AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

EnergyAnalysis

Programowanie urządzeń mobilnych

Autor:
Blavitskyi Mykola
Trudov Mykhailo
Stasiuk Oleh

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

Spis treści

1. Ogólne określenie wymagań

Projekt ma na celu stworzenie systemu do analizy i przechowywania danych energetycznych. System będzie składał się z kilku modułów, które umożliwią wczytywanie danych z plików CSV, przechowywanie ich w strukturach danych, oraz wykonywanie operacji analitycznych na tych danych.

Główne wymagania systemu obejmują:

- Wczytywanie danych z plików CSV i przechowywanie ich w wektorze obiektów klasy `Punkt`.
- Przechowywanie danych w strukturze drzewiastej, umożliwiającej szybki dostęp do danych na podstawie dat.
- Implementacja funkcji analitycznych, takich jak obliczanie sumy autokonsumpcji i eksportu energii dla określonych przedziałów czasowych.
- Logowanie błędów podczas wczytywania danych z plików CSV.
- Testowanie jednostkowe za pomocą biblioteki Google Test.

1.1. Przykład

Poniżej znajduje się przykładowy kod programu, który wczytuje dane z pliku CSV, przechowuje je w strukturze drzewiastej, oraz oblicza sumę autokonsumpcji dla określonego dnia i ćwiartki[1]:

```
#include <iostream>
#include <vector>
#include "Punkt.h"
#include "Drzewo.h"
#include "CSVReader.h"

int main() {
    std::vector<Punkt> dane = wczytajDane("dane.csv");

    Drzewo drzewo;
    for (const auto& punkt : dane) {
        drzewo.dodajPunkt(punkt);
    }

```

1

```
15      std::cout << "Suma autokonsumpcji: " << drzewo.
        sumaAutokonsumpcji(2021, 10, 3, 2) << std::endl;
16
17      return 0;
18  }
```

Listing 1. ObliczanieAutokonsumpcji

Powyższy kod wczytuje dane z pliku `dane.csv` za pomocą funkcji `wczytajDane`. Następnie dodaje te dane do struktury drzewa za pomocą metody `dodajPunkt`. Na końcu oblicza sumę autokonsumpcji dla określonej daty i ćwiartki, a wynik wyświetla na standardowym wyjściu.

2. Analiza problemu

2.1. Wczytywanie danych z pliku CSV

Jednym z głównych wyzwań projektu jest poprawne wczytywanie danych z pliku CSV. Plik CSV może zawierać błędy, takie jak niepoprawne formatowanie danych. W projekcie zastosowano mechanizm logowania błędów do pliku `log_error.txt`, który rejestruje wszelkie problemy napotkane podczas wczytywania danych[2].

```
std::vector<Punkt> wczytajDane(const std::string& nazwaPliku) {
    std::ifstream plik(nazwaPliku);
    std::string linia;
    std::vector<Punkt> dane;

    std::getline(plik, linia); // Pomijamy nag Ć wek

    while (std::getline(plik, linia)) {
        std::stringstream ss(linia);
        std::string data;
        double autokonsumpcja, eksport, import, pobor, produkcja;

        if (ss >> data >> autokonsumpcja >> eksport >> import >>
   pobor >> produkcja) {
            dane.push_back(Punkt(data, autokonsumpcja, eksport,
   import, pobor, produkcja));
        } else {
            std::ofstream logError("log_error.txt", std::ios::app);
            logError << "B Ć d wczytywania linii: " << linia <<
   std::endl;
        }
    }
    return dane;
}
```

Listing 2. wczytajPunkty

2.2. Przechowywanie danych w strukturze drzewiastej

Kolejnym wyzwaniem jest przechowywanie danych w sposób umożliwiający szybki dostęp do nich na podstawie daty. W projekcie wykorzystano strukturę drzewiastą, która przechowuje dane w zagnieżdżonych mapach[3].

[2] wczytajPunkty
[3]

```
void Drzewo::dodajPunkt(const Punkt& punkt) {
    int rok, miesiac, dzien, cwartka;
    rozbijDate(punkt.getData(), rok, miesiac, dzien, cwartka);

    // Doda  dane do struktury drzewa
    dane[rok][miesiac][dzien][cwartka].push_back(punkt);
}
```

Listing 3. dodajPunktDoDrzewa

2.3. Operacje analityczne

Aby umożliwić analizę danych, zaimplementowano funkcje obliczające sumy autokonsumpcji oraz eksportu energii dla określonych przedziałów czasowych[4].

```
double Drzewo::sumaAutokonsumpcji(int rok, int miesiac, int dzien,
    int cwartka) {
    double suma = 0.0;

    // Sprawdzenie dost Źpno Żci danych
    if (dane.find(rok) != dane.end() &&
        dane[rok].find(miesiac) != dane[rok].end() &&
        dane[rok][miesiac].find(dzien) != dane[rok][miesiac].end()
    &&
        dane[rok][miesiac][dzien].find(cwartka) != dane[rok][
    miesiac][dzien].end()) {
        for (const auto& punkt : dane[rok][miesiac][dzien][cwartka
    ]) {
            suma += punkt.getAutokonsumpcja();
        }
    }

    return suma;
}
```

Listing 4. obliczSumeAutokonsumpcji

2.4. Testowanie jednostkowe

Aby zapewnić poprawność działania programu, zastosowano testy jednostkowe za pomocą biblioteki Google Test[5].

4

5

```
#include <gtest/gtest.h>
#include "Drzewo.h"
#include "Punkt.h"

TEST(DrzewoTest, TestSumaAutokonsumpcji) {
    Drzewo drzewo;
    Punkt punkt("2021-10-03 10:15", 150.0, 100.0, 50.0, 80.0,
   200.0);
    drzewo.dodajPunkt(punkt);
    ASSERT_EQ(drzewo.sumaAutokonsumpcji(2021, 10, 3, 2), 150.0);
}
```

Listing 5. testObliczSumeAutokonsumpcji

3. Projektowanie

3.1. Narzędzia

W projekcie będziemy korzystać z następujących narzędzi:

- Język programowania: C++
- Kompilator: g++
- System kontroli wersji: Git
- Biblioteki dodatkowe: Google Test (do testowania jednostkowego)

3.2. Ustawienia kompilatora

Aby skompilować projekt, używamy kompilatora g++ z następującymi ustawieniami:

- `-std=c++11` - Ustawienie standardu C++
- `-Wall` - Włączanie wszystkich ostrzeżeń
- `-lgtest` - Linkowanie z biblioteką Google Test
- `-lgtest_main` - Linkowanie z główną biblioteką Google Test
- `-pthread` - Używanie wątków POSIX

Przykładowe polecenie do kompilacji projektu:

```
g++ -std=c++11 -Wall -o program Program.cpp CSVReader.cpp Drzewo.cpp Punkt.cpp -l
```

3.3. Używanie narzędzi

3.3.1. Git

System kontroli wersji Git będzie używany do zarządzania kodem źródłowym projektu. Podstawowe komendy Git:

- `git clone <URL_repozytorium>` - Sklonowanie repozytorium
- `git add <plik>` - Dodanie pliku do śledzenia
- `git commit -m «wiadomość>"` - Zatwierdzenie zmian z wiadomością

- `git push` - Wysłanie zmian do zdalnego repozytorium
- `git pull` - Pobranie zmian ze zdalnego repozytorium

3.3.2. Google Test

Google Test jest biblioteką do testowania jednostkowego. Testy są definiowane w plikach testowych, a następnie kompilowane i uruchamiane razem z projektem. Przykładowy test jednostkowy[6]:

```
#include <gtest/gtest.h>
#include "Drzewo.h"
#include "Punkt.h"

TEST(DrzewoTest, TestSumaAutokonsumpcji) {
    Drzewo drzewo;
    Punkt punkt("2021-10-03 10:15", 150.0, 100.0, 50.0, 80.0,
   200.0);
    drzewo.dodajPunkt(punkt);
    ASSERT_EQ(drzewo.sumaAutokonsumpcji(2021, 10, 3, 2), 150.0);
}
```

Listing 6. testSumaAutokonsumpcji

3.3.3. Opis narzędzia: Google Test

Google Test to framework do testowania jednostkowego w C++. Umożliwia definiowanie testów, które sprawdzają poprawność działania kodu. Poniżej znajduje się przykładowy scenariusz użycia Google Test:

- Najpierw instalujemy Google Test za pomocą menedżera pakietów:

```
sudo apt-get install libgtest-dev
sudo apt-get install cmake
cd /usr/src/gtest
sudo cmake .
sudo make
sudo cp *.a /usr/lib
```

- Następnie tworzymy plik testowy, w którym definiujemy testy, np. `test_Drzewo.cpp`.

6

- Kompilujemy projekt razem z testami:

```
g++ -std=c++11 -Wall -o test_program test_Drzewo.cpp Drzewo.cpp Punkt.cp
```

- Uruchamiamy skompilowany program testowy, aby wykonać testy:

```
./test_program
```

4. Implementacja

4.1. Struktura projektu

Projekt składa się z następujących plików:

- `CSVReader.cpp` - Implementacja funkcji do wczytywania danych z plików CSV.
- `CSVReader.h` - Nagłówek dla pliku `CSVReader.cpp`.
- `Drzewo.cpp` - Implementacja klasy `Drzewo`, która przechowuje i analizuje dane.
- `Drzewo.h` - Nagłówek dla pliku `Drzewo.cpp`.
- `Punkt.cpp` - Implementacja klasy `Punkt`, która przechowuje pojedynczy rekord danych.
- `Punkt.h` - Nagłówek dla pliku `Punkt.cpp`.
- `Program.cpp` - Główny plik programu, który wczytuje dane, przechowuje je w strukturze drzewiastej i wykonuje operacje analityczne.
- `test_Drzewo.cpp` - Testy jednostkowe dla klasy `Drzewo`.
- `test_Punkt.cpp` - Testy jednostkowe dla klasy `Punkt`.

4.2. Funkcja wczytywania danych z pliku CSV

Implementacja funkcji `wczytajDane` w pliku `CSVReader.cpp`[7]:

```
#include "CSVReader.h"
#include "Punkt.h"
#include <fstream>
#include <sstream>

std::vector<Punkt> wczytajDane(const std::string& nazwaPliku) {
    std::ifstream plik(nazwaPliku);
    std::string linia;
    std::vector<Punkt> dane;

    std::getline(plik, linia); // Pomijamy nag Ć wek

    while (std::getline(plik, linia)) {
        std::stringstream ss(linia);
```

7

```
15          std::string data;
16          double autokonsumpcja, eksport, import, pobor, produkcja;
17
18          if (ss >> data >> autokonsumpcja >> eksport >> import >>
    pobor >> produkcja) {
19              dane.push_back(Punkt(data, autokonsumpcja, eksport,
    import, pobor, produkcja));
20          } else {
21              std::ofstream logError("log_error.txt", std::ios::app);
22              logError << "B Ć d wczytywania linii: " << linia <<
    std::endl;
23          }
24      }
25      return dane;
26  }
```

Listing 7. wczytajDaneZPliku

4.3. Klasa `Drzewo`

Implementacja klasy `Drzewo` w pliku `Drzewo.cpp`, która przechowuje dane w strukturze drzewiastej i umożliwia wykonywanie operacji analitycznych[8]:

```
1  #include "Drzewo.h"
2  #include "Punkt.h"
3  #include <sstream>
4  #include <iostream>
5
6  // Funkcja rozbijaj ca dat Ź na rok, miesi c, dzie  i  wiartkę
7  void Drzewo::rozbijDate(const std::string& data, int& rok, int&
    miesiac, int& dzien, int& cwartka) {
8      std::stringstream ss(data);
9      std::string rok_str, miesiac_str, dzien_str, godzina_str,
    minuta_str;
10     char separator;
11
12     ss >> rok_str >> separator >> miesiac_str >> separator >>
    dzien_str >> godzina_str >> separator >> minuta_str;
13
14     rok = std::stoi(rok_str);
15     miesiac = std::stoi(miesiac_str);
16     dzien = std::stoi(dzien_str);
17
```

[8]

```
18      int godzina = std::stoi(godzina_str);
19      int minuta = std::stoi(minuta_str);
20
21      if (godzina < 6) {
22          cwartka = 0;
23      } else if (godzina < 12) {
24          cwartka = 1;
25      } else if (godzina < 18) {
26          cwartka = 2;
27      } else {
28          cwartka = 3;
29      }
30  }
31
32  // Metoda dodaj ca nowy punkt do drzewa
33  void Drzewo::dodajPunkt(const Punkt& punkt) {
34      int rok, miesiac, dzien, cwartka;
35      rozbijDate(punkt.getData(), rok, miesiac, dzien, cwartka);
36
37      dane[rok][miesiac][dzien][cwartka].push_back(punkt);
38  }
39
40  // Funkcja obliczaj ca sum Ź autokonsumpcji
41  double Drzewo::sumaAutokonsumpcji(int rok, int miesiac, int dzien,
        int cwartka) {
42      double suma = 0.0;
43
44      if (dane.find(rok) != dane.end() &&
45          dane[rok].find(miesiac) != dane[rok].end() &&
46          dane[rok][miesiac].find(dzien) != dane[rok][miesiac].end()
        &&
47          dane[rok][miesiac][dzien].find(cwartka) != dane[rok][
        miesiac][dzien].end()) {
48          for (const auto& punkt : dane[rok][miesiac][dzien][cwartka
        ]) {
49              suma += punkt.getAutokonsumpcja();
50          }
51      }
52
53      return suma;
54  }
```

Listing 8. rozbicieDaty

4.4. Testowanie jednostkowe

Testy jednostkowe dla klasy `Drzewo` w pliku `test_Drzewo.cpp`[9]:

```
#include <gtest/gtest.h>
#include "Drzewo.h"
#include "Punkt.h"

TEST(DrzewoTest, TestSumaAutokonsumpcji) {
    Drzewo drzewo;
    Punkt punkt("2021-10-03 10:15", 150.0, 100.0, 50.0, 80.0,
    200.0);
    drzewo.dodajPunkt(punkt);
    ASSERT_EQ(drzewo.sumaAutokonsumpcji(2021, 10, 3, 2), 150.0);
}
```

Listing 9. testObliczSumeAutokonsumpcji

9

5. Wnioski

Projekt ten umożliwił stworzenie systemu do analizy i przechowywania danych energetycznych, spełniającego określone wymagania. Poniżej znajdują się główne wnioski z realizacji projektu:

5.1. Wnioski dotyczące wczytywania danych

Funkcja `wczytajDane` skutecznie przetwarza dane z plików CSV i przechowuje je w strukturze wektora obiektów klasy `Punkt`. Mechanizm logowania błędów okazał się przydatny w identyfikacji niepoprawnych linii w plikach CSV, co pozwala na ich późniejszą korektę.

5.2. Wnioski dotyczące przechowywania danych

Struktura drzewiasta zastosowana w klasie `Drzewo` pozwala na efektywne przechowywanie i szybki dostęp do danych na podstawie daty. Zastosowanie zagnieżdżonych map umożliwia łatwe grupowanie danych według roku, miesiąca, dnia i ćwiartki.

5.3. Wnioski dotyczące analizy danych

Implementacja funkcji analitycznych, takich jak `sumaAutokonsumpcji` i `sumaEksportu`, umożliwia szybkie i dokładne obliczenia na dużych zbiorach danych. Struktura drzewiasta pozwala na szybkie wyszukiwanie i agregowanie danych, co jest kluczowe dla wydajności systemu.

5.4. Wnioski dotyczące testowania jednostkowego

Zastosowanie biblioteki Google Test do testowania jednostkowego pozwoliło na wczesne wykrywanie błędów i zapewnienie wysokiej jakości kodu. Testy jednostkowe okazały się nieocenione w procesie weryfikacji poprawności działania funkcji i metod klas.

5.5. Porównanie wersji sztucznej inteligencji

Podczas pracy nad projektem korzystano zarówno z ChatGPT, jak i GitHub Copilot. Obie technologie znacząco wspierają proces tworzenia oprogramowania, jednak różnią się swoimi możliwościami i zastosowaniem:

- **ChatGPT:** Jest bardziej elastyczny i wszechstronny w generowaniu odpowiedzi oraz wyjaśnień. Świetnie sprawdza się w zadaniach wymagających złożonych analiz, rozwiązywania problemów i generowania tekstów technicznych.
- **GitHub Copilot:** Jest bardziej zintegrowany z procesem programowania. Automatycznie generuje fragmenty kodu na podstawie kontekstu i znacząco przyspiesza pisanie kodu, zwłaszcza w przypadku powtarzalnych zadań.

Zastosowanie obu narzędzi w projekcie pozwoliło na osiągnięcie wysokiej efektywności, łącząc ich zalety – szybkość generowania kodu z pomocą Copilota oraz dogłębną analizę i dostosowanie rozwiązań dzięki ChatGPT.

5.6. Bibliografia

Podczas realizacji projektu korzystano z następujących źródeł informacji:

- ChatGPT
- GitHub Copilot
- Wikipedia

5.7. Podsumowanie

Projekt zakończył się sukcesem, spełniając wszystkie założone cele i wymagania. System jest w stanie skutecznie wczytywać, przechowywać i analizować dane energetyczne, zapewniając jednocześnie wysoką wydajność i dokładność. W przyszłości można rozważyć rozszerzenie funkcjonalności systemu o dodatkowe analizy i raportowanie danych.

Spis rysunków

Spis tabel

Spis listingów