Projekt 6

6

Wygenerowano za pomocą Doxygen 1.12.0

1 Indeks klas 1

1.1 Lista klas . . . 1

2 Indeks plików 3

2.1 Lista plikÃ³w . . . 3

3 Dokumentacja klas 5

3.1 Dokumentacja struktury TreeData::DayNode . . . 5

3.1.1 Opis szczegÃ³Åowy . . . 5

3.1.2 Dokumentacja atrybutÃ³w skÅadowych . . . 5

3.1.2.1 day . . . 5

3.1.2.2 quarters . . . 6

3.2 Dokumentacja klasy LogManager . . . 6

3.2.1 Opis szczegÃ³Åowy . . . 6

3.2.2 Dokumentacja konstruktora i destruktora . . . 6

3.2.2.1 LogManager() . . . 6

3.2.2.2 ~LogManager() . . . 7

3.2.3 Dokumentacja funkcji skÅadowych . . . 7

3.2.3.1 log() . . . 7

3.2.4 Dokumentacja atrybutÃ³w skÅadowych . . . 8

3.2.4.1 logFile . . . 8

3.3 Dokumentacja struktury TreeData::MonthNode . . . 8

3.3.1 Opis szczegÃ³Åowy . . . 9

3.3.2 Dokumentacja atrybutÃ³w skÅadowych . . . 9

3.3.2.1 days . . . 9

3.3.2.2 month . . . 9

3.4 Dokumentacja struktury TreeData::QuarterNode . . . 9

3.4.1 Opis szczegÃ³Åowy . . . 10

3.4.2 Dokumentacja atrybutÃ³w skÅadowych . . . 10

3.4.2.1 data . . . 10

3.4.2.2 hour . . . 10

3.4.2.3 minute . . . 10

3.4.2.4 quarter . . . 10

3.5 Dokumentacja klasy RowData . . . 11

3.5.1 Opis szczegÃ³Åowy . . . 12

3.5.2 Dokumentacja konstruktora i destruktora . . . 12

3.5.2.1 RowData() [1/2] . . . 12

3.5.2.2 RowData() [2/2] . . . 12

3.5.3 Dokumentacja funkcji skÅadowych . . . 13

3.5.3.1 display() . . . 13

3.5.3.2 displayData() . . . 13

3.5.3.3 getConsumption() . . . 13

3.5.3.4 getDate() . . . 14

3.5.3.5 getExport() . . . 14
3.5.3.6 getImport() . . . 14
3.5.3.7 getProduction() . . . 14
3.5.3.8 getSelfConsumption() . . . 15
3.5.3.9 loadFromBinary() . . . 15
3.5.3.10 saveToBinary() . . . 15
3.5.3.11 toString() . . . 16
3.5.4 Dokumentacja atrybutÃ³w skÅadowych . . . 16
3.5.4.1 consumption . . . 16
3.5.4.2 date . . . 17
3.5.4.3 exportValue . . . 17
3.5.4.4 importValue . . . 17
3.5.4.5 production . . . 17
3.5.4.6 selfConsumption . . . 17
3.6 Dokumentacja klasy TreeData . . . 17
3.6.1 Opis szczegÃ³Åowy . . . 18
3.6.2 Dokumentacja funkcji skÅadowych . . . 19
3.6.2.1 addData() . . . 19
3.6.2.2 calculateAveragesBetweenDates() . . . 19
3.6.2.3 calculateSumsBetweenDates() . . . 20
3.6.2.4 compareDataBetweenDates() . . . 21
3.6.2.5 getDataBetweenDates() . . . 22
3.6.2.6 print() . . . 23
3.6.2.7 searchRecordsWithTolerance() . . . 24
3.6.3 Dokumentacja atrybutÃ³w skÅadowych . . . 24
3.6.3.1 years . . . 24
3.7 Dokumentacja struktury TreeData::YearNode . . . 24
3.7.1 Opis szczegÃ³Åowy . . . 25
3.7.2 Dokumentacja atrybutÃ³w skÅadowych . . . 25
3.7.2.1 months . . . 25
3.7.2.2 year . . . 25

4 Dokumentacja plików **27**
4.1 Dokumentacja pliku GoogleTest/test.cpp . . . 27
4.1.1 Opis szczegÃ³Åowy . . . 27
4.1.2 Dokumentacja funkcji . . . 28
4.1.2.1 TEST() [1/7] . . . 28
4.1.2.2 TEST() [2/7] . . . 28
4.1.2.3 TEST() [3/7] . . . 28
4.1.2.4 TEST() [4/7] . . . 28
4.1.2.5 TEST() [5/7] . . . 29
4.1.2.6 TEST() [6/7] . . . 29

4.1.2.7 TEST() [7/7] . . . 29
4.2 test.cpp . . . 30
4.3 Dokumentacja pliku P6/LineValidation.h . . . 31
4.3.1 Opis szczegÃ³Åowy . . . 31
4.3.2 Dokumentacja funkcji . . . 31
4.3.2.1 lineValidation() . . . 31
4.4 LineValidation.h . . . 32
4.5 Dokumentacja pliku P6/LogManager.cpp . . . 32
4.5.1 Opis szczegÃ³Åowy . . . 33
4.5.2 Dokumentacja zmiennych . . . 33
4.5.2.1 errorLogCount . . . 33
4.5.2.2 errorLogger . . . 33
4.5.2.3 globalLogger . . . 33
4.6 LogManager.cpp . . . 34
4.7 Dokumentacja pliku P6/LogManager.h . . . 34
4.7.1 Opis szczegÃ³Åowy . . . 35
4.7.2 Dokumentacja zmiennych . . . 35
4.7.2.1 errorLogCount . . . 35
4.7.2.2 errorLogger . . . 35
4.7.2.3 globalLogger . . . 35
4.8 LogManager.h . . . 36
4.9 Dokumentacja pliku P6/main.cpp . . . 36
4.9.1 Opis szczegÃ³Åowy . . . 36
4.9.2 Dokumentacja funkcji . . . 37
4.9.2.1 displayMenu() . . . 37
4.9.2.2 main() . . . 37
4.10 main.cpp . . . 39
4.11 Dokumentacja pliku P6/RowData.cpp . . . 41
4.11.1 Opis szczegÃ³Åowy . . . 41
4.12 RowData.cpp . . . 41
4.13 Dokumentacja pliku P6/RowData.h . . . 42
4.13.1 Opis szczegÃ³Åowy . . . 43
4.14 RowData.h . . . 43
4.15 Dokumentacja pliku P6/TreeData.cpp . . . 43
4.15.1 Opis szczegółowy . . . 44
4.16 TreeData.cpp . . . 44
4.17 Dokumentacja pliku P6/TreeData.h . . . 46
4.17.1 Opis szczegÃ³Åowy . . . 46
4.18 TreeData.h . . . 47

Skorowidz **49**

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdujÄ siÄ klasy, struktury, unie i interfejsy wraz z ich krÃ^{3}tkimi opisami:

TreeData::DayNode
: Reprezentuje dane dzienne. Struktura ta zawiera informacje o danym dniu oraz mapê kwartalnych danych w tym dniu 5

LogManager
: Klasa obs^{3}uguj¹ca logowanie komunikatów do plików tekstowych 6

TreeData::MonthNode
: Reprezentuje dane miesiêczne. Struktura ta zawiera informacje o danym miesi¹cu oraz mapê dziennych danych w tym miesi¹cu 8

TreeData::QuarterNode
: Reprezentuje dane z podzia^{3}em na kwarta^{3}y dnia. Struktura ta zawiera informacje o godzinie, minucie oraz dane przypisane do danego kwarta^{3}u 9

RowData
: Klasa reprezentuj¹ca dane jednego wiersza z pliku CSV, zawieraj¹ca ró¿ne parametry energetyczne 11

TreeData
: $<$ Za^{3}¹czenie pliku nag^{3}ówkowego zawieraj¹cego klasê RowData 17

TreeData::YearNode
: Reprezentuje dane roczne. Struktura ta zawiera informacje o roku oraz mapê miesiêcznych danych w tym roku 24

Rozdział 2

Indeks plików

2.1 Lista plikÃ³w

Tutaj znajduje siÄ lista wszystkich plikÃ³w wraz z ich krÃ³tkimi opisami:

GoogleTest/test.cpp
: Zawiera zestaw testów jednostkowych dla modu³ów RowData, TreeData, Logger i LineValidation 27

P6/LineValidation.h
: Zawiera funkcjê waliduj¹c¹ wiersze z pliku CSV 31

P6/LogManager.cpp
: Implementacja klasy LogManager do obs³ugi logowania komunikatów 32

P6/LogManager.h
: Deklaracja klasy LogManager do obs³ugi logowania komunikatów 34

P6/main.cpp
: G³ówny plik programu obs³uguj¹cego analizê danych z pliku CSV 36

P6/RowData.cpp
: Implementacja klasy RowData do obs³ugi danych wierszy z pliku CSV 41

P6/RowData.h
: Deklaracja klasy RowData do przechowywania i przetwarzania danych z pliku CSV 42

P6/TreeData.cpp
: Implementacja klasy TreeData do przechowywania i analizy danych w strukturze drzewa . . . 43

P6/TreeData.h
: Deklaracja klasy TreeData do przechowywania i analizy danych w strukturze drzewa 46

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury TreeData::DayNode

Reprezentuje dane dzienne. Struktura ta zawiera informacje o danym dniu oraz mapê kwartalnych danych w tym dniu.

`#include <TreeData.h>`

Atrybuty publiczne

- int day

  Dzieñ miesi¹ca (1-31).
- std::map< int, QuarterNode > quarters

  Mapa kwartalnych danych w dniu, gdzie kluczem jest numer kwarta^{3}u.

3.1.1 Opis szczegÃ^{3}Åowy

Reprezentuje dane dzienne. Struktura ta zawiera informacje o danym dniu oraz mapê kwartalnych danych w tym dniu.

Definicja w linii 31 pliku TreeData.h.

3.1.2 Dokumentacja atrybutÃ^{3}w skÅadowych

3.1.2.1 day

`int TreeData::DayNode::day`

Dzieñ miesi¹ca (1-31).

Definicja w linii 32 pliku TreeData.h.

3.1.2.2 quarters

```
std::map<int, QuarterNode> TreeData::DayNode::quarters
```

Mapa kwartalnych danych w dniu, gdzie kluczem jest numer kwarta^{3}u.

Definicja w linii 33 pliku TreeData.h.

Dokumentacja dla tej struktury zostaÅa wygenerowana z pliku:

- P6/TreeData.h

3.2 Dokumentacja klasy LogManager

Klasa obs^{3}uguj¹ca logowanie komunikatów do plików tekstowych.

```
#include <LogManager.h>
```

Metody publiczne

- LogManager (const std::string &filename)
  Konstruktor klasy LogManager.
- ∼LogManager ()
  Destruktor klasy LogManager.
- void log (const std::string &message)
  Zapisuje komunikat do pliku logu.

Atrybuty prywatne

- std::ofstream logFile
  Strumieñ pliku logu, u¿ywany do zapisywania komunikatów.

3.2.1 Opis szczegÃ^{3}Åowy

Klasa obs^{3}uguj¹ca logowanie komunikatów do plików tekstowych.

Definicja w linii 12 pliku LogManager.h.

3.2.2 Dokumentacja konstruktora i destruktora

3.2.2.1 LogManager()

```
LogManager::LogManager (
            const std::string & filename)
```

Konstruktor klasy LogManager.

Tworzy plik logu z unikaln¹ nazw¹ opart¹ na aktualnej dacie i godzinie.

Parametry

filename	Nazwa podstawowa pliku logu, która bêdzie u¿yta do stworzenia pe^{3}nej nazwy pliku. Konstruktor inicjalizuje strumieñ wyjciowy do zapisywania komunikatów logu do pliku.

Tworzy plik logu z unikaln¹ nazw¹ opart¹ na aktualnej dacie i godzinie.

Parametry

filename	Nazwa podstawowa pliku logu, która bêdzie u¿yta do stworzenia pe^{3}nej nazwy pliku. Konstruktor generuje nazwê pliku logu opart¹ na dacie i godzinie, a nastêpnie otwiera ten plik do zapisu.

< Pobranie bie¿¹cego czasu.

< Struktura przechowuj¹ca czas w formacie lokalnym.

< Konwersja czasu na lokalny format.

< Strumieñ do tworzenia ^{3}añcucha tekstowego z nazw¹ pliku.

< Budowanie pe^{3}nej nazwy pliku.

< Nazwa pliku z dat¹ i godzin¹.

< Otwieranie pliku logu w trybie do zapisu.

< Rzucenie wyj¹tku, jeli nie uda^{3}o siê otworzyæ pliku.

Definicja w linii 30 pliku LogManager.cpp.

3.2.2.2 ~LogManager()

```
LogManager::~LogManager ()
```

Destruktor klasy LogManager.

Zamyka otwarty plik logu, zapewniaj¹c, ¿e wszystkie dane zosta^{3}y zapisane. < Zamkniêcie strumienia pliku logu.

Definicja w linii 51 pliku LogManager.cpp.

3.2.3 Dokumentacja funkcji skÅadowych

3.2.3.1 log()

```
void LogManager::log (
            const std::string & message)
```

Zapisuje komunikat do pliku logu.

Parametry

message	Komunikat do zapisania w pliku logu. Funkcja ta zapisuje podany komunikat do otwartego pliku logu.
message	Komunikat do zapisania w pliku logu. Funkcja zapisuje podany komunikat do pliku logu wraz z bie¿¹c¹ dat¹ i godzin¹.

< Pobranie bie¿¹cego czasu.

< Struktura przechowuj¹ca czas w formacie lokalnym.

< Konwersja czasu na lokalny format.

< Zapisanie komunikatu z dat¹ i godzin¹.

Definicja w linii 60 pliku LogManager.cpp.

3.2.4 Dokumentacja atrybutÃ³w skÅadowych

3.2.4.1 logFile

```
std::ofstream LogManager::logFile  [private]
```

Strumieñ pliku logu, u¿ywany do zapisywania komunikatów.

Definicja w linii 30 pliku LogManager.h.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ³w:

- P6/LogManager.h
- P6/LogManager.cpp

3.3 Dokumentacja struktury TreeData::MonthNode

Reprezentuje dane miesiêczne. Struktura ta zawiera informacje o danym miesi¹cu oraz mapê dziennych danych w tym miesi¹cu.

```
#include <TreeData.h>
```

Atrybuty publiczne

- int month

  Numer miesi¹ca (1-12).
- std::map< int, DayNode > days

  Mapa dziennych danych w miesi¹cu, gdzie kluczem jest numer dnia.

3.3.1 Opis szczegÃ³Åowy

Reprezentuje dane miesiêczne. Struktura ta zawiera informacje o danym miesi¹cu oraz mapê dziennych danych w tym miesi¹cu.

Definicja w linii 39 pliku TreeData.h.

3.3.2 Dokumentacja atrybutÃ³w skÅadowych

3.3.2.1 days

```
std::map<int, DayNode> TreeData::MonthNode::days
```

Mapa dziennych danych w miesi¹cu, gdzie kluczem jest numer dnia.

Definicja w linii 41 pliku TreeData.h.

3.3.2.2 month

```
int TreeData::MonthNode::month
```

Numer miesi¹ca (1-12).

Definicja w linii 40 pliku TreeData.h.

Dokumentacja dla tej struktury zostaÅa wygenerowana z pliku:

- P6/TreeData.h

3.4 Dokumentacja struktury TreeData::QuarterNode

Reprezentuje dane z podzia³em na kwarta³y dnia. Struktura ta zawiera informacje o godzinie, minucie oraz dane przypisane do danego kwarta³u.

```
#include <TreeData.h>
```

Atrybuty publiczne

- int quarter

  Numer kwarta³u (0-3), np. 0 - pierwsza czêæ godziny, 1 - druga czêæ godziny itd.
- int hour

  Godzina rozpoczêcia kwarta³u (0-23).
- int minute

  Minuta rozpoczêcia kwarta³u (0-59).
- std::vector< RowData > data

  Dane przypisane do kwarta³u, przechowywane jako wektor obiektów RowData.

3.4.1 Opis szczegÃ³Åowy

Reprezentuje dane z podziaÅem na kwarty dnia. Struktura ta zawiera informacje o godzinie, minucie oraz dane przypisane do danego kwartaÅu.

Definicja w linii 21 pliku TreeData.h.

3.4.2 Dokumentacja atrybutÃ³w skÅadowych

3.4.2.1 data

```
std::vector<RowData> TreeData::QuarterNode::data
```

Dane przypisane do kwartaÅu, przechowywane jako wektor obiektów RowData.

Definicja w linii 25 pliku TreeData.h.

3.4.2.2 hour

```
int TreeData::QuarterNode::hour
```

Godzina rozpoczêcia kwartaÅu (0-23).

Definicja w linii 23 pliku TreeData.h.

3.4.2.3 minute

```
int TreeData::QuarterNode::minute
```

Minuta rozpoczêcia kwartaÅu (0-59).

Definicja w linii 24 pliku TreeData.h.

3.4.2.4 quarter

```
int TreeData::QuarterNode::quarter
```

Numer kwartaÅu (0-3), np. 0 - pierwsza czêæ godziny, 1 - druga czêæ godziny itd.

Definicja w linii 22 pliku TreeData.h.

Dokumentacja dla tej struktury zostaÅa wygenerowana z pliku:

- P6/TreeData.h

3.5 Dokumentacja klasy RowData

Klasa reprezentuj¹ca dane jednego wiersza z pliku CSV, zawieraj¹ca ró¿ne parametry energetyczne.

`#include <RowData.h>`

Metody publiczne

- RowData (const string &line)

  Konstruktor przetwarzaj¹cy wiersz danych wejciowych w formacie CSV.
- RowData (ifstream &in)

  Konstruktor odczytuj¹cy dane z pliku binarnego.
- void display () const

  Wypisuje wszystkie dane obiektu na standardowe wyjcie. Funkcja ta drukuje pe³ne dane wiersza, w tym datê i wszystkie wartoci energetyczne.
- void displayData () const

  Wypisuje tylko wartoci energetyczne, pomijaj¹c datê, na standardowe wyjcie. Drukuje wartoci autokonsumpcji, eksportu, importu, poboru i produkcji, ale bez daty.
- string toString ()

  Zwraca dane w formie tekstowego ci¹gu znaków.
- void saveToBinary (ofstream &out) const

  Serializuje obiekt do pliku binarnego.
- void loadFromBinary (ifstream &in)

  Deserializuje obiekt z pliku binarnego.
- string getDate () const

  Zwraca datê, która znajduje siê w wierszu.
- float getSelfConsumption () const

  Zwraca wartoæ autokonsumpcji z wiersza danych.
- float getExport () const

  Zwraca wartoæ eksportu z wiersza danych.
- float getImport () const

  Zwraca wartoæ importu z wiersza danych.
- float getConsumption () const

  Zwraca wartoæ poboru energii z wiersza danych.
- float getProduction () const

  Zwraca wartoæ produkcji energii z wiersza danych.

Atrybuty prywatne

- string date

  Data wiersza w formacie tekstowym (np. "YYYY-MM-DD").
- float selfConsumption

  Autokonsumpcja w watach (W), iloæ energii zu¿ytej lokalnie.
- float exportValue

  Eksport energii w watach (W), iloæ energii oddanej do sieci.
- float importValue

  Import energii w watach (W), iloæ energii pobranej z sieci.
- float consumption

  Pobór energii z sieci w watach (W).
- float production

  Produkcja energii w watach (W), energia wytworzona przez system.

3.5.1 Opis szczegÃ³Åowy

Klasa reprezentuj¹ca dane jednego wiersza z pliku CSV, zawieraj¹ca ró¿ne parametry energetyczne.

Definicja w linii 17 pliku RowData.h.

3.5.2 Dokumentacja konstruktora i destruktora

3.5.2.1 RowData() [1/2]

```
RowData::RowData (
            const string & line)
```

Konstruktor przetwarzaj¹cy wiersz danych wejciowych w formacie CSV.

Konstruktor przetwarzaj¹cy wiersz danych z formatu CSV.

Parametry

line	Wiersz danych w formacie tekstowym, zawieraj¹cy ró¿ne wartoci oddzielone przecinkami. Przetwarza wiersz CSV na odpowiednie pola obiektu, konwertuj¹c wartoci na odpowiednie typy.
line	Wiersz danych wejciowych, zawieraj¹cy wartoci oddzielone przecinkami. Konstruktor dzieli wiersz na poszczególne elementy, konwertuje wartoci na odpowiednie typy i zapisuje je w odpowiednich polach obiektu.

< Wektor przechowuj¹cy rozdzielone wartoci z wiersza CSV.

< Strumieñ do rozdzielania wiersza na elementy.

< Zmienna do przechowywania pojedynczej wartoci wczytanej z wiersza.

< Usuwanie cudzysÃ³owów.

< Dodawanie wartoci do wektora.

< Data wiersza.

< Autokonsumpcja (w watach).

< Eksport energii (w watach).

< Import energii (w watach).

< Pobór energii (w watach).

< Produkcja energii (w watach).

< Logowanie wczytanego wiersza.

Definicja w linii 16 pliku RowData.cpp.

3.5.2.2 RowData() [2/2]

```
RowData::RowData (
            ifstream & in)
```

Konstruktor odczytuj¹cy dane z pliku binarnego.

Parametry

in	Strumieñ wejciowy, z którego odczytywane s¹ zserializowane dane. Inicjalizuje obiekt na podstawie zserializowanych danych zapisanych w pliku binarnym.
in	Strumieñ wejciowy, z którego wczytywane s¹ dane z pliku binarnego. Konstruktor ten deserializuje dane zapisane w pliku binarnym i inicjalizuje obiekt na ich podstawie.

< Deserializacja danych z pliku binarnego.

Definicja w linii 41 pliku RowData.cpp.

3.5.3 Dokumentacja funkcji skÅadowych

3.5.3.1 display()

```
void RowData::display () const
```

Wypisuje wszystkie dane obiektu na standardowe wyjcie. Funkcja ta drukuje pe[3]ne dane wiersza, w tym datê i wszystkie wartoci energetyczne.

Wypisuje wszystkie dane na standardowe wyjcie. Funkcja ta drukuje datê oraz wszystkie wartoci energetyczne obiektu wiersza.

Definicja w linii 47 pliku RowData.cpp.

3.5.3.2 displayData()

```
void RowData::displayData () const
```

Wypisuje tylko wartoci energetyczne, pomijaj¹c datê, na standardowe wyjcie. Drukuje wartoci autokonsumpcji, eksportu, importu, poboru i produkcji, ale bez daty.

Wypisuje tylko dane liczbowe (bez daty) na standardowe wyjcie. Drukuje tylko wartoci energetyczne, pomijaj¹c datê.

Definicja w linii 53 pliku RowData.cpp.

3.5.3.3 getConsumption()

```
float RowData::getConsumption () const  [inline]
```

Zwraca wartoæ poboru energii z wiersza danych.

Zwraca

Wartoæ poboru energii w watach (W) jako liczba zmiennoprzecinkowa. Funkcja ta zwraca wartoæ poboru energii z sieci przypisan¹ do danego wiersza danych.

Definicja w linii 75 pliku RowData.h.

3.5.3.4 getDate()

```
string RowData::getDate () const  [inline]
```

Zwraca datê, która znajduje siê w wierszu.

Zwraca

Data wiersza w formacie tekstowym (np. "YYYY-MM-DD"). Funkcja ta umo¿liwia pobranie daty przypisanej do danego wiersza danych.

Definicja w linii 55 pliku RowData.h.

3.5.3.5 getExport()

```
float RowData::getExport () const  [inline]
```

Zwraca wartoæ eksportu z wiersza danych.

Zwraca

Wartoæ eksportu w watach (W) jako liczba zmiennoprzecinkowa. Funkcja ta zwraca wartoæ eksportu energii przypisan¹ do danego wiersza.

Definicja w linii 65 pliku RowData.h.

3.5.3.6 getImport()

```
float RowData::getImport () const  [inline]
```

Zwraca wartoæ importu z wiersza danych.

Zwraca

Wartoæ importu w watach (W) jako liczba zmiennoprzecinkowa. Funkcja ta zwraca wartoæ importu energii dla danego wiersza.

Definicja w linii 70 pliku RowData.h.

3.5.3.7 getProduction()

```
float RowData::getProduction () const  [inline]
```

Zwraca wartoæ produkcji energii z wiersza danych.

Zwraca

Wartoæ produkcji energii w watach (W) jako liczba zmiennoprzecinkowa. Funkcja ta umo¿liwia dostêp do wartoci produkcji energii przypisanej do danego wiersza.

Definicja w linii 80 pliku RowData.h.

3.5.3.8 getSelfConsumption()

```
float RowData::getSelfConsumption () const  [inline]
```

Zwraca wartoæ autokonsumpcji z wiersza danych.

Zwraca

Wartoæ autokonsumpcji w watach (W) jako liczba zmiennoprzecinkowa. Funkcja ta umo¿liwia dostêp do wartoci autokonsumpcji dla danego wiersza danych.

Definicja w linii 60 pliku RowData.h.

3.5.3.9 loadFromBinary()

```
void RowData::loadFromBinary (
            ifstream & in)
```

Deserializuje obiekt z pliku binarnego.

Parametry

in	Strumieñ wejciowy, z którego wczytywane bêd¹ zserializowane dane obiektu. Inicjalizuje obiekt na podstawie danych w formacie binarnym.
in	Strumieñ wejciowy, z którego wczytywane bêd¹ zserializowane dane obiektu. Funkcja ta odczytuje dane zapisane w formacie binarnym i rekonstruuje obiekt na ich podstawie.

< Zmienna przechowuj¹ca rozmiar daty.

< Wczytanie rozmiaru daty.

< Zmienna do przechowywania daty.

< Wczytanie samej daty.

< Wczytanie wartoci autokonsumpcji.

< Wczytanie wartoci eksportu.

< Wczytanie wartoci importu.

< Wczytanie wartoci poboru.

< Wczytanie wartoci produkcji.

Definicja w linii 83 pliku RowData.cpp.

3.5.3.10 saveToBinary()

```
void RowData::saveToBinary (
            ofstream & out) const
```

Serializuje obiekt do pliku binarnego.

Parametry

out	Strumieñ wyjciowy, do którego zapisane bêd¹ dane obiektu w formacie binarnym. Funkcja ta zapisuje wszystkie wartoci obiektu w postaci binarnej, umo¿liwiaj¹c póniejsze odczytanie.
out	Strumieñ wyjciowy, do którego zapisane bêd¹ dane obiektu. Funkcja zapisuje dane obiektu w formacie binarnym, umo¿liwiaj¹c ich póniejsze odczytanie.

< Rozmiar daty wiersza w bajtach.

< Zapisanie rozmiaru daty.

< Zapisanie daty w postaci tekstowej.

< Zapisanie wartoci autokonsumpcji.

< Zapisanie wartoci eksportu.

< Zapisanie wartoci importu.

< Zapisanie wartoci poboru.

< Zapisanie wartoci produkcji.

Definicja w linii 69 pliku RowData.cpp.

3.5.3.11 toString()

```
string RowData::toString ()
```

Zwraca dane w formie tekstowego ci¹gu znaków.

Zwraca dane jako ci¹g znaków.

Zwraca

Dane w formacie tekstowym, które reprezentuj¹ wszystkie wartoci wiersza. Funkcja ta zamienia dane obiektu na jedn¹ liniê tekstu w celu ^{3}atwego zapisu lub wywietlenia.

Dane w formacie tekstowym, zawieraj¹ce wszystkie wartoci wiersza. Funkcja ta zamienia dane obiektu na ci¹g znaków, co mo¿e byæ przydatne do zapisania lub wywietlenia danych w postaci tekstowej.

Definicja w linii 61 pliku RowData.cpp.

3.5.4 Dokumentacja atrybutÃ^{3}w skÅadowych

3.5.4.1 consumption

```
float RowData::consumption  [private]
```

Pobór energii z sieci w watach (W).

Definicja w linii 87 pliku RowData.h.

3.5.4.2 date

```
string RowData::date  [private]
```

Data wiersza w formacie tekstowym (np. "YYYY-MM-DD").

Definicja w linii 83 pliku RowData.h.

3.5.4.3 exportValue

```
float RowData::exportValue  [private]
```

Eksport energii w watach (W), iloæ energii oddanej do sieci.

Definicja w linii 85 pliku RowData.h.

3.5.4.4 importValue

```
float RowData::importValue  [private]
```

Import energii w watach (W), iloæ energii pobranej z sieci.

Definicja w linii 86 pliku RowData.h.

3.5.4.5 production

```
float RowData::production  [private]
```

Produkcja energii w watach (W), energia wytworzona przez system.

Definicja w linii 88 pliku RowData.h.

3.5.4.6 selfConsumption

```
float RowData::selfConsumption  [private]
```

Autokonsumpcja w watach (W), iloæ energii zu¿ytej lokalnie.

Definicja w linii 84 pliku RowData.h.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ^{3}w:

- P6/RowData.h
- P6/RowData.cpp

3.6 Dokumentacja klasy TreeData

< Za^3^{1}czenie pliku nag^{3}ówkowego zawieraj¹cego klasê RowData.

```
#include <TreeData.h>
```

Komponenty

- struct DayNode

  Reprezentuje dane dzienne. Struktura ta zawiera informacje o danym dniu oraz mapê kwartalnych danych w tym dniu.
- struct MonthNode

  Reprezentuje dane miesiêczne. Struktura ta zawiera informacje o danym miesi¹cu oraz mapê dziennych danych w tym miesi¹cu.
- struct QuarterNode

  Reprezentuje dane z podzia³em na kwarta³y dnia. Struktura ta zawiera informacje o godzinie, minucie oraz dane przypisane do danego kwarta³u.
- struct YearNode

  Reprezentuje dane roczne. Struktura ta zawiera informacje o roku oraz mapê miesiêcznych danych w tym roku.

Metody publiczne

- void addData (const RowData &rowData)

  Dodaje dane do struktury drzewa.
- void print () const

  Wywietla ca³¹ strukturê drzewa.
- std::vector< RowData > getDataBetweenDates (const std::string &startDate, const std::string &endDate) const

  Pobiera dane w okrelonym przedziale czasowym.
- void calculateSumsBetweenDates (const std::string &startDate, const std::string &endDate, float &self↩ ConsumptionSum, float &exportSum, float &importSum, float &consumptionSum, float &productionSum) const

  Oblicza sumy danych w okrelonym przedziale czasowym.
- void calculateAveragesBetweenDates (const std::string &startDate, const std::string &endDate, float &self↩ ConsumptionAvg, float &exportAvg, float &importAvg, float &consumptionAvg, float &productionAvg) const

  Oblicza rednie wartoci danych w okrelonym przedziale czasowym.
- void compareDataBetweenDates (const std::string &startDate1, const std::string &endDate1, const std::string &startDate2, const std::string &endDate2, float &selfConsumptionDiff, float &exportDiff, float &importDiff, float &consumptionDiff, float &productionDiff) const

  Porównuje dane miêdzy dwoma zakresami czasowymi.
- std::vector< RowData > searchRecordsWithTolerance (const std::string &startDate, const std::string &end↩ Date, float value, float tolerance) const

  Wyszukuje rekordy w okrelonym zakresie czasowym z uwzglêdnieniem tolerancji.

Atrybuty prywatne

- std::map< int, YearNode > years

  Mapa lat, w których znajduj¹ siê dane w strukturze drzewa.

3.6.1 Opis szczegÃ³Åowy

< Za³¹czenie pliku nag³ówkowego zawieraj¹cego klasê RowData.

Klasa przechowuj¹ca dane w hierarchicznej strukturze drzewa na podstawie danych z pliku CSV. Klasa ta umo¿liwia organizowanie danych w strukturze drzewa, gdzie dane s¹ przechowywane wed³ug roku, miesi¹ca, dnia i kwarta³u. Dziêki tej strukturze mo¿liwa jest analiza oraz obliczenia na danych w zale¿noci od zakresu czasowego.

Definicja w linii 16 pliku TreeData.h.

3.6.2 Dokumentacja funkcji skÅadowych

3.6.2.1 addData()

```
void TreeData::addData (
            const RowData & rowData)
```

Dodaje dane do struktury drzewa.

Parametry

rowData	Obiekt RowData reprezentuj¹cy wiersz danych do dodania.

Funkcja ta dodaje dane do odpowiedniej pozycji w hierarchii drzewa na podstawie daty i czasu. Tworzy lub aktualizuje odpowiednie wêz^{3}y drzewa, wstawiaj¹c dane w odpowiednim roku, miesi¹cu, dniu i kwartale.

Parametry

rowData	Obiekt RowData reprezentuj¹cy dane wiersza.

Funkcja ta przetwarza dane z obiektu RowData i dodaje je do odpowiedniej lokalizacji w strukturze drzewa, w zale¿noci od daty, godziny, minuty oraz kwarta^{3}u. U¿ywane s¹ dane o roku, miesi¹cu, dniu, godzinie i minucie, aby odpowiednio wstawiæ dane do hierarchii. < Parsowanie ka¿dej czêci daty na liczbê

< Rok wyci¹gniêty z daty

< Miesi¹c wyci¹gniêty z daty

< Dzieñ wyci¹gniêty z daty

< Godzina wyci¹gniêta z daty

< Minuta wyci¹gniêta z daty

< Wyliczanie kwarta^{3}u na podstawie godziny i minuty

< Ustawienie roku w strukturze

< Ustawienie miesi¹ca w strukturze

< Ustawienie dnia w strukturze

< Ustawienie kwarta^{3}u w strukturze

< Ustawienie godziny w strukturze

< Ustawienie minuty w strukturze

< Dodanie danych do kwarta^{3}u

Definicja w linii 17 pliku TreeData.cpp.

3.6.2.2 calculateAveragesBetweenDates()

```
void TreeData::calculateAveragesBetweenDates (
            const std::string & startDate,
            const std::string & endDate,
            float & selfConsumptionAvg,
            float & exportAvg,
            float & importAvg,
            float & consumptionAvg,
            float & productionAvg) const
```

Oblicza rednie wartoci danych w okrelonym przedziale czasowym.

Oblicza rednie wartoci w okrelonym przedziale czasowym.

Parametry

	startDate	Data pocz¹tkowa w formacie dd.mm.yyyy hh:mm.
	endDate	Data koñcowa w formacie dd.mm.yyyy hh:mm.
out	*selfConsumptionAvg*	rednia autokonsumpcji w podanym okresie.
out	*exportAvg*	rednia eksportu w podanym okresie.
out	*importAvg*	rednia importu w podanym okresie.
out	*consumptionAvg*	rednia poboru w podanym okresie.
out	*productionAvg*	rednia produkcji w podanym okresie.

Funkcja ta oblicza rednie wartoci dla autokonsumpcji, eksportu, importu, poboru i produkcji na podstawie danych z wybranego okresu czasu.

Parametry

	startDate	Data pocz¹tkowa w formacie dd.mm.yyyy hh:mm.
	endDate	Data koñcowa w formacie dd.mm.yyyy hh:mm.
out	*selfConsumptionAvg*	rednia autokonsumpcji.
out	*exportAvg*	rednia eksportu.
out	*importAvg*	rednia importu.
out	*consumptionAvg*	rednia poboru.
out	*productionAvg*	rednia produkcji.

Funkcja ta przetwarza dane w okrelonym przedziale czasowym, obliczaj¹c rednie wartoci dla autokonsumpcji, eksportu, importu, poboru i produkcji. < Sumowanie autokonsumpcji

< Sumowanie eksportu

< Sumowanie importu

< Sumowanie poboru

< Sumowanie produkcji

< Zwiêkszanie liczby danych

< Obliczanie redniej autokonsumpcji

< Obliczanie redniej eksportu

< Obliczanie redniej importu

< Obliczanie redniej poboru

< Obliczanie redniej produkcji

Definicja w linii 171 pliku TreeData.cpp.

3.6.2.3 calculateSumsBetweenDates()

```
void TreeData::calculateSumsBetweenDates (
            const std::string & startDate,
            const std::string & endDate,
            float & selfConsumptionSum,
            float & exportSum,
            float & importSum,
            float & consumptionSum,
            float & productionSum) const
```

Oblicza sumy danych w okrelonym przedziale czasowym.

Oblicza sumy wartoci w okrelonym przedziale czasowym.

Parametry

	startDate	Data pocz¹tkowa w formacie dd.mm.yyyy hh:mm.
	endDate	Data koñcowa w formacie dd.mm.yyyy hh:mm.
out	*selfConsumptionSum*	Suma autokonsumpcji w podanym okresie.
out	*exportSum*	Suma eksportu w podanym okresie.
out	*importSum*	Suma importu w podanym okresie.
out	*consumptionSum*	Suma poboru w podanym okresie.
out	*productionSum*	Suma produkcji w podanym okresie.

Funkcja ta przeszukuje dane w podanym przedziale czasowym i oblicza sumy wartoci autokonsumpcji, eksportu, importu, poboru oraz produkcji.

Parametry

	startDate	Data pocz¹tkowa w formacie dd.mm.yyyy hh:mm.
	endDate	Data koñcowa w formacie dd.mm.yyyy hh:mm.
out	*selfConsumptionSum*	Suma autokonsumpcji.
out	*exportSum*	Suma eksportu.
out	*importSum*	Suma importu.
out	*consumptionSum*	Suma poboru.
out	*productionSum*	Suma produkcji.

Funkcja ta przetwarza dane w okrelonym przedziale czasowym i oblicza sumy dla poszczególnych parametrów: autokonsumpcji, eksportu, importu, poboru i produkcji. $<$ Dodanie wartoci autokonsumpcji

$<$ Dodanie wartoci eksportu

$<$ Dodanie wartoci importu

$<$ Dodanie wartoci poboru

$<$ Dodanie wartoci produkcji

Definicja w linii 140 pliku TreeData.cpp.

3.6.2.4 compareDataBetweenDates()

```
void TreeData::compareDataBetweenDates (
            const std::string & startDate1,
            const std::string & endDate1,
            const std::string & startDate2,
            const std::string & endDate2,
            float & selfConsumptionDiff,
            float & exportDiff,
            float & importDiff,
            float & consumptionDiff,
            float & productionDiff) const
```

Porównuje dane miêdzy dwoma zakresami czasowymi.

Parametry

	startDate1	Data pocz¹tkowa pierwszego zakresu.
	endDate1	Data koñcowa pierwszego zakresu.
	startDate2	Data pocz¹tkowa drugiego zakresu.
	endDate2	Data koñcowa drugiego zakresu.
out	*selfConsumptionDiff*	Ró¿nica autokonsumpcji miêdzy dwoma zakresami czasowymi.
out	*exportDiff*	Ró¿nica eksportu miêdzy dwoma zakresami czasowymi.
out	*importDiff*	Ró¿nica importu miêdzy dwoma zakresami czasowymi.
out	*consumptionDiff*	Ró¿nica poboru miêdzy dwoma zakresami czasowymi.
out	*productionDiff*	Ró¿nica produkcji miêdzy dwoma zakresami czasowymi.

Funkcja ta porównuje dane w dwóch okrelonych zakresach czasowych i oblicza ró¿nice w wartociach dla autokonsumpcji, eksportu, importu, poboru i produkcji.

3.6.2.5 getDataBetweenDates()

```
std::vector< RowData > TreeData::getDataBetweenDates (
            const std::string & startDate,
            const std::string & endDate) const
```

Pobiera dane w okrelonym przedziale czasowym.

Pobiera dane z drzewa w okrelonym przedziale czasowym.

Parametry

startDate	Data pocz¹tkowa w formacie dd.mm.yyyy hh:mm.
endDate	Data koñcowa w formacie dd.mm.yyyy hh:mm.

Zwraca

Wektor obiektów RowData, które mieszcz¹ siê w podanym przedziale czasowym.

Funkcja ta filtruje dane i zwraca wszystkie rekordy, które mieszcz¹ siê w okrelonym zakresie dat.

Parametry

startDate	Data pocz¹tkowa w formacie dd.mm.yyyy hh:mm.
endDate	Data koñcowa w formacie dd.mm.yyyy hh:mm.

Zwraca

Wektor obiektów RowData w podanym przedziale czasowym.

Funkcja ta iteruje po wszystkich danych w strukturze drzewa, porównuj¹c daty i zwracaj¹c dane znajduj¹ce siê w podanym przedziale czasowym. < Wektor do przechowywania wyników

< Wczytanie daty pocz¹tkowej

< Konwersja na typ time_t

< Czyszczenie strumienia

< Ustawienie strumienia na datê koñcow¹

< Wczytanie daty koñcowej

< Konwersja na typ time_t

< Pobieranie wêz^{3}a roku

< Pobieranie wêz^{3}a miesi¹ca

< Pobieranie wêz^{3}a dnia

< Pobieranie wêz^{3}a kwarta^{3}u

< Wczytanie daty z danych

< Konwersja na typ time_t

< Dodanie danych do wyników, jeli mieszcz¹ siê w przedziale

< Zwrócenie wyników

Definicja w linii 87 pliku TreeData.cpp.

3.6.2.6 print()

```
void TreeData::print () const
```

Wywietla ca^{3}¹ strukturê drzewa.

Wywietla zawartoæ drzewa na standardowym wyjciu.

Funkcja ta wypisuje ca^{3}¹ strukturê danych, pocz¹wszy od lat, przez miesi¹ce, dni, a¿ po kwarta^{3}y. Pozwala na wizualizacjê danych w drzewiastej strukturze hierarchicznej.

Funkcja ta rekurencyjnie przegl¹da wszystkie elementy drzewa, pocz¹wszy od lat, przez miesi¹ce, dni, kwarta^{3}y, a¿ po same dane. Wypisuje wszystkie dostêpne dane z poszczególnych wêz^{3}ów. < Pobieranie wêz^{3}a roku

< Wypisanie roku

< Pobieranie wêz^{3}a miesi¹ca

< Wypisanie miesi¹ca

< Pobieranie wêz^{3}a dnia

< Wypisanie dnia

< Pobieranie wêz^{3}a kwarta^{3}u

< Wypisanie kwarta^{3}u, godziny i minuty

< Wywo^{3}anie metody wypisuj¹cej dane z RowData

Definicja w linii 49 pliku TreeData.cpp.

3.6.2.7 searchRecordsWithTolerance()

```
std::vector< RowData > TreeData::searchRecordsWithTolerance (
            const std::string & startDate,
            const std::string & endDate,
            float value,
            float tolerance) const
```

Wyszukuje rekordy w okrelonym zakresie czasowym z uwzglêdnieniem tolerancji.

Parametry

startDate	Data pocz¹tkowa w formacie dd.mm.yyyy hh:mm.
endDate	Data koñcowa w formacie dd.mm.yyyy hh:mm.
value	Wartoæ wyszukiwana.
tolerance	Tolerancja dla wartoci wyszukiwania.

Zwraca

Wektor obiektów RowData, które spe^{3}niaj¹ kryteria wyszukiwania.

Funkcja ta umo¿liwia wyszukiwanie danych w okrelonym przedziale czasowym, uwzglêdniaj¹c tolerancjê dla wyszukiwanej wartoci.

3.6.3 Dokumentacja atrybutÃ^{3}w skÅadowych

3.6.3.1 years

```
std::map<int, YearNode> TreeData::years  [private]
```

Mapa lat, w których znajduj¹ siê dane w strukturze drzewa.

Definicja w linii 127 pliku TreeData.h.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ^{3}w:

- P6/TreeData.h
- P6/TreeData.cpp

3.7 Dokumentacja struktury TreeData::YearNode

Reprezentuje dane roczne. Struktura ta zawiera informacje o roku oraz mapê miesiêcznych danych w tym roku.

```
#include <TreeData.h>
```

Atrybuty publiczne

- int year

  Rok (np. 2023).
- std::map< int, MonthNode > months

  Mapa miesiêcznych danych w roku, gdzie kluczem jest numer miesi¹ca.

3.7.1 Opis szczegÃ³Åowy

Reprezentuje dane roczne. Struktura ta zawiera informacje o roku oraz mapê miesiêcznych danych w tym roku.

Definicja w linii 47 pliku TreeData.h.

3.7.2 Dokumentacja atrybutÃ³w skÅadowych

3.7.2.1 months

```
std::map<int, MonthNode> TreeData::YearNode::months
```

Mapa miesiêcznych danych w roku, gdzie kluczem jest numer miesi¹ca.

Definicja w linii 49 pliku TreeData.h.

3.7.2.2 year

```
int TreeData::YearNode::year
```

Rok (np. 2023).

Definicja w linii 48 pliku TreeData.h.

Dokumentacja dla tej struktury zostaÅa wygenerowana z pliku:

- P6/TreeData.h

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku GoogleTest/test.cpp

Zawiera zestaw testów jednostkowych dla modu^{3}ów RowData, TreeData, Logger i LineValidation.

```
#include "pch.h"
#include "../P6/RowData.h"
#include "../P6/RowData.cpp"
#include "../P6/TreeData.h"
#include "../P6/TreeData.cpp"
#include "../P6/LogManager.h"
#include "../P6/LogManager.cpp"
#include "../P6/LineValidation.h"
```

Funkcje

- TEST (RowDataTest, ConstructorFromString)

  Testuje konstruktor klasy RowData.
- TEST (RowDataTest, Serialization)

  Testuje serializacjê i deserializacjê klasy RowData.
- TEST (LineValidationTest, EmptyLine)

  Testuje funkcjê waliduj¹c¹ puste linie.
- TEST (LineValidationTest, HeaderLine)

  Testuje funkcjê waliduj¹c¹ linie z nag^{3}ówkami.
- TEST (LineValidationTest, LineWithInvalidCharacters)

  Testuje funkcjê waliduj¹c¹ linie z nieprawid^{3}owymi znakami.
- TEST (TreeDataTest, AddData)

  Testuje dodawanie danych do struktury drzewa.
- TEST (TreeDataTest, CalculateSumsBetweenDates)

  Testuje obliczanie sum w strukturze drzewa.

4.1.1 Opis szczegÃ^{3}Åowy

Zawiera zestaw testów jednostkowych dla modu^{3}ów RowData, TreeData, Logger i LineValidation.

Testy jednostkowe zosta^{3}y zaimplementowane z u¿yciem frameworka GoogleTest.

Definicja w pliku test.cpp.

4.1.2 Dokumentacja funkcji

4.1.2.1 TEST() [1/7]

```
TEST (
            LineValidationTest ,
            EmptyLine )
```

Testuje funkcjê waliduj¹c¹ puste linie.

Sprawdza, czy funkcja zwraca false dla pustej linii. < Walidacja pustej linii.

Definicja w linii 54 pliku test.cpp.

4.1.2.2 TEST() [2/7]

```
TEST (
            LineValidationTest ,
            HeaderLine )
```

Testuje funkcjê waliduj¹c¹ linie z nag^{3}ówkami.

Sprawdza, czy funkcja zwraca false dla linii z nag^{3}ówkiem. < Walidacja linii z nag^{3}ówkiem.

Definicja w linii 61 pliku test.cpp.

4.1.2.3 TEST() [3/7]

```
TEST (
            LineValidationTest ,
            LineWithInvalidCharacters )
```

Testuje funkcjê waliduj¹c¹ linie z nieprawid^{3}owymi znakami.

Sprawdza, czy funkcja zwraca false dla linii zawieraj¹cych nieprawid^{3}owe znaki. < Walidacja linii z nieprawid^{3}owymi znakami.

Definicja w linii 68 pliku test.cpp.

4.1.2.4 TEST() [4/7]

```
TEST (
            RowDataTest ,
            ConstructorFromString )
```

Testuje konstruktor klasy RowData.

Sprawdza, czy wartoci wczytane z ci¹gu znaków s¹ poprawnie przypisane.

Definicja w linii 18 pliku test.cpp.

4.1.2.5 TEST() [5/7]

```
TEST (
            RowDataTest ,
            Serialization )
```

Testuje serializacjê i deserializacjê klasy RowData.

Sprawdza, czy dane zapisane do pliku binarnego s[1] poprawnie odczytywane. < Serializacja danych do pliku binarnego.

< Deserializacja danych z pliku binarnego.

Definicja w linii 32 pliku test.cpp.

4.1.2.6 TEST() [6/7]

```
TEST (
            TreeDataTest ,
            AddData )
```

Testuje dodawanie danych do struktury drzewa.

Sprawdza, czy dane s[1] poprawnie przechowywane w strukturze drzewa. < Dodanie danych do struktury drzewa.

< Sprawdzenie, czy dodany rekord jest w drzewie.

Definicja w linii 75 pliku test.cpp.

4.1.2.7 TEST() [7/7]

```
TEST (
            TreeDataTest ,
            CalculateSumsBetweenDates )
```

Testuje obliczanie sum w strukturze drzewa.

Sprawdza, czy sumy s[1] poprawnie obliczane w okrelonym przedziale czasowym. < Obliczanie sum w okrelonym przedziale czasowym.

< Weryfikacja sum dla autokonsumpcji.

< Weryfikacja sum dla eksportu.

< Weryfikacja sum dla importu.

< Weryfikacja sum dla poboru.

< Weryfikacja sum dla produkcji.

Definicja w linii 93 pliku test.cpp.

4.2 test.cpp

Idź do dokumentacji tego pliku.

```
00001
00004
00005 #include "pch.h"
00006 #include "../P6/RowData.h"
00007 #include "../P6/RowData.cpp"
00008 #include "../P6/TreeData.h"
00009 #include "../P6/TreeData.cpp"
00010 #include "../P6/LogManager.h"
00011 #include "../P6/LogManager.cpp"
00012 #include "../P6/LineValidation.h"
00013
00014 using namespace std;
00015
00018 TEST(RowDataTest, ConstructorFromString) {
00019     string line = "15.10.2023 12:00:00,100.5,200.5,300.5,400.5,500.5";
00020     RowData rd(line);
00021
00022     EXPECT_EQ(rd.getDate(), "15.10.2023 12:00:00");
00023     EXPECT_FLOAT_EQ(rd.getConsumption(), 100.5);
00024     EXPECT_FLOAT_EQ(rd.getExport(), 200.5);
00025     EXPECT_FLOAT_EQ(rd.getImport(), 300.5);
00026     EXPECT_FLOAT_EQ(rd.getSelfConsumption(), 400.5);
00027     EXPECT_FLOAT_EQ(rd.getProduction(), 500.5);
00028 }
00029
00032 TEST(RowDataTest, Serialization) {
00033     string line = "15.10.2023 12:00:00,100.5,200.5,300.5,400.5,500.5";
00034     RowData rd(line);
00035
00036     ofstream out("test.bin", ios::binary);
00037     rd.saveToBinary(out);
00038     out.close();
00039
00040     ifstream in("test.bin", ios::binary);
00041     RowData rd2(in);
00042     in.close();
00043
00044     EXPECT_EQ(rd2.getDate(), "15.10.2023 12:00:00");
00045     EXPECT_FLOAT_EQ(rd2.getSelfConsumption(), 100.5);
00046     EXPECT_FLOAT_EQ(rd2.getExport(), 200.5);
00047     EXPECT_FLOAT_EQ(rd2.getImport(), 300.5);
00048     EXPECT_FLOAT_EQ(rd2.getConsumption(), 400.5);
00049     EXPECT_FLOAT_EQ(rd2.getProduction(), 500.5);
00050 }
00051
00054 TEST(LineValidationTest, EmptyLine) {
00055     string line = "";
00056     EXPECT_FALSE(lineValidation(line));
00057 }
00058
00061 TEST(LineValidationTest, HeaderLine) {
00062     string line = "Time,Autokonsumpcja (W),Eksport (W),Import (W),Pobor (W),Produkcja (W)";
00063     EXPECT_FALSE(lineValidation(line));
00064 }
00065
00068 TEST(LineValidationTest, LineWithInvalidCharacters) {
00069     string line = "2023-10-15 12:00:00,X,200.5,300.5,400.5,500.5";
00070     EXPECT_FALSE(lineValidation(line));
00071 }
00072
00075 TEST(TreeDataTest, AddData) {
00076     TreeData treeData;
00077     string line = "15.10.2023 12:00:00,100.5,200.5,300.5,400.5,500.5";
00078     RowData rd(line);
00079     treeData.addData(rd);
00080
00081     vector<RowData> data = treeData.getDataBetweenDates("15.10.2023 00:00", "15.10.2023 23:59");
00082     ASSERT_EQ(data.size(), 1);
00083     EXPECT_EQ(data[0].getDate(), "15.10.2023 12:00:00");
00084     EXPECT_FLOAT_EQ(data[0].getSelfConsumption(), 100.5);
00085     EXPECT_FLOAT_EQ(data[0].getExport(), 200.5);
00086     EXPECT_FLOAT_EQ(data[0].getImport(), 300.5);
00087     EXPECT_FLOAT_EQ(data[0].getConsumption(), 400.5);
00088     EXPECT_FLOAT_EQ(data[0].getProduction(), 500.5);
00089 }
00090
00093 TEST(TreeDataTest, CalculateSumsBetweenDates) {
00094     TreeData treeData;
00095     string line1 = "15.10.2023 12:00:00,100.5,200.5,300.5,400.5,500.5";
00096     string line2 = "15.10.2023 18:00:00,150.5,250.5,350.5,450.5,550.5";
00097     RowData rd1(line1);
00098     RowData rd2(line2);
```

```
00099     treeData.addData(rd1);
00100     treeData.addData(rd2);
00101
00102     float autokonsumpcjaSum, eksportSum, importSum, poborSum, produkcjaSum;
00103     treeData.calculateSumsBetweenDates("15.10.2023 00:00", "15.10.2023 23:59", autokonsumpcjaSum,
      eksportSum, importSum, poborSum, produkcjaSum);
00104
00105     EXPECT_FLOAT_EQ(autokonsumpcjaSum, 251.0);
00106     EXPECT_FLOAT_EQ(eksportSum, 451.0);
00107     EXPECT_FLOAT_EQ(importSum, 651.0);
00108     EXPECT_FLOAT_EQ(poborSum, 851.0);
00109     EXPECT_FLOAT_EQ(produkcjaSum, 1051.0);
00110 }
```

4.3 Dokumentacja pliku P6/LineValidation.h

Zawiera funkcjê waliduj¹c¹ wiersze z pliku CSV.

```
#include <string>
#include <cctype>
#include <algorithm>
#include "LogManager.h"
```

Funkcje

- bool lineValidation (const std::string &line)
  < Za³ ¹czenie pliku nag³ ówkowego do obs³ ugi logowania.

4.3.1 Opis szczegÃ³Åowy

Zawiera funkcjê waliduj¹c¹ wiersze z pliku CSV.

Definicja w pliku LineValidation.h.

4.3.2 Dokumentacja funkcji

4.3.2.1 lineValidation()

```
bool lineValidation (
            const std::string & line)
```

< Za³ ¹czenie pliku nag³ ówkowego do obs³ ugi logowania.

Funkcja sprawdzaj¹ca poprawnoæ wiersza danych.

Sprawdza, czy wiersz zawiera odpowiedni¹ liczbê parametrów, czy nie jest pusty i czy nie zawiera liter.

Parametry

line	Wiersz danych wejciowych.

Zwraca

true, jeli wiersz jest poprawny, false w przeciwnym razie. Funkcja ta waliduje wiersz CSV poprzez sprawdzenie jego zawartoci, takich jak liczba parametrów oraz obecnoæ niepo¿¹danych liter.

< Logowanie b^{3}êdu, gdy linia jest pusta.

< Logowanie b^{3}êdu, gdy wiersz zawiera nag^{3}ówek.

< Logowanie b^{3}êdu, gdy wiersz zawiera litery.

< Logowanie b^{3}êdu, gdy liczba parametrów jest niepoprawna.

< Zwracanie true, jeli wiersz jest poprawny.

Definicja w linii 17 pliku LineValidation.h.

4.4 LineValidation.h

IdÅº do dokumentacji tego pliku.

```
00001
00003
00004 #ifndef LINEVALIDATION_H
00005 #define LINEVALIDATION_H
00006
00007 #include <string>
00008 #include <cctype>
00009 #include <algorithm>
00010 #include "LogManager.h"
00011
00017 bool lineValidation(const std::string& line)
00018 {
00019     // Sprawdzenie, czy wiersz jest pusty.
00020     if (line.empty())
00021     {
00022         errorLogger.log("Pusta linia");
00023         return false;
00024     }
00025     // Sprawdzenie, czy wiersz zawiera nag³ówek.
00026     else if (line.find("Time") != std::string::npos)
00027     {
00028         errorLogger.log("Znaleziono nag³ówek: " + line);
00029         return false;
00030     }
00031     // Sprawdzenie, czy wiersz zawiera litery.
00032     else if (std::any_of(line.begin(), line.end(), [](char c) { return std::isalpha(c); }))
00033     {
00034         errorLogger.log("Znaleziono inne dane: " + line);
00035         return false;
00036     }
00037     // Sprawdzenie, czy wiersz zawiera odpowiedni¹ liczbê parametrów (5 przecinków).
00038     else if (std::count(line.begin(), line.end(), ',') != 5)
00039     {
00040         errorLogger.log("Nieprawid³owa liczba parametrów: " + line);
00041         return false;
00042     }
00043     else
00044     {
00045         return true;
00046     }
00047 }
00048
00049 #endif // LINEVALIDATION_H
```

4.5 Dokumentacja pliku P6/LogManager.cpp

Implementacja klasy LogManager do obs^{3}ugi logowania komunikatów.

```
#include "LogManager.h"
#include <iomanip>
#include <ctime>
#include <cstdio>
#include <sstream>
```

Zmienne

- LogManager globalLogger ("log")
- LogManager errorLogger ("log_error")
- int errorLogCount = 0

  Licznik wyst¹pieñ b^{3}êdów logowanych przez errorLogger. Zmienna ta przechowuje liczbê b^{3}êdów zarejestrowanych przez errorLogger.

4.5.1 Opis szczegÃ^{3}Åowy

Implementacja klasy LogManager do obs^{3}ugi logowania komunikatów.

Definicja w pliku LogManager.cpp.

4.5.2 Dokumentacja zmiennych

4.5.2.1 errorLogCount

```
int errorLogCount = 0
```

Licznik wyst¹pieñ b^{3}êdów logowanych przez errorLogger. Zmienna ta przechowuje liczbê b^{3}êdów zarejestrowanych przez errorLogger.

Licznik wyst¹pieñ b^{3}êdów logowanych przez errorLogger. Licznik jest zwiêkszany za ka¿dym razem, gdy logger↩
Error zapisuje komunikat b^{3}êdu.

Definicja w linii 23 pliku LogManager.cpp.

4.5.2.2 errorLogger

```
LogManager errorLogger("log_error") (
            "log_error" )
```

4.5.2.3 globalLogger

```
LogManager globalLogger("log") (
            "log" )
```

4.6 LogManager.cpp

IdÅº do dokumentacji tego pliku.

```
00001
00003
00004 #include "LogManager.h"
00005 #include <iomanip>
00006 #include <ctime>
00007 #include <cstdio>
00008 #include <sstream>
00009
00013 LogManager globalLogger("log");
00014
00018 LogManager errorLogger("log_error");
00019
00023 int errorLogCount = 0;
00024
00025
00030 LogManager::LogManager(const std::string& filename) {
00031     auto t = std::time(nullptr);
00032     std::tm tm;
00033     localtime_s(&tm, &t);
00034     std::ostringstream oss;
00035     oss « filename « "_" « std::put_time(&tm, "%d%m%Y_%H%M%S") « ".txt";
00036     std::string datedFilename = oss.str();
00037
00038     // Usuwanie istniej¹cego pliku logu, jeli ju¿ istnieje.
00039     if (std::remove(datedFilename.c_str()) != 0) {
00040         // Plik nie istnieje lub nie mo¿na go usun¹æ.
00041     }
00042
00043     logFile.open(datedFilename, std::ios::out | std::ios::app);
00044     if (!logFile.is_open()) {
00045         throw std::runtime_error("Nie mo¿na otworzyæ pliku logu");
00046     }
00047 }
00048
00051 LogManager::~LogManager() {
00052     if (logFile.is_open()) {
00053         logFile.close();
00054     }
00055 }
00056
00060 void LogManager::log(const std::string& message) {
00061     if (logFile.is_open()) {
00062         auto t = std::time(nullptr);
00063         std::tm tm;
00064         localtime_s(&tm, &t);
00065         logFile « std::put_time(&tm, "%d.%m.%Y %H:%M:%S") « " " « message « std::endl;
00066     }
00067
00068     // Zwiêkszanie licznika b³êdów, jeli logowany komunikat pochodzi z errorLogger.
00069     if (this == &errorLogger) {
00070         ++errorLogCount;
00071     }
00072 }
```

4.7 Dokumentacja pliku P6/LogManager.h

Deklaracja klasy LogManager do obs³ugi logowania komunikatów.

```
#include <fstream>
#include <string>
```

Komponenty

- class LogManager

  Klasa obs³uguj¹ca logowanie komunikatów do plików tekstowych.

Zmienne

- LogManager globalLogger

  Globalny logger dla standardowych komunikatów. Logger ten jest u¿ywany do zapisywania komunikatów informacyjnych oraz standardowych.
- LogManager errorLogger

  Globalny logger dla komunikatów b^{3}êdów. Logger ten jest u¿ywany do zapisywania komunikatów zwi¹zanych z b^{3}êdami oraz wyj¹tkami.
- int errorLogCount

  Licznik wyst¹pieñ b^{3}êdów logowanych przez errorLogger. Zmienna ta przechowuje liczbê b^{3}êdów zarejestrowanych przez errorLogger.

4.7.1 Opis szczegÃ^{3}Åowy

Deklaracja klasy LogManager do obs^{3}ugi logowania komunikatów.

Definicja w pliku LogManager.h.

4.7.2 Dokumentacja zmiennych

4.7.2.1 errorLogCount

```
int errorLogCount  [extern]
```

Licznik wyst¹pieñ b^{3}êdów logowanych przez errorLogger. Zmienna ta przechowuje liczbê b^{3}êdów zarejestrowanych przez errorLogger.

Licznik wyst¹pieñ b^{3}êdów logowanych przez errorLogger. Licznik jest zwiêkszany za ka¿dym razem, gdy logger↩ Error zapisuje komunikat b^{3}êdu.

Definicja w linii 23 pliku LogManager.cpp.

4.7.2.2 errorLogger

```
LogManager errorLogger  [extern]
```

Globalny logger dla komunikatów b^{3}êdów. Logger ten jest u¿ywany do zapisywania komunikatów zwi¹zanych z b^{3}êdami oraz wyj¹tkami.

Globalny logger dla komunikatów b^{3}êdów. Logger ten jest u¿ywany do zapisywania komunikatów zwi¹zanych z b^{3}êdami do pliku.

4.7.2.3 globalLogger

```
LogManager globalLogger  [extern]
```

Globalny logger dla standardowych komunikatów. Logger ten jest u¿ywany do zapisywania komunikatów informacyjnych oraz standardowych.

Globalny logger dla standardowych komunikatów. Logger ten jest u¿ywany do zapisywania standardowych komunikatów informacyjnych do pliku.

4.8 LogManager.h

Idź do dokumentacji tego pliku.

```
00001
00003
00004 #ifndef LOGMANAGER_H
00005 #define LOGMANAGER_H
00006
00007 #include <fstream>
00008 #include <string>
00009
00012 class LogManager {
00013 public:
00018     LogManager(const std::string& filename);
00019
00022     ~LogManager();
00023
00027     void log(const std::string& message);
00028
00029 private:
00030     std::ofstream logFile;
00031 };
00032
00036 extern LogManager globalLogger;
00037
00038
00042 extern LogManager errorLogger;
00043
00044
00048 extern int errorLogCount;
00049
00050 #endif // LOGMANAGER_H
```

4.9 Dokumentacja pliku P6/main.cpp

G^{3}ówny plik programu obs^{3}uguj¹cego analizê danych z pliku CSV.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include "RowData.h"
#include "LogManager.h"
#include "TreeData.h"
#include "LineValidation.h"
```

Funkcje

- void displayMenu ()

  Wywietla menu u¿ytkownika.
- int main ()

  Funkcja g^{3}ówna programu.

4.9.1 Opis szczegÃ^{3}Åowy

G^{3}ówny plik programu obs^{3}uguj¹cego analizê danych z pliku CSV.

Program umo¿liwia wczytywanie danych z pliku CSV, ich analizê, przetwarzanie oraz zapisywanie w pliku binarnym. Oferuje funkcje takie jak obliczanie sum i rednich, porównywanie danych oraz wyszukiwanie z tolerancj¹.

Definicja w pliku main.cpp.

4.9.2 Dokumentacja funkcji

4.9.2.1 displayMenu()

```
void displayMenu ()
```

Wywietla menu u¿ytkownika.

Funkcja drukuje dostêpne opcje programu na standardowe wyjcie.

Definicja w linii 21 pliku main.cpp.

4.9.2.2 main()

```
int main ()
```

Funkcja g^{3}ówna programu.

G^{3}ówna pêtla programu, która obs^{3}uguje menu i poszczególne funkcjonalnoci.

Zwraca

Zwraca 0 w przypadku pomylnego zakoñczenia programu.

< Struktura drzewa do przechowywania danych.

< Zmienna przechowuj¹ca dane w postaci wierszy.

< Aktualnie przetwarzany wiersz danych.

< Strumieñ do odczytu pliku CSV.

< Daty u¿ywane w analizie danych.

< Wyniki obliczeñ sum.

< Wyniki porównañ.

< Parametry do wyszukiwania z tolerancj¹.

< Zmienna do przechowywania wyników wyszukiwania z tolerancj¹.

< Wywietlanie menu u¿ytkownika.

< Ignoruj znak nowej linii pozostawiony w buforze.

Wczytanie danych z pliku CSV.

Dane s¹ wczytywane do struktury drzewa i wektora, a niepoprawne wiersze s¹ logowane.

< Walidacja wiersza danych.

< Tworzenie obiektu RowData z wiersza CSV.

< Dodanie wiersza do wektora danych.

< Dodanie wiersza do struktury drzewa.

Wywietlenie struktury drzewa.

< Wywietlanie struktury drzewa danych.

Pobranie danych w okrelonym przedziale czasowym.

< Pobranie danych z drzewa w podanym zakresie dat.

< Iteracja po danych w przedziale czasowym.

< Wywietlanie danych.

Obliczenie sum w okrelonym przedziale czasowym.

< Obliczanie sum dla danych z przedzia^{3}u czasowego.

Obliczenie rednich w okrelonym przedziale czasowym.

< Obliczanie rednich dla danych z przedzia^{3}u czasowego.

Porównanie danych miêdzy dwoma zakresami czasowymi.

< Porównanie danych miêdzy dwoma zakresami czasowymi.

Wyszukiwanie danych w okrelonym przedziale czasowym z tolerancj[1].

< Wyszukiwanie rekordów z tolerancj[1] w podanym zakresie dat.

< Iteracja po wynikach wyszukiwania.

< Wywietlanie wyników wyszukiwania.

Zapisanie danych do pliku binarnego.

< Zapisanie wiersza danych do pliku binarnego.

Wczytanie danych z pliku binarnego.

< Wczytanie wiersza danych z pliku binarnego.

< Dodanie wiersza danych do struktury drzewa.

Wyjcie z programu.

Definicja w linii 39 pliku main.cpp.

4.10 main.cpp

IdÅº do dokumentacji tego pliku.

```
00001
00005
00006 #include <iostream>
00007 #include <fstream>
00008 #include <string>
00009 #include <sstream>
00010 #include <vector>
00011
00012 #include "RowData.h"
00013 #include "LogManager.h"
00014 #include "TreeData.h"
00015 #include "LineValidation.h"
00016
00017 using namespace std;
00018
00021 void displayMenu() {
00022     cout « "Menu:" « endl;
00023     cout « "1. Load data from file" « endl;
00024     cout « "2. Print tree structure" « endl;
00025     cout « "3. Get data between dates" « endl;
00026     cout « "4. Calculate sums between dates" « endl;
00027     cout « "5. Calculate averages between dates" « endl;
00028     cout « "6. Compare data between dates" « endl;
00029     cout « "7. Search records with tolerance" « endl;
00030     cout « "8. Save data to binary file" « endl;
00031     cout « "9. Load data from binary file" « endl;
00032     cout « "10. Exit" « endl;
00033     cout « "Enter your choice: ";
00034 }
00035
00039 int main() {
00040     TreeData treeData;
00041     vector<RowData> data;
00042     string line;
00043     ifstream file;
00044     string startDate, endDate, startDate1, endDate1, startDate2, endDate2;
00045     float autokonsumpcjaSum, eksportSum, importSum, poborSum, produkcjaSum;
00046     float autokonsumpcjaDiff, eksportDiff, importDiff, poborDiff, produkcjaDiff;
00047     float searchValue, tolerance;
00048     vector<RowData> filteredData, recordsWithTolerance;
00049
00050     while (true) {
00051         displayMenu();
00052         int choice;
00053         cin » choice;
00054         cin.ignore();
00055
00056         switch (choice) {
00057         case 1:
00060             file.open("Chart Export.csv");
00061             if (!file.is_open()) {
00062                 cerr « "Error opening file" « endl;
00063                 return 1;
00064             }
00065
00066             while (getline(file, line)) {
00067                 if (lineValidation(line)) {
00068                     RowData rd(line);
00069                     data.push_back(rd);
00070                     treeData.addData(rd);
00071                 }
00072             }
00073
00074             file.close();
00075             cout « "Data loaded successfully." « endl;
00076             cout « "Loaded " « data.size() « " lines" « endl;
00077             cout « "Found " « errorLogCount « " faulty lines" « endl;
00078             cout « "Check log and log_error files for more details" « endl;
00079             break;
00080
00081         case 2:
00083             treeData.print();
00084             break;
00085
00086         case 3:
00088             cout « "Enter start date (dd.mm.yyyy hh:mm): ";
00089             getline(cin, startDate);
00090             cout « "Enter end date (dd.mm.yyyy hh:mm): ";
00091             getline(cin, endDate);
00092             filteredData = treeData.getDataBetweenDates(startDate, endDate);
00093             cout « "Data between " « startDate « " and " « endDate « ":" « endl;
00094             for (const auto& rd : filteredData) {
```

```
00095                 rd.display();
00096             }
00097             break;
00098
00099         case 4:
00101             cout « "Enter start date (dd.mm.yyyy hh:mm): ";
00102             getline(cin, startDate);
00103             cout « "Enter end date (dd.mm.yyyy hh:mm): ";
00104             getline(cin, endDate);
00105             treeData.calculateSumsBetweenDates(startDate, endDate, autokonsumpcjaSum, eksportSum,
      importSum, poborSum, produkcjaSum);
00106             cout « "Sums between " « startDate « " and " « endDate « ":" « endl;
00107             cout « "Autokonsumpcja: " « autokonsumpcjaSum « endl;
00108             cout « "Eksport: " « eksportSum « endl;
00109             cout « "Import: " « importSum « endl;
00110             cout « "Pobór: " « poborSum « endl;
00111             cout « "Produkcja: " « produkcjaSum « endl;
00112             break;
00113
00114         case 5:
00116             cout « "Enter start date (dd.mm.yyyy hh:mm): ";
00117             getline(cin, startDate);
00118             cout « "Enter end date (dd.mm.yyyy hh:mm): ";
00119             getline(cin, endDate);
00120             treeData.calculateAveragesBetweenDates(startDate, endDate, autokonsumpcjaSum, eksportSum,
      importSum, poborSum, produkcjaSum);
00121             cout « "Averages between " « startDate « " and " « endDate « ":" « endl;
00122             cout « "Autokonsumpcja: " « autokonsumpcjaSum « endl;
00123             cout « "Eksport: " « eksportSum « endl;
00124             cout « "Import: " « importSum « endl;
00125             cout « "Pobór: " « poborSum « endl;
00126             cout « "Produkcja: " « produkcjaSum « endl;
00127             break;
00128
00129         case 6:
00131             cout « "Enter first start date (dd.mm.yyyy hh:mm): ";
00132             getline(cin, startDate1);
00133             cout « "Enter first end date (dd.mm.yyyy hh:mm): ";
00134             getline(cin, endDate1);
00135             cout « "Enter second start date (dd.mm.yyyy hh:mm): ";
00136             getline(cin, startDate2);
00137             cout « "Enter second end date (dd.mm.yyyy hh:mm): ";
00138             getline(cin, endDate2);
00139             treeData.compareDataBetweenDates(startDate1, endDate1, startDate2, endDate2,
      autokonsumpcjaDiff, eksportDiff, importDiff, poborDiff, produkcjaDiff);
00140             cout « "Differences between ranges:" « endl;
00141             cout « "Autokonsumpcja: " « autokonsumpcjaDiff « endl;
00142             cout « "Eksport: " « eksportDiff « endl;
00143             cout « "Import: " « importDiff « endl;
00144             cout « "Pobór: " « poborDiff « endl;
00145             cout « "Produkcja: " « produkcjaDiff « endl;
00146             break;
00147
00148         case 7:
00150             cout « "Enter start date (dd.mm.yyyy hh:mm): ";
00151             getline(cin, startDate);
00152             cout « "Enter end date (dd.mm.yyyy hh:mm): ";
00153             getline(cin, endDate);
00154             cout « "Enter search value: ";
00155             cin » searchValue;
00156             cout « "Enter tolerance: ";
00157             cin » tolerance;
00158             recordsWithTolerance = treeData.searchRecordsWithTolerance(startDate, endDate,
      searchValue, tolerance);
00159             cout « "Records within tolerance:" « endl;
00160             for (const auto& rd : recordsWithTolerance) {
00161                 rd.display();
00162             }
00163             break;
00164
00165         case 8:
00167         {
00168             ofstream binaryFile("data.bin", ios::binary);
00169             if (!binaryFile.is_open()) {
00170                 cerr « "Error opening binary file" « endl;
00171                 return 1;
00172             }
00173             for (const auto& rd : data) {
00174                 rd.saveToBinary(binaryFile);
00175             }
00176             binaryFile.close();
00177             cout « "Data saved successfully." « endl;
00178         }
00179         break;
00180
00181         case 9:
00183         {
```

```
00184             ifstream binaryFileIn("data.bin", ios::binary);
00185             if (!binaryFileIn.is_open()) {
00186                 cerr « "Error opening binary file for reading" « endl;
00187                 return 1;
00188             }
00189             while (binaryFileIn.peek() != EOF) {
00190                 RowData rd(binaryFileIn);
00191                 treeData.addData(rd);
00192             }
00193             binaryFileIn.close();
00194             cout « "Data loaded successfully." « endl;
00195         }
00196         break;
00197
00198         case 10:
00200             cout « "Exiting..." « endl;
00201             return 0;
00202
00203         default:
00204             cout « "Invalid choice. Please try again." « endl;
00205             break;
00206         }
00207     }
00208
00209     return 0;
00210 }
```

4.11 Dokumentacja pliku P6/RowData.cpp

Implementacja klasy RowData do obs^{3}ugi danych wierszy z pliku CSV.

```
#include "RowData.h"
#include "LogManager.h"
#include <algorithm>
#include <iostream>
#include <sstream>
```

4.11.1 Opis szczegÃ^{3}Åowy

Implementacja klasy RowData do obs^{3}ugi danych wierszy z pliku CSV.

Definicja w pliku RowData.cpp.

4.12 RowData.cpp

IdÅº do dokumentacji tego pliku.

```
00001
00003
00004 #include "RowData.h"
00005 #include "LogManager.h"
00006 #include <algorithm>
00007 #include <iostream>
00008 #include <sstream>
00009
00010 using namespace std;
00011
00016 RowData::RowData(const string& line) {
00017     vector<string> values;
00018     stringstream ss(line);
00019     string value;
00020
00021     // Przetwarzanie ka¿dej wartoci z wiersza CSV, usuwanie cudzys$^3$ow i dodawanie do wektora.
00022     while (getline(ss, value, ',')) {
00023         value.erase(remove(value.begin(), value.end(), '\"'), value.end());
00024         values.push_back(value);
```

```
00025     }
00026
00027     // Inicjalizacja pól obiektu na podstawie wczytanych wartoci.
00028     this->date = values[0];
00029     this->selfConsumption = stof(values[1]);
00030     this->exportValue = stof(values[2]);
00031     this->importValue = stof(values[3]);
00032     this->consumption = stof(values[4]);
00033     this->production = stof(values[5]);
00034
00035     globalLogger.log("Wczytano linie: " + this->toString());
00036 }
00037
00041 RowData::RowData(ifstream& in) {
00042     loadFromBinary(in);
00043 }
00044
00047 void RowData::display() const {
00048     cout « date « " " « selfConsumption « " " « exportValue « " " « importValue « " " « consumption «
      " " « production « endl;
00049 }
00050
00053 void RowData::displayData() const {
00054     cout « "\t\t\t\t" « selfConsumption « " " « exportValue « " " « importValue « " " « consumption «
      " " « production « endl;
00055 }
00056
00061 string RowData::toString() {
00062     return date + " " + to_string(selfConsumption) + " " + to_string(exportValue) + " " +
      to_string(importValue) + " " +
00063         to_string(consumption) + " " + to_string(production);
00064 }
00065
00069 void RowData::saveToBinary(ofstream& out) const {
00070     size_t dateSize = date.size();
00071     out.write(reinterpret_cast<const char*>(&dateSize), sizeof(dateSize));
00072     out.write(date.c_str(), dateSize);
00073     out.write(reinterpret_cast<const char*>(&selfConsumption), sizeof(selfConsumption));
00074     out.write(reinterpret_cast<const char*>(&exportValue), sizeof(exportValue));
00075     out.write(reinterpret_cast<const char*>(&importValue), sizeof(importValue));
00076     out.write(reinterpret_cast<const char*>(&consumption), sizeof(consumption));
00077     out.write(reinterpret_cast<const char*>(&production), sizeof(production));
00078 }
00079
00083 void RowData::loadFromBinary(ifstream& in) {
00084     size_t dateSize;
00085     in.read(reinterpret_cast<char*>(&dateSize), sizeof(dateSize));
00086     date.resize(dateSize);
00087     in.read(&date[0], dateSize);
00088     in.read(reinterpret_cast<char*>(&selfConsumption), sizeof(selfConsumption));
00089     in.read(reinterpret_cast<char*>(&exportValue), sizeof(exportValue));
00090     in.read(reinterpret_cast<char*>(&importValue), sizeof(importValue));
00091     in.read(reinterpret_cast<char*>(&consumption), sizeof(consumption));
00092     in.read(reinterpret_cast<char*>(&production), sizeof(production));
00093 }
```

4.13 Dokumentacja pliku P6/RowData.h

Deklaracja klasy RowData do przechowywania i przetwarzania danych z pliku CSV.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
```

Komponenty

- class RowData

  Klasa reprezentuj¹ca dane jednego wiersza z pliku CSV, zawieraj¹ca ró¿ne parametry energetyczne.

4.13.1 Opis szczegÃ³Åowy

Deklaracja klasy RowData do przechowywania i przetwarzania danych z pliku CSV.

Definicja w pliku RowData.h.

4.14 RowData.h

Idź do dokumentacji tego pliku.

```
00001
00003
00004 #ifndef ROWDATA_H
00005 #define ROWDATA_H
00006
00007 #include <iostream>
00008 #include <fstream>
00009 #include <string>
00010 #include <sstream>
00011 #include <vector>
00012
00013 using namespace std;
00014
00017 class RowData {
00018 public:
00022     RowData(const string& line);
00023
00027     RowData(ifstream& in);
00028
00031     void display() const;
00032
00035     void displayData() const;
00036
00040     string toString();
00041
00045     void saveToBinary(ofstream& out) const;
00046
00050     void loadFromBinary(ifstream& in);
00051
00055     string getDate() const { return date; }
00056
00060     float getSelfConsumption() const { return selfConsumption; }
00061
00065     float getExport() const { return exportValue; }
00066
00070     float getImport() const { return importValue; }
00071
00075     float getConsumption() const { return consumption; }
00076
00080     float getProduction() const { return production; }
00081
00082 private:
00083     string date;
00084     float selfConsumption;
00085     float exportValue;
00086     float importValue;
00087     float consumption;
00088     float production;
00089 };
00090
00091 #endif // ROWDATA_H
```

4.15 Dokumentacja pliku P6/TreeData.cpp

Implementacja klasy TreeData do przechowywania i analizy danych w strukturze drzewa.

```
#include "TreeData.h"
#include <iostream>
#include <sstream>
#include <iomanip>
#include <ctime>
```

4.15.1 Opis szczegółowy

Implementacja klasy TreeData do przechowywania i analizy danych w strukturze drzewa.

Definicja w pliku TreeData.cpp.

4.16 TreeData.cpp

IdÅº do dokumentacji tego pliku.

```
00001
00003
00004 #include "TreeData.h"
00005 #include <iostream>
00006 #include <sstream>
00007 #include <iomanip>
00008 #include <ctime>
00009
00010 using namespace std;
00011
00017 void TreeData::addData(const RowData& rowData) {
00018     // Tworzenie strumienia z dat¹ z obiektu RowData
00019     stringstream ss(rowData.getDate());
00020     string token;
00021     vector<int> dateParts;
00022
00023     // Rozdzielenie daty na poszczególne czêci (dzieñ, miesi¹c, rok)
00024     while (getline(ss, token, '.')) {
00025         dateParts.push_back(stoi(token));
00026     }
00027
00028     // Zmienne dla roku, miesi¹ca, dnia, godziny, minuty oraz kwarta³u
00029     int year = dateParts[2];
00030     int month = dateParts[1];
00031     int day = dateParts[0];
00032     int hour = stoi(rowData.getDate().substr(11, 2));
00033     int minute = stoi(rowData.getDate().substr(14, 2));
00034     int quarter = (hour * 60 + minute) / 360;
00035
00036     // Przypisanie wartoci do struktury drzewa na podstawie wyodrêbnionych danych
00037     years[year].year = year;
00038     years[year].months[month].month = month;
00039     years[year].months[month].days[day].day = day;
00040     years[year].months[month].days[day].quarters[quarter].quarter = quarter;
00041     years[year].months[month].days[day].quarters[quarter].hour = hour;
00042     years[year].months[month].days[day].quarters[quarter].minute = minute;
00043     years[year].months[month].days[day].quarters[quarter].data.push_back(rowData);
00044 }
00045
00049 void TreeData::print() const {
00050     // Iterowanie po wszystkich latach w drzewie
00051     for (const auto& yearPair : years) {
00052         const YearNode& yearNode = yearPair.second;
00053         cout « "Year: " « yearNode.year « endl;
00054
00055         // Iterowanie po wszystkich miesi¹cach w danym roku
00056         for (const auto& monthPair : yearNode.months) {
00057             const MonthNode& monthNode = monthPair.second;
00058             cout « "\tMonth: " « monthNode.month « endl;
00059
00060             // Iterowanie po dniach w danym miesi¹cu
00061             for (const auto& dayPair : monthNode.days) {
00062                 const DayNode& dayNode = dayPair.second;
00063                 cout « "\t\tDay: " « dayNode.day « endl;
00064
00065                 // Iterowanie po kwarta³ach w danym dniu
00066                 for (const auto& quarterPair : dayNode.quarters) {
00067                     const QuarterNode& quarterNode = quarterPair.second;
00068                     cout « "\t\t\tQuarter: " « quarterNode.quarter
00069                         « " (Hour: " « quarterNode.hour « ", Minute: " « quarterNode.minute « ")" «
      endl;
00070
00071                     // Wypisanie danych przypisanych do tego kwarta³u
00072                     for (const auto& rowData : quarterNode.data) {
00073                         rowData.displayData();
00074                     }
00075                 }
00076             }
00077         }
```

```
00078     }
00079 }
00080
00087 std::vector<RowData> TreeData::getDataBetweenDates(const std::string& startDate, const std::string&
      endDate) const {
00088     std::vector<RowData> result;
00089
00090     // Konwersja daty pocz¹tkowej na strukturê tm
00091     std::tm tm = {};
00092     std::istringstream ss(startDate);
00093     ss » std::get_time(&tm, "%d.%m.%Y %H:%M");
00094     time_t start = mktime(&tm);
00095
00096     // Konwersja daty koñcowej na strukturê tm
00097     ss.clear();
00098     ss.str(endDate);
00099     ss » std::get_time(&tm, "%d.%m.%Y %H:%M");
00100     time_t end = mktime(&tm);
00101
00102     // Iterowanie po wszystkich latach w drzewie
00103     for (const auto& yearPair : years) {
00104         const YearNode& yearNode = yearPair.second;
00105         for (const auto& monthPair : yearNode.months) {
00106             const MonthNode& monthNode = monthPair.second;
00107             for (const auto& dayPair : monthNode.days) {
00108                 const DayNode& dayNode = dayPair.second;
00109                 for (const auto& quarterPair : dayNode.quarters) {
00110                     const QuarterNode& quarterNode = quarterPair.second;
00111                     for (const auto& rowData : quarterNode.data) {
00112                         std::tm tm = {};
00113                         std::istringstream ss(rowData.getDate());
00114                         ss » std::get_time(&tm, "%d.%m.%Y %H:%M");
00115                         time_t dataTime = mktime(&tm);
00116
00117                         // Porównanie czasu danych z przedzia³em czasowym
00118                         if (dataTime >= start && dataTime <= end) {
00119                             result.push_back(rowData);
00120                         }
00121                     }
00122                 }
00123             }
00124         }
00125     }
00126
00127     return result;
00128 }
00129
00140 void TreeData::calculateSumsBetweenDates(const std::string& startDate, const std::string& endDate,
00141     float& selfConsumptionSum, float& exportSum, float& importSum,
00142     float& consumptionSum, float& productionSum) const {
00143     // Inicjalizacja sum na 0
00144     selfConsumptionSum = 0.0f;
00145     exportSum = 0.0f;
00146     importSum = 0.0f;
00147     consumptionSum = 0.0f;
00148     productionSum = 0.0f;
00149
00150     // Pobranie danych w podanym przedziale czasowym
00151     std::vector<RowData> data = getDataBetweenDates(startDate, endDate);
00152     for (const auto& rowData : data) {
00153         selfConsumptionSum += rowData.getSelfConsumption();
00154         exportSum += rowData.getExport();
00155         importSum += rowData.getImport();
00156         consumptionSum += rowData.getConsumption();
00157         productionSum += rowData.getProduction();
00158     }
00159 }
00160
00171 void TreeData::calculateAveragesBetweenDates(const std::string& startDate, const std::string& endDate,
00172     float& selfConsumptionAvg, float& exportAvg, float& importAvg,
00173     float& consumptionAvg, float& productionAvg) const {
00174     // Inicjalizacja zmiennych sumuj¹cych oraz liczby danych
00175     float selfConsumptionSum = 0.0f, exportSum = 0.0f, importSum = 0.0f, consumptionSum = 0.0f,
      productionSum = 0.0f;
00176     int count = 0;
00177
00178     // Pobranie danych w podanym przedziale czasowym
00179     std::vector<RowData> data = getDataBetweenDates(startDate, endDate);
00180     for (const auto& rowData : data) {
00181         selfConsumptionSum += rowData.getSelfConsumption();
00182         exportSum += rowData.getExport();
00183         importSum += rowData.getImport();
00184         consumptionSum += rowData.getConsumption();
00185         productionSum += rowData.getProduction();
00186         count++;
00187     }
00188
```

```
00189     // Obliczanie rednich, jeli dane istniej¹
00190     if (count > 0) {
00191         selfConsumptionAvg = selfConsumptionSum / count;
00192         exportAvg = exportSum / count;
00193         importAvg = importSum / count;
00194         consumptionAvg = consumptionSum / count;
00195         productionAvg = productionSum / count;
00196     }
00197 }
```

4.17 Dokumentacja pliku P6/TreeData.h

Deklaracja klasy TreeData do przechowywania i analizy danych w strukturze drzewa.

```
#include <map>
#include <string>
#include <vector>
#include "RowData.h"
```

Komponenty

- class TreeData

  < Za³ ¹czenie pliku nag³ ówkowego zawieraj¹cego klasê RowData.
- struct TreeData::QuarterNode

  Reprezentuje dane z podzia³em na kwarta³y dnia. Struktura ta zawiera informacje o godzinie, minucie oraz dane przypisane do danego kwarta³u.
- struct TreeData::DayNode

  Reprezentuje dane dzienne. Struktura ta zawiera informacje o danym dniu oraz mapê kwartalnych danych w tym dniu.
- struct TreeData::MonthNode

  Reprezentuje dane miesiêczne. Struktura ta zawiera informacje o danym miesi¹cu oraz mapê dziennych danych w tym miesi¹cu.
- struct TreeData::YearNode

  Reprezentuje dane roczne. Struktura ta zawiera informacje o roku oraz mapê miesiêcznych danych w tym roku.

4.17.1 Opis szczegÃ³Åowy

Deklaracja klasy TreeData do przechowywania i analizy danych w strukturze drzewa.

Definicja w pliku TreeData.h.

4.18 TreeData.h

Idź do dokumentacji tego pliku.

```
00001
00003
00004 #ifndef TREEDATA_H
00005 #define TREEDATA_H
00006
00007 #include <map>
00008 #include <string>
00009 #include <vector>
00010 #include "RowData.h"
00011
00016 class TreeData {
00017 public:
00021     struct QuarterNode {
00022         int quarter;
00023         int hour;
00024         int minute;
00025         std::vector<RowData> data;
00026     };
00027
00031     struct DayNode {
00032         int day;
00033         std::map<int, QuarterNode> quarters;
00034     };
00035
00039     struct MonthNode {
00040         int month;
00041         std::map<int, DayNode> days;
00042     };
00043
00047     struct YearNode {
00048         int year;
00049         std::map<int, MonthNode> months;
00050     };
00051
00056     void addData(const RowData& rowData);
00057
00061     void print() const;
00062
00068     std::vector<RowData> getDataBetweenDates(const std::string& startDate, const std::string& endDate)
      const;
00069
00080     void calculateSumsBetweenDates(const std::string& startDate, const std::string& endDate,
00081         float& selfConsumptionSum, float& exportSum, float& importSum,
00082         float& consumptionSum, float& productionSum) const;
00083
00094     void calculateAveragesBetweenDates(const std::string& startDate, const std::string& endDate,
00095         float& selfConsumptionAvg, float& exportAvg, float& importAvg,
00096         float& consumptionAvg, float& productionAvg) const;
00097
00110     void compareDataBetweenDates(const std::string& startDate1, const std::string& endDate1,
00111         const std::string& startDate2, const std::string& endDate2,
00112         float& selfConsumptionDiff, float& exportDiff, float& importDiff,
00113         float& consumptionDiff, float& productionDiff) const;
00114
00123     std::vector<RowData> searchRecordsWithTolerance(const std::string& startDate, const std::string&
      endDate,
00124         float value, float tolerance) const;
00125
00126 private:
00127     std::map<int, YearNode> years;
00128 };
00129
00130 #endif // TREEDATA_H
```

Skorowidz

~LogManager
    LogManager, 7

addData
    TreeData, 19

calculateAveragesBetweenDates
    TreeData, 19
calculateSumsBetweenDates
    TreeData, 20
compareDataBetweenDates
    TreeData, 21
consumption
    RowData, 16

data
    TreeData::QuarterNode, 10
date
    RowData, 16
day
    TreeData::DayNode, 5
days
    TreeData::MonthNode, 9
display
    RowData, 13
displayData
    RowData, 13
displayMenu
    main.cpp, 37

errorLogCount
    LogManager.cpp, 33
    LogManager.h, 35
errorLogger
    LogManager.cpp, 33
    LogManager.h, 35
exportValue
    RowData, 17

getConsumption
    RowData, 13
getDataBetweenDates
    TreeData, 22
getDate
    RowData, 13
getExport
    RowData, 14
getImport
    RowData, 14
getProduction
    RowData, 14
getSelfConsumption
    RowData, 14
globalLogger
    LogManager.cpp, 33
    LogManager.h, 35
GoogleTest/test.cpp, 27, 30

hour
    TreeData::QuarterNode, 10

importValue
    RowData, 17

lineValidation
    LineValidation.h, 31
LineValidation.h
    lineValidation, 31
loadFromBinary
    RowData, 15
log
    LogManager, 7
logFile
    LogManager, 8
LogManager, 6
    ~LogManager, 7
    log, 7
    logFile, 8
    LogManager, 6
LogManager.cpp
    errorLogCount, 33
    errorLogger, 33
    globalLogger, 33
LogManager.h
    errorLogCount, 35
    errorLogger, 35
    globalLogger, 35

main
    main.cpp, 37
main.cpp
    displayMenu, 37
    main, 37
minute
    TreeData::QuarterNode, 10
month
    TreeData::MonthNode, 9
months
    TreeData::YearNode, 25

P6/LineValidation.h, 31, 32

P6/LogManager.cpp, 32, 34
P6/LogManager.h, 34, 36
P6/main.cpp, 36, 39
P6/RowData.cpp, 41
P6/RowData.h, 42, 43
P6/TreeData.cpp, 43, 44
P6/TreeData.h, 46, 47
print
TreeData, 23
production
RowData, 17

quarter
TreeData::QuarterNode, 10
quarters
TreeData::DayNode, 5

RowData, 11
consumption, 16
date, 16
display, 13
displayData, 13
exportValue, 17
getConsumption, 13
getDate, 13
getExport, 14
getImport, 14
getProduction, 14
getSelfConsumption, 14
importValue, 17
loadFromBinary, 15
production, 17
RowData, 12
saveToBinary, 15
selfConsumption, 17
toString, 16

saveToBinary
RowData, 15
searchRecordsWithTolerance
TreeData, 23
selfConsumption
RowData, 17

TEST
test.cpp, 28, 29
test.cpp
TEST, 28, 29
toString
RowData, 16
TreeData, 17
addData, 19
calculateAveragesBetweenDates, 19
calculateSumsBetweenDates, 20
compareDataBetweenDates, 21
getDataBetweenDates, 22
print, 23
searchRecordsWithTolerance, 23
years, 24
TreeData::DayNode, 5
day, 5
quarters, 5
TreeData::MonthNode, 8
days, 9
month, 9
TreeData::QuarterNode, 9
data, 10
hour, 10
minute, 10
quarter, 10
TreeData::YearNode, 24
months, 25
year, 25

year
TreeData::YearNode, 25
years
TreeData, 24