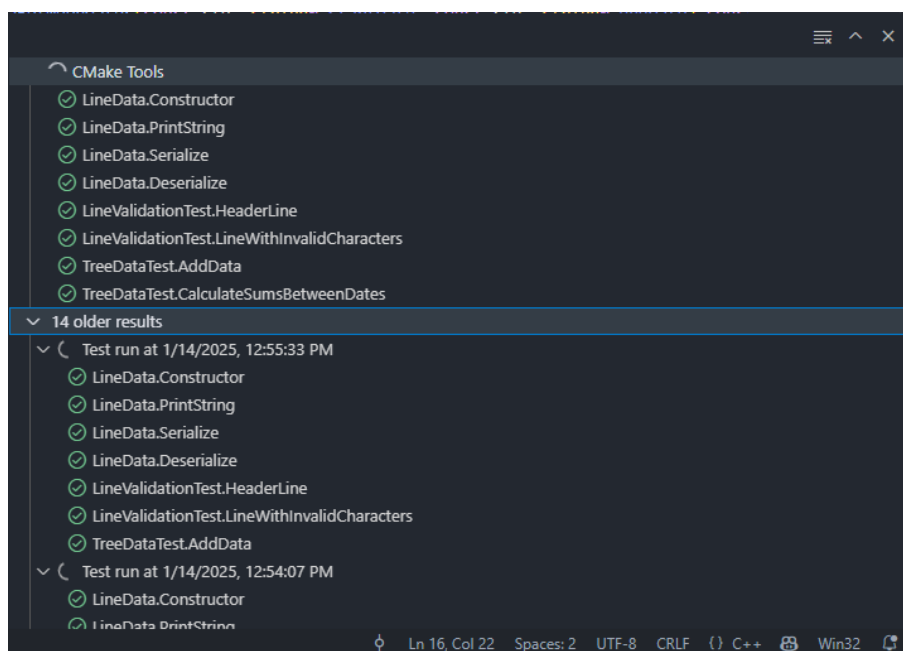
Możemy przetestować jeszcze 5 opcję która oblicza średnie wartości w określonym przedziale czasowym, działanie zostało przedstawione na rysunku nr. 4.5 (s. 19).

```
9. Wczytaj dane z pliku binarnego  
10. Wyjdź  
  
Wybierz działanie: 5  
  
Podaj datę początkową (dd.mm.yyyy hh:mm): 1.10.2020 0:00  
Podaj datę końcową (dd.mm.yyyy hh:mm): 1.10.2020 9:45  
Średnie wartości pomiędzy 1.10.2020 0:00 a 1.10.2020 9:45:  
Autokonsumpcja: 3.60185  
Eksport: 0  
Import: 566.815  
Pobór: 570.417  
Produkcja: 3.60185  
  
MENU:  
  
1. Załaduj dane z pliku  
2. Wyświetl strukturę drzewa
```

Rys. 4.5. Testowanie opcji programu

## 4.6. Testy programu

W projekcie został przeprowadzony test za pomocą frameworka Google Test, który umożliwił dokładną weryfikację poprawności działania kluczowych modułów aplikacji. Na rysunku nr. 4.6 (s. 20) został przeprowadzony zrzut z wykonanych testów.



Rys. 4.6. Testy kodu za pomocą Google Test

Google Test został wybrany, ponieważ jest to narzędzie wydajne, elastyczne i dostosowane do wymagań projektu. Jego zastosowanie umożliwiło szybkie i dokładne przeprowadzenie testów jednostkowych, zwiększając niezawodność aplikacji. W projekcie do testowania można było jeszcze użyć takiego narzędzie jak Catch2 który jest bardzo łatwy w użyciu oraz ma intuicyjny system asercji i czytelne raport o wynikach testów, lecz jest mniej zaawansowany niż Google Test w przypadku bardziej złożonych scenariuszy, takich jak testy parametrów czy współpraca z dużymi projektami.

## 4.7. Pomoc AI w projekcie

W projekcie został wykorzystany GitHub Copilot, który pomógł przy pisaniu różnych fragmentów kodu. AI szczególnie był pomocny przy:

- **Generowanie szkieletu kodu** - Copilot był szczególnie pomocny w tworzeniu szkieletu kodu. Automatycznie proponował struktury funkcji, deklaracje klas oraz szablony kodu bazujące na nazwach funkcji i zmiennych. To znacznie przyspieszyło proces tworzenia początkowych wersji modułów, takich jak obsługa menu czy operacje na plikach.
- **Operacje na strukturach danych** - Podczas implementacji funkcji dla struktury drzewa (`treeData.hpp`), Copilot dobrze sugerował metody wyszukiwania, filtrowania i iteracji po węzłach. Przyspieszyło to implementację takich funkcji, jak obliczanie sum czy wyszukiwanie danych w przedziale czasowym.
- **Obsługa plików** - Przy implementacji funkcji wczytywania i zapisywania danych (CSV, binarne), Copilot pomógł w generowaniu kodu do operacji na plikach, takich jak otwieranie plików, odczytywanie danych w pętlach oraz zapisywanie ich w odpowiednich formatach. Umożliwiło to skrócenie czasu pracy nad tymi funkcjami.

Copilot nie był pomocny w kilku kwestiach:

- **Specyficzne wymagania projektu** - Copilot często sugerował ogólne rozwiązania, które nie zawsze pasowały do wymagań projektu. Na przykład przy pracy z niestandardową strukturą drzewa (`TreeData`) lub specyficznym formatem danych w plikach CSV konieczne było ręczne dopracowywanie kodu. Podpowiedzi wymagały dostosowania do rzeczywistych potrzeb projektu.
- **Logika biznesowa i analizy danych** - Chociaż Copilot był pomocny w pisaniu szkieletu funkcji, nie zawsze trafnie rozumiał zaawansowane operacje, takie jak obliczanie średnich, porównywanie danych między zakresami czasowymi czy implementacja tolerancji w wyszukiwaniu. Te fragmenty wymagały manualnego dostosowania.

## 5. Wnioski

Realizacja projektu pozwoliła na stworzenie aplikacji umożliwiającej efektywne przetwarzanie i analizowanie danych energetycznych zapisanych w formacie CSV. Aplikacja spełnia założenia projektowe, zapewniając użytkownikom narzędzia do organizowania danych w hierarchiczną strukturę drzewa oraz przeprowadzania różnorodnych analiz. Podczas pracy nad projektem zidentyfikowano kluczowe wyzwania i rozwiązania, które miały istotny wpływ na jakość i funkcjonalność aplikacji.

1. **Efektywność algorytmu i struktura danych** - Kluczowym elementem projektu było stworzenie wydajnego algorytmu organizowania danych w strukturę drzewa, co pozwala na szybki dostęp do informacji w różnych przedziałach czasowych. Zastosowanie struktury drzewa okazało się optymalnym rozwiązaniem, umożliwiającym łatwe zarządzanie danymi oraz przeprowadzanie operacji analitycznych. Dzięki temu użytkownicy mogą szybko uzyskać dostęp do wymaganych informacji oraz wykonywać obliczenia na danych w sposób efektywny.
2. **Zarządzanie błędnymi danymi** - Projekt wymagał szczególnej uwagi w zakresie obsługi błędnych danych, takich jak puste linie, niekompletne rekordy czy powtarzające się wiersze. Zostały zaimplementowane mechanizmy walidacji, które umożliwiły odrzucanie błędnych danych i generowanie szczegółowych logów. Dzięki temu program jest odporny na błędy w danych wejściowych, co zapewnia jego niezawodność i stabilność w trakcie użytkowania.
3. **Optymalizacja wydajności** - Przetwarzanie dużych zbiorów danych energetycznych wiąże się z koniecznością zoptymalizowania pamięci oraz algorytmów. W trakcie realizacji projektu zastosowano odpowiednie techniki optymalizacji, które pozwalają na efektywne przetwarzanie danych bez nadmiernego obciążania systemu. Zastosowanie języka C++ oraz narzędzi takich jak GitHub Copilot, pozwoliło na szybsze generowanie kodu oraz implementację algorytmów, co wpłynęło na skrócenie czasu rozwoju projektu.

Projekt został zrealizowany zgodnie z założeniami i spełnia wymagania dotyczące efektywnego przetwarzania danych oraz analizy energii. Aplikacja zapewnia nie tylko odpowiednią funkcjonalność, ale również odporność na błędne dane wejściowe, wysoką wydajność oraz elastyczność w zarządzaniu danymi. Zastosowanie nowoczesnych narzędzi i technologii przyczyniło się do przyspieszenia procesu programowania oraz poprawy jakości aplikacji, co stanowi solidną podstawę do rozwoju projektu w przyszłości.

## Bibliografia

- [1] *Github Copilot*. URL: <https://github.com/features/copilot> (term. wiz. 12.01.2025).
- [2] *pbiecek gitbooks*. URL: [https://pbiecek.gitbooks.io/przewodnik/content/Programowanie/jak\\_wczytac\\_tabele\\_danych\\_z\\_pliku\\_csv\\_lub\\_txt.html](https://pbiecek.gitbooks.io/przewodnik/content/Programowanie/jak_wczytac_tabele_danych_z_pliku_csv_lub_txt.html) (term. wiz. 12.01.2025).
- [3] *Algorytm edu*. URL: <https://www.algorytm.edu.pl/pliki-tekstowe/czytanie-pliku.html> (term. wiz. 12.01.2025).
- [4] *Binarnie c++*. URL: <https://binarnie.pl/odczyt-zapis-danych-pliku-c/> (term. wiz. 12.01.2025).

## Spis rysunków

4.1. Menu programu . . . . .	17
4.2. Załadowanie pliku .csv . . . . .	18
4.3. Testowanie opcji . . . . .	18
4.4. Zapis do pliku binarnego . . . . .	19
4.5. Testowanie opcji programu . . . . .	19
4.6. Testy kodu za pomocą Google Test . . . . .	20



## Spis listingów

1.	LineData::serialize(ofstream& out) const . . . . .	14
2.	Logger::Logger(const std::string& filename) . . . . .	15
3.	Metoda double oblicz_pi . . . . .	16