Katowice, 22.06.2018 r.

Laboratorium Programowania Komputerów

Temat:
Organizer Studencki

Autor: Mirosław Ściebura
studia stacjonarne inżynierskie, kierunek Informatyka,
rok II, semestr IV, grupa nr 3
Prowadzący: dr hab. inż. Roman Starosolski
Link: https://github.com/polsl-aei-pk4/Miroslaw-Sciebura-gr32-repo/tree/master/ProjektQt/Projekt-OrganizerStudencki

1. Temat

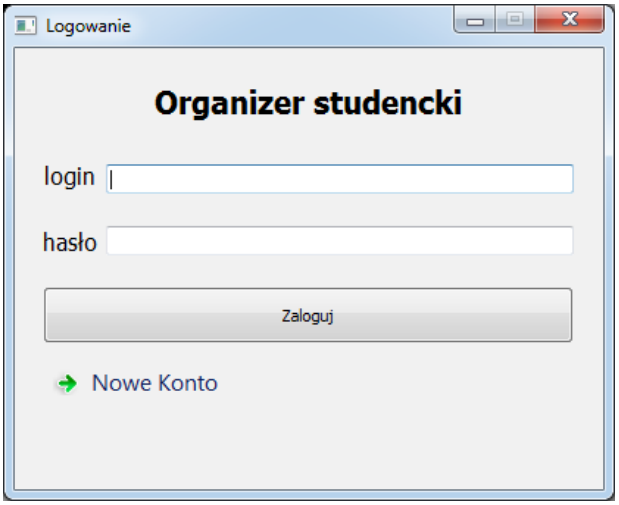
Tematem było stworzenie aplikacji „Organizer studencki”, która jest formą kalendarza dostosowaną do potrzeb studenta.

2. Analiza tematu

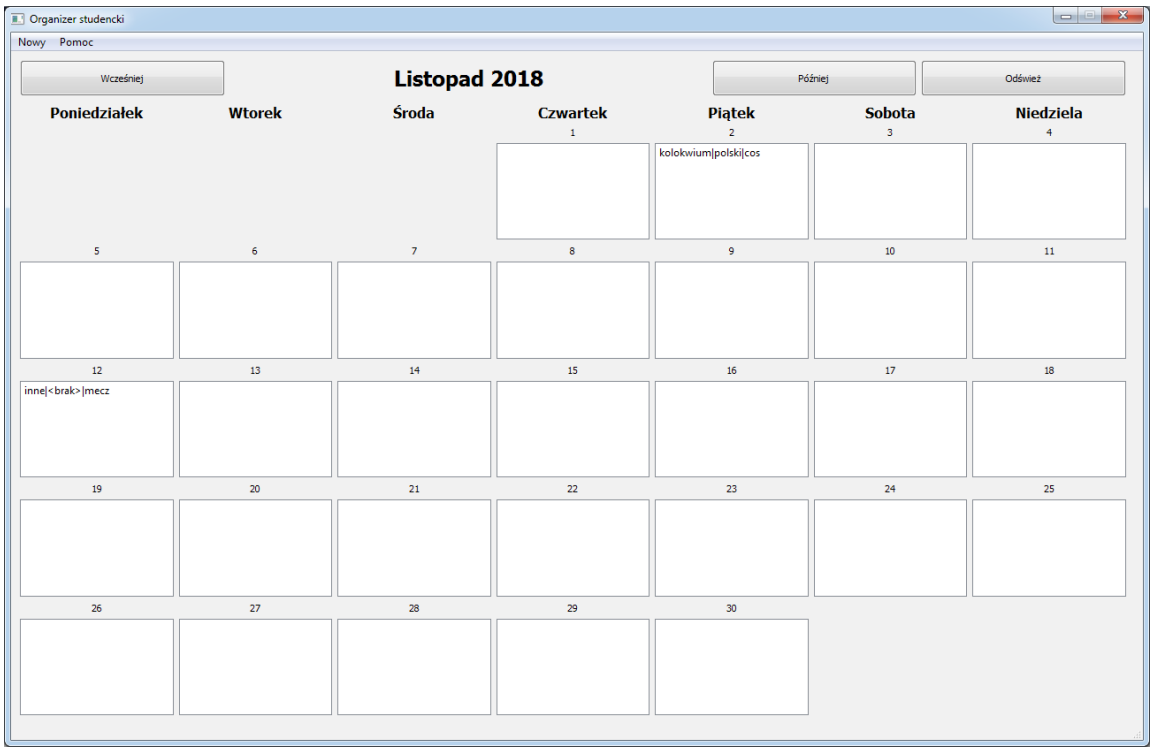
Aplikacja jest pewną formą kalendarza do którego można dodawać takie zdarzenia jak zajęcia lub wydarzenia (z uwzględnieniem przedmiotu jakiego dotyczą). Jako, że chciałem aby aplikacja miała graficzny interfejs użytkownika, użyłem środowiska Qt do utworzenia odpowiednich okienek. Jako, że program bazuje na operacjach związanych z czasem, używałem również biblioteki ctime, która wydała mi się najbardziej przejrzysta.

3. Specyfikacja zewnętrzna

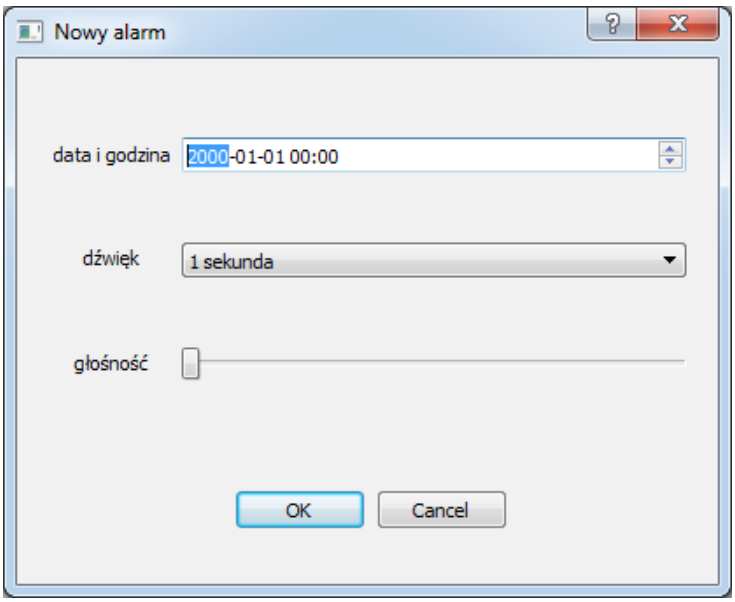
Po włączeniu ukazuje się okno logowania. Gdy mamy utworzone konto do programu wchodzimy przyciskiem „Zaloguj”. Możemy też utworzyć nowego użytkownika przyciskiem „Nowe konto”.



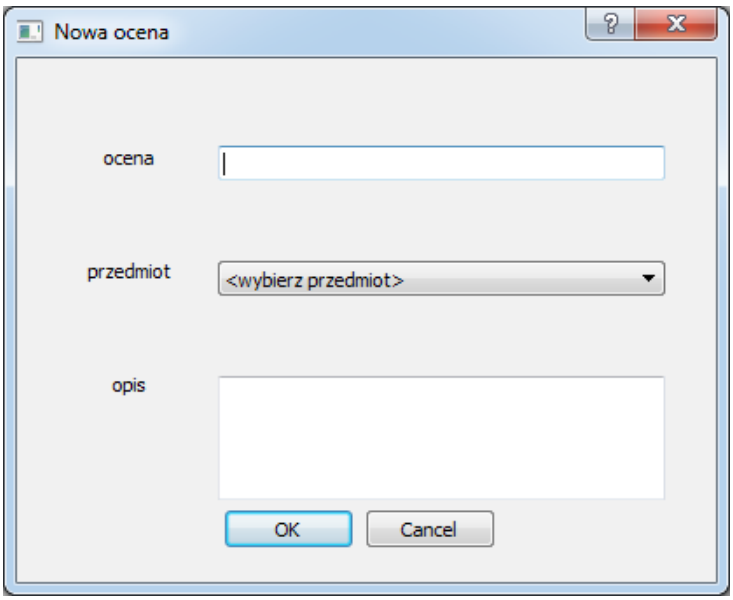
Po zalogowaniu pokazuje się główne okno programu. Możemy przechodzić między miesiącami przyciskami „Wcześniej” i „Później”. Mamy także przycisk „Odśwież”, który odpowiada za pokazywanie komunikatów (naciśnięcie odpowiada za wyświetlenie aktualnie oczekujących na pokazanie komunikatów – nie trzeba naciskać, gdyż sygnał tego przycisku jest emitowany co sekundę). Jak można zauważyć w odpowiednich polach kalendarza wpisywane są odpowiednie alarmy, wydarzenia i zajęcia. Widzimy też pasek menu z którego można wybierać elementy do dodania do kalendarza lub je usuwać. Można też wyświetlać oceny.



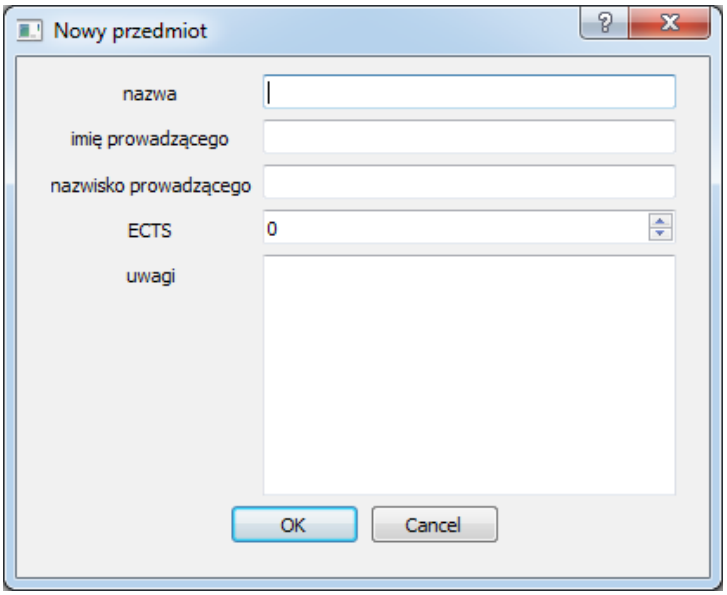
Pierwszym z okien do wyboru odpowiedniego elementu jest okno alarmu. Pozwala ono na dodanie alarmu o odpowiedniej godzinie (nie może być wcześniejsza niż aktualna). Należy też wybrać długość trwania i głośność (częstotliwość)



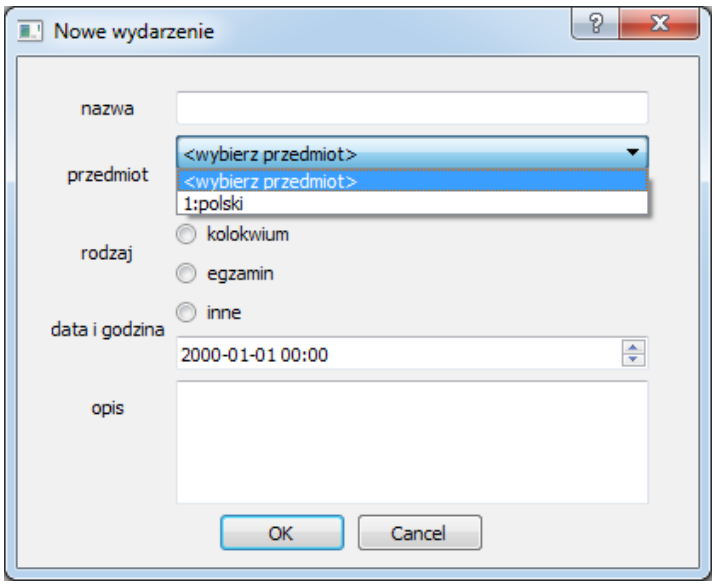
Następnym jest okno oceny, w którym możemy dodać ocenę z konkretnego przedmiotu i jej opis (wartość rzeczywista z zakresu 2 do 5).



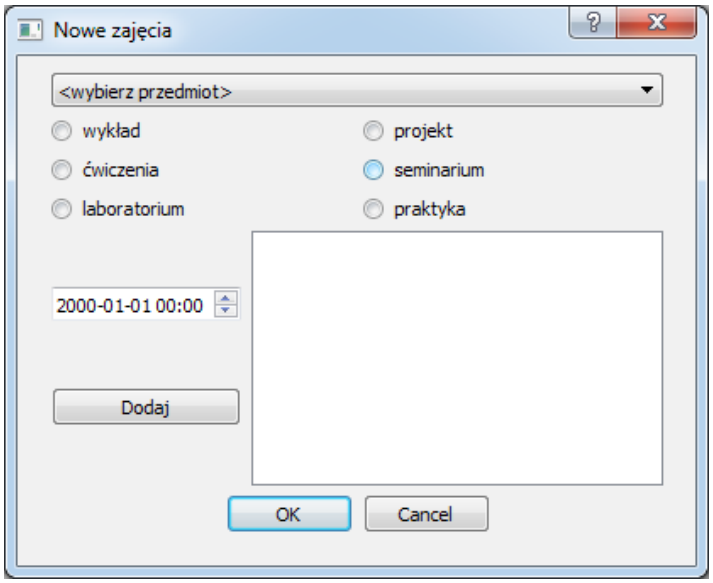
Kolejne okno to okno przedmiotu, w którym możemy dodać przedmiot (jego nazwę, imię i nazwisko prowadzącego, większą od 0 i mniejszą od 30 liczbę ECTS oraz ewentualne uwagi). Musimy wypełnić 3 pierwsze pola.



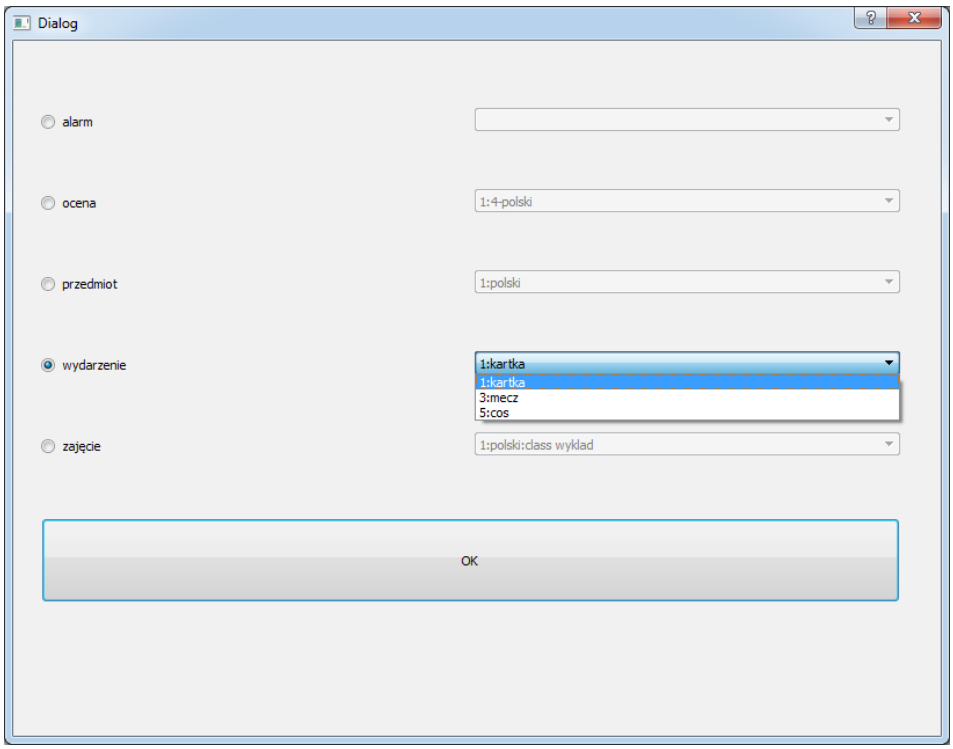
Mamy też okno dodawania wydarzenia. Musimy wpisać jego nazwę, wybrać przedmiot (chyba, że jest to wydarzenie typu inne) oraz jego datę i godzinę. Ewentualnie możemy też dopisać jego opis.



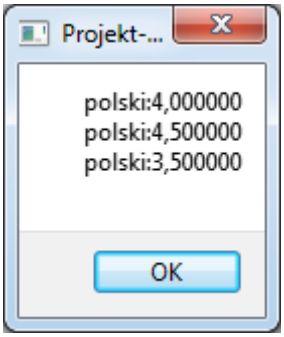
Ostatnim z okien dodawania jest okno dodawania zajęć. Dodajemy zajęcia z konkretnego przedmiotu i konkretnego typu. Na dole wybieramy datę i przyciskiem „Dodaj” dodajemy do listy dat.



Możemy też usuwać za pomocą okna usuwania, znajdującego się w podmenu „Pomoc”. Wybieramy co chcemy usunąć i wybieramy z listy odpowiedni element i zatwierdzamy przyciskiem „OK”.



Możemy też wyświetlić posiadane przez użytkownika oceny przy pomocy okna ocen. Wyświetla ono nazwę przedmiotu i wartość konkretnej oceny.



4. Specyfikacja wewnętrzna

Użyte klasy to:

4.1 OknoLogowania

Klasa odpowiadająca za początkowe okno programu

Pola:

Ui::OknoLogowania *ui – wskaźnik do elementów interfejsu graficznego,

std::shared_ptr<std::vector<profil>> profile – wskaźnik na wektor zobiektami klasy profil,

std::unique_ptr<QMessageBox> okienko – wskaźnik na okienko typu MessageBox (wskaźnik unikalny),

int * nrprofilu – adres do zmiennej w której będziemy przechowywać aktualny numer profilu

Metody:

explicit OknoLogowania(QWidget *parent = 0) – konstruktor domyślny utworzony przez środowisko Qt,
explicit OknoLogowania(int*) własny konstruktor, przyjmujący adres zmiennej którą będziemy zmieniać w celu zasygnalizowania aktualnego profilu
~OknoLogowania() - destruktor domyślny Qt, usuwający ui.

Specjalną kategorią są sloty, które obsługują akcje takie jak naciśnięcie przycisku:

private slots:

void on_przycisklogowania_clicked() - obsługuje naciśnięcie przycisku logowania (sprawdza wektor profili w poszukiwaniu danego loginu i porównania hasła, w przypadku błędu wyświetla MessageBox),

void on_przycisknowegokonta_clicked() - obsługuje naciśnięcie przycisku nowego konta (sprawdzenie wektora profili w celu znalezienia loginu, w przypadku błędu wyświetla MessageBox)

4.2. OknoGlowne

Główna klasa programu w większości kontrolująca działanie programu i przechowująca kontenery z odopwiednimi obiektami.

Pola:

Ui::OknoGlowne *ui – wskaźnik do wszystkich elementów GUI,
std::array<std::string, 12> miesiace – tablicza z nazwami miesięcy utworzonymi przy pomocy biblioteki ctime,
bool przel – zmienna do przełączania pętli w wątku nadającym sygnały klawisza odśwież,
int aktmiesiac, int aktrok – zmienne określające aktualny miesiąc w kalendarzu,
int nrprofilu – zmienna przechowująca nr profilu (potrzebna do tworzenia nazw plików),
int numeratorprzedm, int numeratorwyd, int numeratorzaj, int numeratorocen – zmienne nadające klucze w strukturze map do przechowywania danych obiektów,
std::unique_ptr<oknoalarmu> nowyalarm,
std::unique_ptr<oknooceny> nowaocena,
std::unique_ptr<oknoprzedmiotu> nowyprzedmiot,
std::unique_ptr<oknowydarzenia> nowewydarzenie,
std::unique_ptr<oknozajec> nowezajecie,
std::unique_ptr<oknousuwania> noweusuwanie – wskaźniki do okien nowych elementów i usuwania (Singletony),
std::unique_ptr<QMessageBox> okienko - wskaźnik na okienko typu MessageBox do wyświetlania komunikatów powiadamiających (Singleton),
std::shared_ptr<std::vector<alarm>> wektoralarmow – kontener przechowujący alarmy dodane w programie lub czytane z pliku, wskaźnik ze względu na przekazywanie do okien w których się go uzupełnia, wektor gdyż zakładam, że liczba sekund (time_t) będzie czymś na kształt klucza,
std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> mapaprzedmiotow,
std::shared_ptr<std::map<int,std::shared_ptr<zajecia>>> mapazajec,
std::shared_ptr<std::map<int,std::shared_ptr<wydarzenie>>> mapawydarzen,

std::shared_ptr<std::map<int,std::shared_ptr<ocena>>> mapaocen – wskaźniki do kontenera map z odpowiednimi obiektami (coś na kształt bazy danych),

std::shared_ptr<std::list<std::pair<std::string, std::string>>> listazdarzen – wskaźnik na listę ze oczekującymi zdarzniami (jest opróżniana co sekundę),

std::string nazwaplikualarmow,

std::string nazwaplikuwydarzen,

std::string nazwaplikuzaj,

std::string nazwaplikuocen – nazwy plików w których przechowuje się odpowiednie informacje dla odpowowiednich klas

Metody:

void wykonajkalendarz() - metoda tworząca tablicę z nazwami miesięcy,

void widok() - metoda odświeżająca widok danego miesiąca i wpisująca odpowiednie zdarzenia do pól,

void wyswietlanie() - inna metoda pomocnicza do wyświetlania alarmów,

void sprawdzanie(bool &przel) – metoda wykonywana przez wątek obsługujący sygnał odświeżania listy powiadomień,

void wczytaniealarmow(), void wczytanieprzedmiotow(), void wczytaniewydarzen(),

void wczytaniezajec(), void wczytanieocen() - metody wczytujące dane odpowiednich obiektów z pliku *.txt

Metody slotów:

void on_wczesniej_clicked(), void on_pozniej_clicked() - metody związane z przyciskami „Później” oraz „Wcześniej”, odpowiadają za odpowiednie wyświetlenie kalendarza i elementów w nim,

void on_actionAlarm_triggered(), void on_actionOcena_triggered(),

void on_actionPrzedmiot_triggered(), void on_actionZako_cz_triggered(),

void on_actionZaj_cia_triggered(), void on_actionWydarzenie_triggered(),

void on_actionO_programie_triggered(), void on_actionOceny_triggered() - sloty związane z przyciskami w menu górnym (on_actionO_programie_triggered związane z usuwaniem) – wywołują konstruktory odpowiednich okien i przekazują do nich inteligentne wskaźniki na kontenery,

void on_pushButton_clicked() - metoda slotu związana z przyciskiem odświeżania, tworzy odpowiednie MessageBoxy dla zdarzeń na liście i ją opróżnia

4.3. Klasa oknoalarmu

Klasa odpowiedzialna za okno tworzenia nowego alarmu. Obsługuje dodawanie alarmów do wektora z obsługą sytuacji wyjątkowych.

Pola:

Ui::oknoalarmu *ui wskaźnik do elementów GUI, utworzony przez środowisko,

std::shared_ptr<std::vector<alarm>> alarmy – wskaźnik na wektor z obiektami typu alarm przechowywanymi w trakcie działania programu,

std::unique_ptr<QMessageBox> okienko – unikalny wskaźnik na okienko MessageBox do wyświetlania komunikatów

std::string nazwapliku – nazwa pliku do którego będziemy zapisywać

Metody:

explicit oknoalarmu(QWidget *parent = 0) – konstruktor domyślny, utworzony przez środowisko,

explicit oknoalarmu(std::shared_ptr<std::vector<alarm>>, int nrusera, std::string _nazwapliku) – konstruktor okna przyjmujący wektor z alarmami

~oknoalarmu() - destruktor okna, niszczy całe ui związane z oknem,

Metody slotów:

void on_przyciski_accepted() - metoda wywoływana przy naciśnięciu „OK”, tworzy nowy alarm i zapisuje go do pliku lub obsługuje wyjątek (MessageBox),

void on_przyciski_rejected() - zamyka okno przy „Cancel”

void on_glosnosc_sliderReleased() - metoda pomocna przy tworzeniu aplikacji, wyświetla na ostream wartość slidera

4.4. oknooceny

Okno dialogowe do dodawania nowych ocen. Obsługuje dodawanie nowych ocen z uwzględnieniem sytuacji wyjątkowych.

Pola:

Ui::oknooceny *ui – wskaźnik do elementów GUI, utworzony przez środowisko

std::unique_ptr<QMessageBox> okienko – wskaźnik do okienka MessageBox (singleton),

std::shared_ptr<std::map<int,std::shared_ptr<ocena>>> mapaocen – wskaźnik na kontener z ocenami (uzupełninie podczas dodawania, mapa ze względu na postać bazy danych, wskaźniki ze względu na ewentualne wykorzystanie gdzie indziej),

std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> mapaprzedmiotow – wskaźnik na mapę przedmiotów,

int* numerator – wskaźnik do zmiennej przechowującej numer oceny nadawanej kolejnej dodawanej,

std::string nazwapliku – string z nazwą pliku do którego zapisujemy informacje

Metody:

explicit oknooceny(QWidget *parent = 0) – domyślny konstruktor, utworzony przez środowisko Qt

explicit oknooceny(std::shared_ptr<std::map<int,std::shared_ptr<ocena>>>,
std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>>,
int* , std::string) – konstruktor przyjmujący wskaźniki do poszczegolnych, potrzebnych kontenerów oraz wskaźnik do zmienianego numeratora oraz nazwę pliku,

Metody slotów:

void on_buttonBox_accepted() - metoda wywoływana przy naciśnięciu przycisku „OK”, wpisuje nową ocenę do map (z obsługą wyjątków)

void on_buttonBox_rejected() - metoda zamykająca okno przy naciśnięciu przycisku „Cancel”

void on_comboBox_currentIndexChanged(int index) – metoda pomocnicza do wypisywania na ostream aktualnego indeksu menu rozwijanego combobox,

void on_checkBox_toggled(bool checked) – metoda dodatkowa na wypadek umieszczenia przycisku checkbox do uruchamiania combobox,

4.5. oknoprzedmiotu

Klasa odpowiadająca za wyświetlanie okna dodawania przedmiotu, Dodaje przedmiot do mapy z obsługą sytuacji wyjątkowych.

Pola:

Ui::oknoprzedmiotu *ui – wskaźnik do GUI utworzonego przez środowisko Qt,
std::unique_ptr<QMessageBox> okienko – wskaźnik na okno typu MessageBox (singleton)
std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> mapaprzedmiotow -wskaźnik na map ze wskaźnikami do przedmiotów (coś na kształt bazy danych),
int* numerator - wskaźnik do zmiennej przechowującej odpowiedni numer nadawany w zmiennej map

Metody:

explicit oknoprzedmiotu(QWidget *parent = 0)
explicit oknoprzedmiotu(std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> _mapaprzedmiotow, int*) - własny konstruktor przyjmujący wskaźnik do mapy ze wskaźnikami na przedmioty i wskaźnik do zmiennej która odpowiada za numerowanie,
~oknoprzedmiotu() - destruktor, niszczy ui

Metody slotów:

void on_buttonBox_accepted() - metoda wywoływana po naciśnięciu przycisku „OK”, dodaje do pliku i do map nowy przedmiot (z obsługą sytuacji wyjątkowych),
void on_buttonBox_rejected() - metoda zamykająca okno w przypadku naciśnięcia przycisku „Cancel”

4.5. oknousuwania

Klasa odpowiadająca za wyświetlane okna usuwania, usuwa element żądany przez użytkownika.

Pola:

Ui::oknousuwania *ui – wskaźnik do ui utworzonego przez środowisko Qt,
std::shared_ptr<std::vector<alarm>> wektoralarmow,
std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> mapaprzedmiotow,
std::shared_ptr<std::map<int,std::shared_ptr<zajecia>>> mapazajec,
std::shared_ptr<std::map<int,std::shared_ptr<wydarzenie>>> mapawydarzen,
std::shared_ptr<std::map<int,std::shared_ptr<ocena>>> mapaocen – wskaźniki do kontenerów, z których można usuwać elementy
std::string nazwaplikualarmow, std::string nazwaplikuwydarzen,
std::string nazwaplikuprzed, std::string nazwaplikuzaj, std::string nazwaplikuocen – stringi przechowujące nazwy plików z których będziemy usuwali

Metody:

explicit oknousuwania(QWidget *parent = 0) – konstruktor domyślny, utworzony przez środowisko Qt,

explicit oknousuwania(const std::shared_ptr<std::vector<alarm>> &,
const std::shared_ptr<std::map<int,std::shared_ptr<ocena>>> &,
const std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> &,
const std::shared_ptr<std::map<int,std::shared_ptr<wydarzenie>>>& wydarzenia,
const std::shared_ptr<std::map<int,std::shared_ptr<zajecia>>> & zajecia,
std::string, std::string, std::string, std::string, std::string) – konstruktor w którym przekazujemy wskaźniki na wszystkie kontenery które można zmieniać oraz stringi z nazwami plików

~oknousuwania() - destruktor, usuwa ui

Metody slotów:

void on_radioButton_toggled(bool checked),
void on_radioButton_2_toggled(bool checked),
void on_radioButton_3_toggled(bool checked),
void on_radioButton_4_toggled(bool checked),
void on_radioButton_5_toggled(bool checked) – metody włączające odpowiednie menu rozwijane w zależności od ustawionego radio buttona,
void on_pushButton_clicked() - metoda usuwająca z odpowiedniego kontenera i zapisująca wszystko do pliku

4.6. oknowydarzenia

Klasa odpowiedzialna za obsługę okna dodawania wydarzeń, dodaje odpowiednie wydarzenie do map i do pliku.

Pola:

Ui::oknowydarzenia *ui – wskaźnik do ui utworzonego przez środowisko Qt,
std::unique_ptr<QMessageBox> okienko – wskaźnik na okienko typu MessageBox do wyświetlania komunikatów o błędzie (singleton),
std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> mapaprzedmiotow – wskaźnik na map z wskaźnikami obiekty typu przedmiot, potrzebne aby zrobić odniesienie do wydarzenia,
std::shared_ptr<std::map<int,std::shared_ptr<wydarzenie>>> mapawydarzen – wskaźnik na map z obiektami typu wydarznie (struktura prostej bazy danych)
std::string nazwapliku – nazwa pliku do którego będzie prowadzony zapis,
int* numerator – wskaźnik do zminnej przechowującej aktualny najwyższy numer wydarzenia

Metody:

explicit oknowydarzenia(QWidget *parent = 0) – konstruktor domyślny utworzony przez środowisko,
explicit oknowydarzenia(std::shared_ptr<std::map<int,std::shared_ptr<wydarzenie>>> _mapawydarzen, std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> _mapaprzedmiotow, std::string, int*) - konstruktor przekazujący wskaźnik na map z wskaźnikami na wydarzenia i map z wskaźnikami na przedmioty oraz nazwę pliku jako string i wskaźnik do zmiennej przechowującej aktualny najwyższy numer wydarzenia
~oknowydarzenia() - destruktor, który usuwa ui

Metody slotów:

void on_buttonBox_rejected() - metoda zamykająca okno przy naciśnięciu przycisku „Cancel”,

void on_buttonBox_accepted() - metoda dodająca wydarzenie do map wydarzeń i zapisująca do pliku (z obsługą sytuacji wyjątkowych),

void on_radioButton_4_toggled(bool checked) – metoda odpowiedzialna za wyłączenie wyboru przedmiotów przy zaznaczeniu kaegorii „inne”

4.7. oknozajec

Klasa do obsługi okna zajeć. Obsługuje dodawanie zajęć do kontenera w odpowiednich datach z obsługą sytuacji wyjątkowych.

Pola:

Ui::oknozajec *ui – wskaźnik do ui utworzonego przez Qt,

std::vector<time_t> daty – pomocniczy wektor do wpisywania dat,

std::unique_ptr<QMessageBox> okienko -wskaźnik na okno typu MessageBox(singleton),

std::shared_ptr<std::map<int,std::shared_ptr<zajecia>>> mapazajec – wskaźnik do map przechowującego wskaźniki do zajęć (prosta baza danych),

std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> mapaprzedmiotow – wskaźnik na map z wskaźnikami do przedmiotów (potrzebne do przypisywania zajęcium przedmiotów,

int *numerator – wskaźnik na zmienną która przechowuje ostatni numer wydarzenia,

std::string nazwapliku – przechowuje nazwę pliku do którego prowadzony jest zapisuje

Metody:

explicit oknozajec(QWidget *parent = 0) – konstruktor domyślny utworzony przez środowisko Qt,

explicit oknozajec(std::shared_ptr<std::map<int,std::shared_ptr<zajecia>>> _mapazajec, std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> _mapaprzedmiotow, int*, std::string) – konstruktor otrzymujący wskaźniki do kontenerów z zajęciami (uzupełnianie) i przedmiotami (przypisywanie do zajęć) oraz wskaźnik do zmiennej gdzie przechowywany jest numer największego ID zajęć oraz nazwa pliku do którego będzie prowadzony zapis.

~oknozajec() - destruktor, usuwa ui okna

Metody slotów:

void on_buttonBox_accepted() - metoda wywaoływana przy naciśnięciu przycisku „Ok”, tworzy obiekt zajęcia (odpowiedniego typu) i przepisuje do niego daty,

void on_buttonBox_rejected() - metoda zamykająca gdy naciśniemy „Cancel”,

void on_pushButton_clicked() - metoda dodająca do listy i wectora nową datę

4.8. alarm

Klasa będąca odpowiednikiem alarmu (powiadomienia dźwiękowego o danej godzinie).

Pola:

time_t datawyd – ilość sekund od początku epoki (1.01.1970 r.) określająca datę i godzinę wydarzenia,

int dźwięk – określenie który rodzaj dźwięku ma być odtwarzany (1,3 czy 5 sekund),

int glosnosc – określenie głośności (częstotliwości dźwięku wydawanego przez komputer)

Metody:

alarm(time_t, int, int) – konstruktor tworzący obiekt na podstawie 3 wartości, wyrzuca wyjątki gdy data jest przeszła, typ alarmu jest różny od 0, 1 lub 2 oraz głośność nie jest z przedziału 0-99,
time_t dajdate() - getter daty,
int dajdzwiek() - getter rodzaju dźwięku,
int dajglosnosc() - getter głośności,
bool sprawdzmoment(const time_t &) - metoda do porównywania czasu (czy jest równy z podanym)
~alarm() - destruktor

4.9. ocena

Klasa reprezentująca ocenę akademicką z danego przedmiotu.

Pola:

double wartosc – wartość danej oceny (od 2.0 do 5.0)
przedmiot* przedm – wskaźnik do przedmiotu (jedynie adres, nic nie alokujemy),
std::string opis - - opis oceny

Metody:

ocena(double, przedmiot*) - konstruktor przyjmujący wartość oceny i wskaźnik do przedmiotu (wyrzuca wyjątek w przypadku oceny spoza przedziału 2.0 – 5.0 lub nullptr dla przedmiotu),
ocena(double, przedmiot*, std::string) – konstruktor który dodatkowo przyjmuje string z opisem,
double dajwartosc() - getter wartości oceny,
przedmiot* dajprzedm() - getter przedmiotu,
std::string dajopis() - getter opisu,
~ocena() - destruktor

4.10. profil

Reprezentuje profil użytkownika danego kalendarza.

Pola:
std::string login – login danego użytkownika,
int ID – numer ID danego użytkownika nadawany przy tworzeniu nowego konta,
std::string haslo – string z hasłem danego użytkownika (będzie szyfrowane),
static int przydzial – zmienna statyczna do nadawania ID, przechowuje numer przy użyciu którego będziemy wpisywać do pola nowotworzonego obiektu

Metody:

profil(std::string, std::string) – konstruktor przyjmujący stringi z loginem i hasłem, rzuca wyjątki gdy mają długości równe 0,
static std::string zaszyfruj(const std::string &) - statyczna metoda (wykorzystuje tą samą daną szyfrowania),

bool sprawdzlogin(const std::string &) - metoda porównująca login z podanym,
bool sprawdzhaslo(const std::string &) - metoda porównująca hasło z podanym,
int dajID() - getter numeru ID,
std::string dajlogin() - getter loginu,
std::string dajhaslo() - getter hasła,
void static wpiszprzyd(int) – metoda statyczna do wpisywania przydziału do obiektu,
~profil() - destruktor

4.11. przedmiot

Klasa reprezentująca przedmiot akademicki.

Pola:

std::string nazwa – nazwa przedmiotu,
std::string nazwisko – nazwisko prowadzącego,
std::string imię – imię prowadzącego,
int ECTS – liczba ECTS przedmiotu,
std::string uwagi – string z uwagami do przedmiotu

Metody:

przedmiot(std::string, std::string, std::string, int) – konstruktor, przyjmujący nazwę oraz dane prowadzącego i liczbę ECTS (rzuca wyjątki w przypadku ciągów o długości 0),
przedmiot(std::string, std::string, std::string, int, std::string) – konstruktor, który dodatkowo przyjmuje string z opisem,
std::string dajnazwe() - getter nazwy,
std::string dajprowadzacego() - getter danych prowadzącego,
std::string dajopis() - getter opisu;
int dajECTS() - getter liczby ECTS,
~przedmiot() - destruktor

4.11. wydarzenie

Klasa reprezentująca wydarzenie takie jak egzamin, kartkówka, kolokwium lub inne, niezwiązane z żadnym przedmiotem (są to klasy pochodne).

Pola:

time_t data – data (liczba sekund od 1970 r.) w którym nastąpi wydarzenie,
std::string nazwa – nazwa wydarzenia,
std::string opis – opis wydarzenia

w klasach pochodnych egzamin, kartkówka, kolokwium dodatkowo:

przedmiot* przedm – adres przedmiotu do którego odnosi się wydarzenie (jedynie adres, nic nie alokujemy)

Metody:

wydarzenie(time_t, std::string) – konstruktor przyjmujący datę i nazwę (wyjątek gdy nazwa jest pusta lub data przeszła),

wydarzenie(time_t, std::string, std::string) – konstruktor który dodatkowo przyjmuje opis,
time_t dajdate() - getter daty,
std::string dajnazwe() - getter nazwy,
std::string dajopis() - getter opisu,
virtual przedmiot* dajprzedm() - getter przedmiotu (wirtualny ze względu na dziedziczenie)
virtual ~wydarzenie() - wirtualny (dziedziczenie) destruktor

4.12. zajecia

Klasa reprezentująca serię zajęć danego typu z konkretnego przedmiotu. Dziedziczą po niej klasy wyklad, cwiczenia, laboratorium, projekt, seminarium i praktyka.

Pola:

przedmiot* przedm – wskaźnik do przedmiotu, z którego są zajęcia (jedynie odnośnik, nie alokujemy),
std::vector<time_t> daty - wektor dat kiedy zajęcia się odbędą

Metody:

zajecia(przedmiot*) - konstruktor przyjmujący adres przedmiotu (rzuca wyjątek gdy nullptr),
void wpiszdaty(std::vector<time_t> _daty) – setter wektora dat,
std::vector<time_t> dajdaty() - getter wektora dat,
przedmiot* dajprzedm() - getter przedmiotu,
virtual ~zajecia() - wirtualny (dziedziczenie0 destruktor

Struktury danych

Najczęściej wykorzystywaną strukturą danych był kontener map. Pomagał on w odwzorowaniu struktury prostej bazy danych dla wszystkich elementów. Przechowywane były tam najczęściej inteligentne wskaźniki do obiektów. Korzystałem również z kontenerów vector w prostszych zastosowaniach oraz list dla listy zdarzeń wyświetlanych w określonym momencie przez program.

Zastosowane techniki obiektowe

1. Wątek

Zastosowałem wątek wywoływany co sekundę do przeszukiwania kontenerów z obiektami programu (alarmy, wydarzenia, zajęcia) aby dodawać informacje o aktualnych do listy zdarzeń. W oknie głównym musiałem wykorzystać drugi wątek do emitowania co sekundę sygnału przycisku „Odśwież”, gdyż ze względu na specyfikę aplikacji okienkowej nie byłem wstanie wykonywać co sekundę jakieś metody (pętla okna uwzględnia jedynie metody do obsługi np. przycisków).

```
void    odczytczasu(bool&    _przelacznik,    std::mutex&    _blokada,    time_t&    _czas,    const
std::shared_ptr<std::vector<alarm>> & _alarmy,
                const std::shared_ptr<std::map<int,std::shared_ptr<wydarzenie>>> & _wydarzenia,
                const std::shared_ptr<std::map<int,std::shared_ptr<zajecia>>> & _kontenerzajecia,
                const std::shared_ptr<std::list<std::pair<std::string, std::string>>> &
_listazdarzen)
{
    while (_przelacznik)
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
```

```
        _blokada.lock();
        _czas = time(NULL);
        //std::cout << ctime(&_czas) << std::endl;
        std::vector<alarm>::iterator italarm;
        for (italarm = _alarmy->begin(); italarm != _alarmy->end(); italarm++)
            if (_czas == (*italarm).dajdate())
            {
                std::string a = std::to_string((*italarm).dajdzwiek());
                a += " ";
                a += std::to_string((*italarm).dajglosnosc());
               _listazdarzen->push_front(std::pair<std::string,
std::string>(typeid(*italarm).name(), a));
            }
        std::map<int,std::shared_ptr<wydarzenie>>::iterator itwyd;
        for (itwyd = _wydarzenia->begin(); itwyd != _wydarzenia->end(); itwyd++)
        {
            std::shared_ptr<wydarzenie> lok = std::get<1>(*itwyd);
            if (lok->dajdate() == _czas)
            {
                _listazdarzen->push_front(std::pair<std::string,
std::string>(typeid(*(lok.get())).name(), lok->dajnazwe()));
            }
        }
        std::map<int,std::shared_ptr<zajecia>>::iterator itzaj;
        for (itzaj = _kontenerzajecia->begin(); itzaj != _kontenerzajecia->end(); itzaj++)
        {
            std::shared_ptr<zajecia> lok = std::get<1>(*itzaj);
            std::vector<time_t> pom = lok->dajdaty();
            for (int i = 0; i < pom.size(); i++)
                if (pom[i] == _czas)
                {
                    _listazdarzen->push_front(std::pair<std::string,
std::string>(typeid(*(lok.get())).name(), lok->dajprzedm()->dajnazwe()));
                }
        }
        _blokada.unlock();
    }
}


//obslugaczasu - watek
    bool a1 = true;
    std::mutex blokada;
    time_t czas;
    std::thread czassystemowy(odczytczasu, std::ref(a1), std::ref(blokada), std::ref(czas),
std::ref(alarmy)
                              , std::ref(wydarzenia), std::ref(kontenerzajecia),
std::ref(listazdarzen));
    czassystemowy.detach();
    //obsługa czasu end
```

2. Kontenery STL

Do przechowywania obiektów używałem głównie kontenera map (imitacja bazy danych), a także vector (dla alarmów i dat w oknie zajęć), array (miesiące) oraz list (lista zdarzeń).

```
        std::shared_ptr<std::vector<alarm>> wektoralarmow;
        std::shared_ptr<std::map<int,std::shared_ptr<przedmiot>>> mapaprzedmiotow;
        std::shared_ptr<std::map<int,std::shared_ptr<zajecia>>> mapazajec;
        std::shared_ptr<std::map<int,std::shared_ptr<wydarzenie>>> mapawydarzen;
        std::shared_ptr<std::map<int,std::shared_ptr<ocena>>> mapaocen;
        std::shared_ptr<std::list<std::pair<std::string, std::string>>> listazdarzen;
        std::array<std::string, 12> miesiace;
```

3. Iteratory STL

Do nawigowania po strukturze map, aby wyświetlić wszystkie elementy nie wystarczy wyszukiwanie po kluczu i trzeba było wykorzystać iterator.

```
std::map<int,std::shared_ptr<wydarzenie>>::iterator itwyd;
        for (itwyd = _wydarzenia->begin(); itwyd != _wydarzenia->end(); itwyd++)
        {
            std::shared_ptr<wydarzenie> lok = std::get<1>(*itwyd);
```

```
            if (lok->dajdate() == _czas)
            {
                _listazdarzen->push_front(std::pair<std::string,
std::string>(typeid(*(lok.get())).name(), lok->dajnazwe()));
            }
        }
```

4. RTTI

Potrzebne było głównie do rozróżniania konkretnego typu zajęcia lub wydarzenia przy wskaźniku.

```
 plikwej << std::get<0>((*mapazajec->rbegin())) << " ";
            std::shared_ptr<zajecia> lokalny = std::get<1>((*mapazajec->rbegin()));
            std::string klasa = typeid(*lokalny.get()).name();
            klasa.erase(0, 6);
            plikwej << klasa << " ";
```

5. Wyjątki

Potrzebne do sygnalizacji nie właściwych danych wejściowych konstruktora.

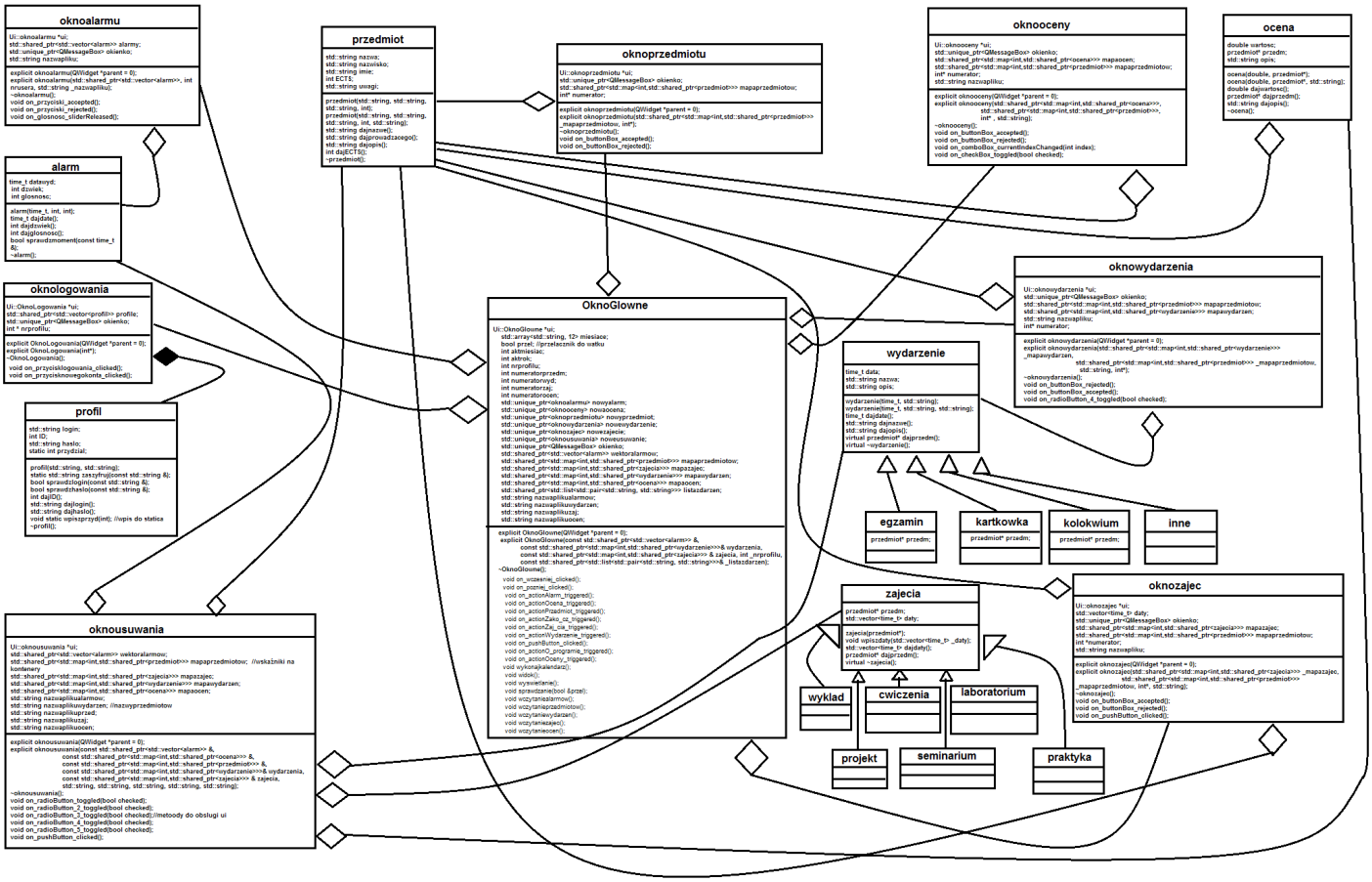
```
alarm::alarm(time_t ldata, int ldzwiek, int lglosnosc):dzwiek(ldzwiek), glosnosc(lglosnosc)
{
    time_t aktualnyczas;//=time(NULL); - też da radę
    time(&aktualnyczas);
    if (difftime(ldata, aktualnyczas) < 0)
        throw std::logic_error("data"); //sprawdzanie daty czy nie jest przeszła
    if (ldzwiek != 0 && ldzwiek != 1 && ldzwiek != 2)
        throw std::exception(); //odp rodzaj dzwieku i czestotliwosc
    if (lglosnosc < 0 || lglosnosc > 100)
        throw std::exception();
    datawyd = ldata;
}
```

6. Inteligentne wskaźniki

Wykorzystywane dość często w celu uniknięcia wycieków pamięci i łatwiejszego przekazywania obiektów między klasami.

```
oknoalarmu::oknoalarmu(std::shared_ptr<std::vector<alarm>>   _alarmy,   int   nrusera,   std::string
_nazwapliku)
```

Diagram klas



5. Uruchamianie i testowanie

Testowanie odbywało się na zasadzie kompilowania programu po serii mniejszych zmian. Najczęściej notowanymi błędami były złe odwołania do wskaźników (nullptr lub niezaalokowane), błędy zapisu do plików oraz błędy związane z złymi wartościami liczb. Również próba wybrania elementu o nieistniejącym kluczu w kontenerze map była dość częsta.

6. Wnioski

Realizacja projektu pozwoliła w dość dobrze poprawić zdolności programistyczne oraz poznać nowsze elementy języka C++. Stworzenie graficznego interfejsu użytkownika pozwoliło na znacznie łatwiejsze korzystanie z aplikacji niż wersja tekstowa. Umiejętność tworzenia takiego interfejsu w środowisku Qt jest również cechą pożądaną na rynku pracy, zatem uważam że realizacja projektu była bardzo potrzebna i przydatna.