

Politechnika Świętokrzyska w Kielcach

Wydział Elektrotechniki, Automatyki i Informatyki
Katedra Informatyki, Elektroniki i Elektrotechniki

Przedmiot:

Programowanie w języku C 2

Tytuł:

eStaurant

Program do obsługi zamówień dla lokali gastronomicznych

Autor:

Bartosz Bukowski

Grupa:

2ID11B

Studia:

Stacjonarne I stopnia

Kierunek:

Informatyka

Spis treści

1. Wstęp	1
2. Specyfikacja projektowa.....	2
3. Techniczna implementacja.....	
3.1. Niezbędne pliki	3
3.2. Uruchomienie programu i ekran powitalny	4
3.3. Okno główne aplikacji.....	5
3.4. Okno dialogowe „Klienci”	13
3.5. Okno dialogowe „Menu”	17
3.6. Okno dialogowe „Archiwum”	19
3.7. Okno dialogowe „eStaurant – informacje”	21
4. Testowanie projektu.....	23
5. Wnioski	25
6. Lista załączników projektowych	26
7. Instrukcja uruchomienia	
7.1. Wymagania systemowe	28
7.2. Instrukcja kompilacji projektu	28
7.3. Instrukcja obsługi programu	32
8. Oświadczenie o samodzielności wykonania projektu	35

1. Wstęp

Celem zadania projektowego było stworzenie programu pozwalającego na obsługiwanie zamówień w lokalu gastronomicznym. Program musiał być prosty oraz intuicyjny w obsłudze oraz oferować użytkownikowi wachlarz niezbędnych do realizacji powierzonych mu zadań narzędzi. W tym celu program wyposażono w następujące funkcje:

- Dodawanie nowego zamówienia (dane klienta, zamówiony posiłek, sposób płatności)
- Wyświetlenie listy aktualnych zamówień
- Usuwanie zamówienia (archiwizacja)
- Wyświetlenie archiwum zamówień
- Czyszczenie archiwum zamówień
- Wyświetlenie menu dostępnych potraw
- Dodawanie nowej potrawy do menu (nazwa, opis, gramatura, cena)
- Modyfikacja danych dotyczących wybranej potrawy
- Usuwanie potrawy z menu
- Dodawanie do listy klientów na podstawie złożonych przez nich zamówień
- Modyfikacja danych dotyczących wybranego klienta
- Usuwanie klientów z listy

Program stworzono w języku C++ w oparciu o bibliotekę wxWidgets 3.0.5.

2. Specyfikacja projektowa

Program został stworzony w języku C++ przy pomocy środowiska Code::Blocks 20.03. Do kompilacji użyto wbudowanego kompilatora GNU GCC Compiler, bazując na 64-bitowym systemie operacyjnym Windows 10 (wersja 1909). Do stworzenia graficznego interfejsu użytkownika wykorzystano wieloplatformową bibliotekę klas C++ wxWidgets w najnowszej, stabilnej wersji 3.0.5 (stan na 10.01.2021).

Wybór języka C++ podyktowany był głównie przez jego obiektowość, jak i dostępność bibliotek przystosowanych do tworzenia aplikacji z graficznym interfejsem użytkownika. Przy tworzeniu programu, bezcenne okazały się takie cechy tego języka jak dziedziczenie, metody wirtualne, konstruktury i destruktory czy przeciążanie funkcji i operatorów.

Niebywałą zaletą programowania przy pomocy biblioteki wxWidgets w środowisku Code::Blocks jest dostępność pluginu wxSmith, znacząco ułatwiającego pracę nad GUI w sposób tzw. WYSIWYG (ang. what you see is what you get). Możliwe dzięki temu jest wyświetlanie podczas pracy (jeszcze przed kompilacją) rzeczywistych jej efektów – interfejs programu widoczny jest w takiej formie, w jakiej widoczny będzie dla użytkownika. Ponadto wxSmith oferuje szybkie tworzenie oraz dodawanie do aplikacji wszelkich kontrolek, np. przycisków wyboru (ang. check box), pól wyboru (ang. combo box), przycisków (ang. buttons), a także gotowych, bądź spersonalizowanych pod własne potrzeby okien dialogowych.

Działanie podstawowych funkcji programu opiera się na obsłudze plików tekstowych z rozszerzeniem TXT, zlokalizowanych w folderze *eStaurant/data*. Zdecydowanym plusem takiego rozwiązania jest niewielkie zapotrzebowanie na pamięć masową komputera, a także niskie wykorzystanie pamięci RAM. Minusem może okazać się zbyt niska wydajność takiego systemu w przypadku dużych baz danych. Program obsługuje polskie znaki diakrytyczne, dzięki zastosowaniu kodowania UTF-8. Wszelkie grafiki (logo programu, ekran powitalny) pobierane są z przygotowanych na potrzeby projektu plików BMP, zawartych w katalogu *eStaurant/resources*.

3. Techniczna implementacja

3.1. Niezbędne pliki

Plik wykonywalny *eStaurant.exe* znajduje się w głównym katalogu *eStaurant*. Do właściwego jego wykonania wymagane są zawarte w głównym folderze pliki:

- *libgcc_s_seh-1.dll*
- *libstdc++-6.dll*
- *libwinpthread-1.dll*
- *wxbase30u_gcc810_x64.dll*
- *wxmsw30u_adv_gcc810_x64.dll*
- *wxmsw30u_core_gcc810_x64.dll*
- *libwxmsw30u_adv.a*

Są to niezbędne pliki biblioteki wxWidgets.

Ponadto w katalogu głównym powinny znajdować się dwa foldery:

- *data* – zawierający 4 pliki tekstowe TXT*:
 - *archives_list.txt* – zawiera dane archiwum zamówień
 - *customers_list.txt* – zawiera dane listy klientów
 - *dishes_list.txt* – zawiera dane listy posiłków
 - *orders_list.txt* – zawiera ostatnią listę zamówień
- *resources* – zawierający 5 plików graficznych:
 - *icon.ico* – ikona programu wyświetlana podczas jego działania na pasku zadań, oraz w lewym, górnym rogu na pasku tytułowym
 - *icon2.ico* – ikona pliku wykonywalnego *eStaurant.exe*
 - *logo.bmp* – większe logo programu, wyświetlane w oknie dialogowym „eStaurant – informacje”
 - *logo_small.bmp* – mniejsze logo programu, wyświetlane w lewym, dolnym rogu głównego okna aplikacji
 - *splash.bmp* – ekran powitalny, wyświetlany po uruchomieniu programu, zanim główne jego okno zostanie w pełni utworzone i wyświetlone

Wszystkie wymienione wyżej pliki są konieczne do uruchomienia, oraz poprawnego działania programu.

* w przypadku braku któregoś z wymienionych plików, należy utworzyć w w/w katalogu nowy, pusty plik tekstowy TXT o podanej nazwie.

Program w sumie korzysta z 6 własnych plików nagłówkowych, 30 plików nagłówkowych biblioteki wxWidgets oraz 2 plików nagłówkowych podstawowych bibliotek języka C++:

```
#include "eStaurantApp.h"
#include "eStaurantMain.h"
#include "Customers_Dialog.h"
#include "Menu_Dialog.h"
#include "Archive_Dialog.h"
#include "Info_Dialog.h"

#include <wx/app.h>
#include <wx/frame.h>
#include <wx/dialog.h>
#include <wx/sizer.h>
#include <wx/textfile.h>
#include <wx/wfstream.h>
#include <wx/txtstrm.h>
#include <wx/log.h>
#include <wx/font.h>
#include <wx/intl.h>
#include <wx/string.h>
#include <wx/splash.h>
#include <wx/image.h>
#include <wx/msgdlg.h>
#include <wx/utils.h>
#include <wx/headercol.h>
#include <wx/bitmap.h>
#include <wx/icon.h>
#include <wx/button.h>
#include <wx/listctrl.h>
#include <wx/menu.h>
#include <wx/textdlg.h>
#include <wx/checkbox.h>
#include <wx/choice.h>
#include <wx/combobox.h>
#include <wx/panel.h>
#include <wx/statbmp.h>
#include <wx/stattext.h>
#include <wx/textctrl.h>
#include <wx/valtext.h>

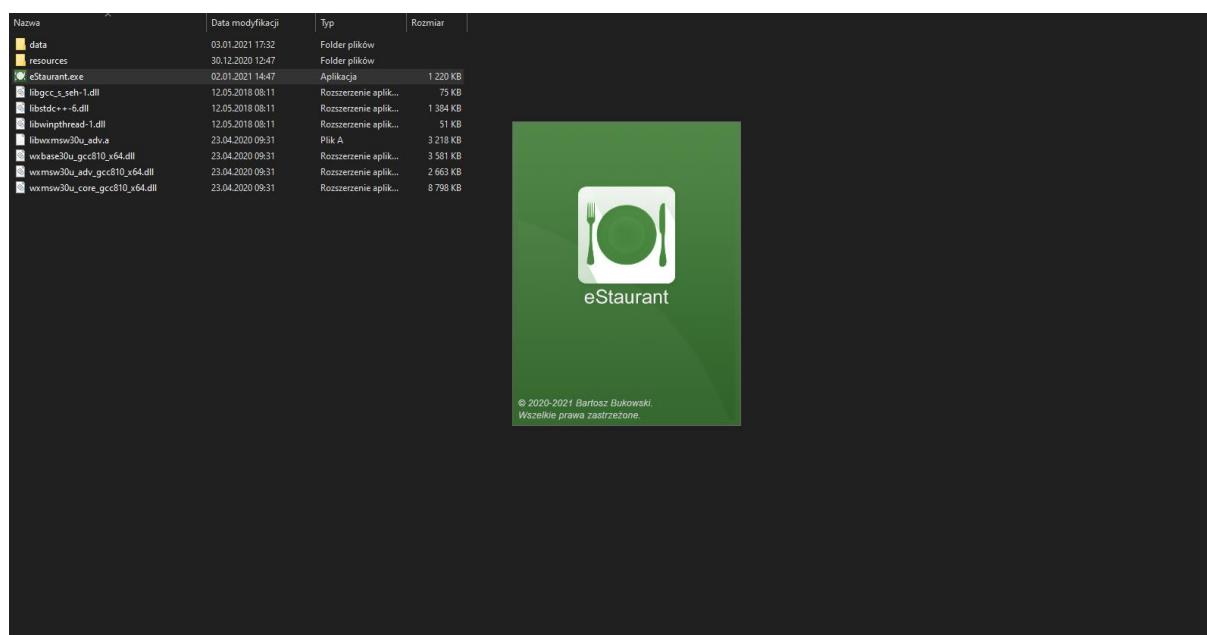
#include <vector>
#include <algorithm>
```

Listing 1 – wszystkie pliki nagłówkowe wykorzystane w programie.

3.2. Uruchomienie programu i ekran powitalny

Kod odpowiedzialny za uruchomienie aplikacji zawarty jest w pliku *eStaurantApp.cpp*, przy czym korzysta on z własnego pliku nagłówkowego *eStaurantApp.h*.

Poprawnemu uruchomieniu programu towarzyszy wyświetlenie w pierwszej kolejności ekranu powitalnego – małego okna o wymiarach 300x400 pikseli, zawierającego logo aplikacji, nazwę „eStaurant” oraz krótką informację o twórcy.



Zrzut 1 – ekran powitalny wyświetlany podczas uruchamiania programu.

Ekran powitalny wyświetlany jest przez 1,5 sekundy. W tym czasie inicjalizowane jest główne okno aplikacji.

Za wykonanie powyższych operacji odpowiada metoda *bool OnInit()*, wywoływana podczas uruchomienia programu. Zwraca ona wartość *wxsOK (true)* w przypadku, gdy cała procedura przebiegła pomyślnie.

```
bool eStaurantApp::OnInit()
{
    bool wxsOK=true;
    wxInitAllImageHandlers();
    if(wxsOK)
    {
        wxBitmap splash_bmp(_T("resources/splash.bmp"), wxBITMAP_TYPE_BMP);
        wxSplashScreen* splashscreen=new wxSplashScreen(splash_bmp,
        wxSPLASH_CENTRE_ON_SCREEN|wxSPLASH_TIMEOUT, 1500, NULL, wxID_ANY);
        splashscreen->Show();
        eStaurantFrame* Frame=new eStaurantFrame(0);
        Frame->Show();
        SetTopWindow(Frame);
    }
    return wxsOK;
}
```

Listing 2 – metoda *bool OnInit()*, wywoływana podczas uruchamiania programu.

3.3. Okno główne aplikacji

Kod głównego okna programu zawarty jest w pliku źródłowym *eStaurantMain.cpp*, natomiast wykorzystywany plik nagłówkowy nosi nazwę *eStaurantMain.h*.

Główne okno aplikacji (*eStaurantFrame*) uruchamiane jest w trybie pełnoekranowym, w rozmiarze dostosowanym do ustawionej rozdzielczości ekranu użytkownika. Taka elastyczność możliwa jest dzięki rozmieszczaniu kontrolki aplikacji w *sizerach*, czyli obiektach przechowujących informacje o wzajemnym położeniu tych kontrolki względem siebie w obrębie danego okna. Przyjmują one układ współrzędnych i rozmiar okna, na

których pracują, a wielkość poszczególnych kontroltek określana jest przy tym na podstawie ich wzajemnej proporcji – określone są one w kodzie poprzez liczby całkowite.

```
Create(parent, wxID_ANY, _T("eStaurant"), wxDefaultPosition, wxDefaultSize,
wxDEFAULT_FRAME_STYLE|wxTAB_TRAVERSAL, _T("wxID_ANY"));
SetBackgroundColour(wxColour(247,247,247));
wxFont
thisFont(10,wxFONTFAMILY_SWISS,wxFONTSTYLE_NORMAL,wxFONTWEIGHT_NORMAL,false,_T("Arial"),wxFON
TENCODING_DEFAULT);
SetFont(thisFont);
wxTopLevelWindow::Maximize(true);
{
    wxIcon FrameIcon;
    FrameIcon.CopyFromBitmap(wxBitmap(wxImage(_T("resources/icon.ico"))));
    SetIcon(FrameIcon);
}
```

Listing 3 – konstruowanie okna głównego oraz jego maksymalizacja, a także ustawienia domyślnego fontu oraz koloru tła.

Główne okno aplikacji podzielone jest na 3 części:

- I część (skrajnie lewa) – zawiera przyciski (*wxButton*) nawigujące: „Klienci”, „Menu” oraz „Archiwum”, po wciśnięciu których wywoływane są odpowiednie metody, odpowiedzialne za wyświetlane okien dialogowych. W dolnej części widoczne jest logo w rozmiarze 48x48 pikseli (*wxStaticBitmap*) oraz nazwa programu, wraz z rokiem kompilacji ostatniej, aktualnej wersji.

```
void eStaurantFrame::OnMenu_ButtonClick(wxCommandEvent& event)
{
    Menu_Dialog MenuDialog(this);
    if(MenuDialog.ShowModal()==wxID_CANCEL)
        return;
    DishComboBoxesClear();
    DishComboBoxesAddItems();
}

void eStaurantFrame::OnCustomers_ButtonClick(wxCommandEvent& event)
{
    Customers_Dialog CustomersDialog(this);
    if(CustomersDialog.ShowModal()==wxID_CANCEL)
        return;
    Customer_ComboBox->Clear();
    CustomerComboBoxAddItems();
}

void eStaurantFrame::OnArchive_ButtonClick(wxCommandEvent& event)
{
    Archive_Dialog ArchiveDialog(this);
    if(ArchiveDialog.ShowModal()==wxID_CANCEL)
        return;
}
```

Listing 4 – metody odpowiedzialne za wyświetlanie okien dialogowych, kolejno: *Menu*, *Klienci*, *Archiwum*.

- II część (środkowa) – lista (*wxListCtrl*) zawierająca informacje o bieżących zamówieniach. Podzielona jest przy tym na 7 kolumn:
 - „Numer” – numer zamówienia, ustalany automatycznie podczas dodawania nowego zamówienia. Jego wartość to numer ostatniego zamówienia + 1
 - „Adres” – adres, na który dostarczone ma zostać zamówienie. Pobierany jest z pola „Adres” podczas tworzenia nowego zamówienia

- „Kwota” – łączna kwota zamówienia. Obliczana jest automatycznie na podstawie wybranych podczas tworzenia zamówienia potraw
- „Płatność” – typ płatności użytej do opłacenia zamówienia przez klienta. Pobierana jest z pola wyboru „Płatność” podczas tworzenia zamówienia
- „Numer tel.” – numer telefonu klienta składającego zamówienie. Pobierany jest z pola „Numer tel.” przy tworzeniu zamówienia
- „Data” – data złożenia zamówienia w formacie GG:MM:SS RRRR/MM/DD (godzina:minuta:sekunda rok/miesiąc/dzień). Pobierana jest automatycznie podczas dodawania nowego zamówienia
- „Szczegóły” – informacje o zamawianych potrawach oraz ich ilości. Pobierane są z pól „Danie nr X” oraz położonych obok nich pól wyboru ilości porcji

```

wxListItem Col_Number;
Col_Number.SetId(0);
Col_Number.SetText(_T("Numer"));
Col_Number.SetWidth(50);
Orders_ListCtrl->InsertColumn(0, Col_Number);

wxListItem Col_Address;
Col_Address.SetId(1);
Col_Address.SetText(_T("Adres"));
Col_Address.SetWidth(120);
Orders_ListCtrl->InsertColumn(1, Col_Address);

wxListItem Col_Price;
Col_Price.SetId(2);
Col_Price.SetText(_T("Kwota"));
Col_Price.SetWidth(85);
Orders_ListCtrl->InsertColumn(2, Col_Price);

wxListItem Col_Payment;
Col_Payment.SetId(3);
Col_Payment.SetText(_T("Płatność"));
Col_Payment.SetWidth(100);
Orders_ListCtrl->InsertColumn(3, Col_Payment);

wxListItem Col_Telephone;
Col_Telephone.SetId(4);
Col_Telephone.SetText(_T("Numer tel.));
Col_Telephone.SetWidth(80);
Orders_ListCtrl->InsertColumn(4, Col_Telephone);

wxListItem Col_Date;
Col_Date.SetId(5);
Col_Date.SetText(_T("Data"));
Col_Date.SetWidth(135);
Orders_ListCtrl->InsertColumn(5, Col_Date);

wxListItem Col_Info;
Col_Info.SetId(6);
Col_Info.SetText(_T("Szczegóły"));
Col_Info.SetWidth(350);
Orders_ListCtrl->InsertColumn(6, Col_Info);

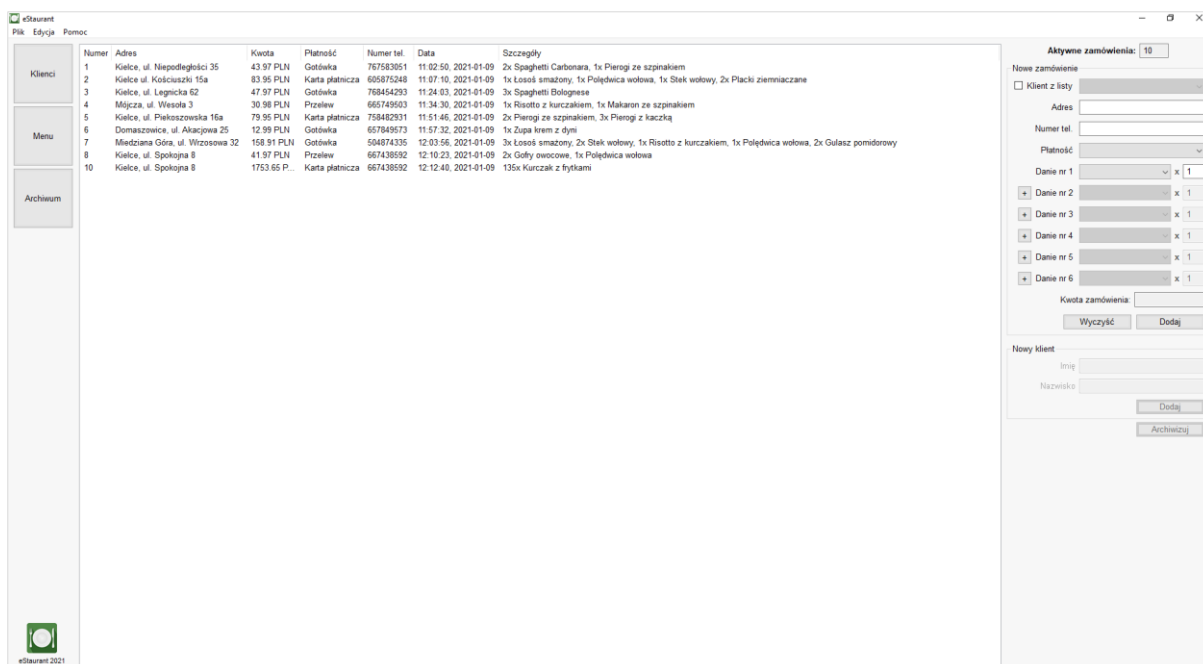
```

Listing 5 – dodawanie oraz konfigurowanie kolumn dla listy zamówień.

Domyślnie, po włączeniu programu lista jest pusta. Od razu można przystąpić do tworzenia jej, poprzez dodawanie nowych zamówień. Możliwe również jest wczytanie ostatniej listy zamówień z pliku *orders_list.txt*. Należy w tym celu wybrać opcję

„Plik” > „Wczytaj ostatnią listę” w górnym pasku menu. Po wczytaniu listy, licznik aktywnych zamówień (patrz punkt niżej) zostanie zaktualizowany (metoda *SetTotalOrdersNumber()*), a także ustawiony zostanie numer dla następnego, nowego zamówienia (metody *FillVector()* oraz *SetOrderNumber()*). Lista zapisywana jest automatycznie do pliku *orders_list.txt* (nadpisuje poprzedni plik) po każdym dodaniu bądź usunięciu zamówienia. Po kliknięciu lewym przyciskiem myszy na daną kolumnę listy, zostanie ona posortowana rosnąco według danych znajdujących się w tej kolumnie.

- III część (skrajnie prawa) – zawiera automatycznie aktualizowany licznik aktywnych zamówień, przycisk „Archiwizuj” służący do usunięcia wybranego z listy zamówienia oraz umieszczeniu go w *Archiwum*, a także dwa panele (*wxPanel*), służące do tworzenia nowych zamówień oraz dodawaniu nowych klientów do listy *Klienci*:
 - Panel „Nowe zamówienie” – umożliwia stworzenie nowego zamówienia. Elementy panelu znajdują się wSizerze *wxStaticBoxSizer*. Zawiera pola wejściowe (*wxTextCtrl*): „Adres”, „Numer tel.” (umożliwia wprowadzenie jedynie znaków numerycznych); pola wyboru (*wxChoice* i *wxComboBox*): „Klient z listy”, umożliwiające automatyczne wypełnienie pól „Adres” oraz „Numer tel.” po wybraniu danego klienta (aby opcja była dostępna, należy najpierw zaznaczyć znajdujący się obok przycisk wyboru (*wxCheckBox*)), „Płatność” (dostępne opcje to *Gotówka*, *Karta płatnicza*, *Przelew* oraz *Inny*), 6 pól wyboru dań (pola od 2 do 6 wymagają wcześniejszego ich włączenia przyciskiem „+” znajdującym się przy każdym z tych pól) wraz z odpowiadającymi im polami wymagającymi wprowadzenia ilości zamawianych porcji (domyślnie 1), a także pole „Kwota zamówienia”, zawierające informację o kwocie tworzonego zamówienia – kwota jest automatycznie aktualizowana po każdej modyfikacji zamówienia (zmianie wybranych potraw lub ich ilości), a ceny poszczególnych potraw pobierane są z pliku *dishes_list.txt*. Do zatwierdzania nowego zamówienia służy przycisk „Dodaj”, a do czyszczenia kontrolki wejścia dla tego panelu – przycisk „Wyczyść” (czyszczenie odbywa się także automatycznie po zatwierdzeniu nowego zamówienia).
 - Panel „Nowy klient” – umożliwia dodanie nowego klienta do listy „Klienci”. Również tutaj, elementy panelu umieszczone są wSizerze *wxStaticBoxSizer*. Panel aktywuje się jedynie w momencie wyboru konkretnego zamówienia z listy aktywnych zamówień. Zawiera pola wejściowe (*wxTextCtrl*): „Imię” oraz „Nazwisko”, natomiast adres oraz numer telefonu, które mają zostać przypisane do klienta, pobierane są automatycznie z wybranego zamówienia. Do zatwierdzenia operacji służy przycisk „Dodaj”.



Zrzut 2 – główne okno aplikacji z przykładową listą zamówień.

```
NewOrderPanel = new wxPanel(this, ID_PANEL1, wxDefaultPosition, wxDefaultSize,
wxTAB_TRAVERSAL, _T("ID_PANEL1"));
NewOrderPanel->SetBackgroundColour(wxColour(247,247,247));
NewOrderSizer = new wxStaticBoxSizer(wxVERTICAL, NewOrderPanel, _T("Nowe zamówienie"));
NewCustomerPanel = new wxPanel(this, ID_PANEL2, wxDefaultPosition, wxDefaultSize,
wxTAB_TRAVERSAL, _T("ID_PANEL2"));
NewCustomerPanel->SetBackgroundColour(wxColour(247,247,247));
NewCustomerSizer = new wxStaticBoxSizer(wxVERTICAL, NewCustomerPanel, _T("Nowy klient"));
```

Listing 6 – konstruowanie paneli oraz sizerów wxStaticBoxSizer.

```
void eStaurantFrame::OnOrderAdd_ButtonClick(wxCommandEvent& event)
{
    long maxIndex=Orders_ListCtrl->GetItemCount();
    wxString OrderNumber;
    OrderNumber << OrderNum;
    wxString Dishes;
    wxString Address=Address_TextCtrl->GetValue();
    wxString Payment=Payment_ChoiceBox->GetString(Payment_ChoiceBox->GetSelection());
    wxString Telephone=Telephone_TextCtrl->GetValue();
    wxDateTime now=wxDateTime::Now();
    wxString day=now.FormatDate();
    wxString hour=now.FormatTime();
    wxString FullDate;
    FullDate << hour << ", " << day;
    wxString Dish1=Dish1_ComboBox->GetString(Dish1_ComboBox->GetSelection());
    if(NumberOfDishes==1)
        Dishes << StrDish1Multiplier << wxT("x ") << Dish1;
    if(NumberOfDishes>1)
    {
        wxString Dish2=Dish2_ComboBox->GetString(Dish2_ComboBox->GetSelection());
        if(NumberOfDishes==2 && Dish1!=wxT("") && Dish2!=wxT(""))
            Dishes << StrDish1Multiplier << wxT("x ") << Dish1 << wxT(", ") <<
                StrDish2Multiplier << wxT("x ") << Dish2;
        if(NumberOfDishes>2)
        {
            wxString Dish3=Dish3_ComboBox->GetString(Dish3_ComboBox->GetSelection());
            if(NumberOfDishes==3 && Dish1!=wxT("") && Dish2!=wxT("") && Dish3!=wxT(""))
                Dishes << StrDish1Multiplier << wxT("x ") << Dish1 << wxT(", ") <<
                    StrDish2Multiplier << wxT("x ") << Dish2 << wxT(", ") <<
                    StrDish3Multiplier << wxT("x ") << Dish3;

            if(NumberOfDishes>3)
```

```

        {
            wxString Dish4=Dish4_ComboBox->GetString(Dish4_ComboBox->GetSelection());
            if(NumberOfDishes==4 && Dish1!=wxT("") && Dish2!=wxT("") && Dish3!=wxT("")
            && Dish4!=wxT(""))
                Dishes << StrDish1Multiplier << wxT("x ") << Dish1 << wxT(", ") <<
                StrDish2Multiplier << wxT("x ") << Dish2 << wxT(", ") <<
                StrDish3Multiplier << wxT("x ") << Dish3 << wxT(", ") <<
                StrDish4Multiplier << wxT("x ") << Dish4;
            if(NumberOfDishes>4)
            {
                wxString Dish5=Dish5_ComboBox->GetString(Dish5_ComboBox->
                >GetSelection());
                if(NumberOfDishes==5 && Dish1!=wxT("") && Dish2!=wxT("") &&
                Dish3!=wxT("") && Dish4!=wxT("") && Dish5!=wxT(""))
                    Dishes << StrDish1Multiplier << wxT("x ") << Dish1 << wxT(", ") <<
                    StrDish2Multiplier << wxT("x ") << Dish2 << wxT(", ") <<
                    StrDish3Multiplier << wxT("x ") << Dish3 << wxT(", ") <<
                    StrDish4Multiplier << wxT("x ") << Dish4 << wxT(", ") <<
                    StrDish5Multiplier << wxT("x ") << Dish5;
                if(NumberOfDishes>5)
                {
                    wxString Dish6=Dish6_ComboBox->GetString(Dish6_ComboBox->
                    >GetSelection());
                    if(NumberOfDishes==6 && Dish1!=wxT("") && Dish2!=wxT("") &&
                    Dish3!=wxT("") && Dish4!=wxT("") && Dish5!=wxT("") &&
                    Dish6!=wxT(""))
                        Dishes << StrDish1Multiplier << wxT("x ") << Dish1 << wxT(", ") <<
                        << StrDish2Multiplier << wxT("x ") << Dish2 << wxT(", ") <<
                        StrDish3Multiplier << wxT("x ") << Dish3 << wxT(", ") <<
                        StrDish4Multiplier << wxT("x ") << Dish4 << wxT(", ") <<
                        StrDish5Multiplier << wxT("x ") << Dish5 << wxT(", ") <<
                        StrDish6Multiplier << wxT("x ") << Dish6;
                    }
                }
            }
        }
    }
    if(!(Address==wxT("") || Telephone==wxT("") || Payment==wxT("") || Dishes==wxT("")))
    {
        long itemIndex=Orders_ListCtrl->InsertItem(maxIndex, OrderNumber);
        Orders_ListCtrl->SetItem(itemIndex, 0, OrderNumber);
        Orders_ListCtrl->SetItem(itemIndex, 1, Address);
        Orders_ListCtrl->SetItem(itemIndex, 2, StrOrderValue);
        Orders_ListCtrl->SetItem(itemIndex, 3, Payment);
        Orders_ListCtrl->SetItem(itemIndex, 4, Telephone);
        Orders_ListCtrl->SetItem(itemIndex, 5, FullDate);
        Orders_ListCtrl->SetItem(itemIndex, 6, Dishes);
        SetTotalOrdersNumber();
        SaveOrdersFile();
        ClearOrderInputData();
        OrderNum++;
    }
    else
        return;
}

```

Listing 7 – metoda `void OnOrderAdd_ButtonClick(wxCommandEvent& event)`, wywoływana po wciśnięciu przycisku „Dodaj” dla nowego zamówienia. Sprawdza, ile dań zostało ustalonych dla zamówienia, po czym kompletuje wprowadzone dane oraz odpowiednio je formatuje. Po dodaniu zamówienia na listę, wywoływane są metody: `SetTotalOrdersNumber()` w celu aktualizacji licznika aktywnych zamówień, `SaveOrdersFile()` w celu zapisania aktualnej listy zamówień do pliku TXT oraz `ClearOrderInputData()` czyszcząca pola wejściowe dla nowego zamówienia.

```

void eStaurantFrame::OnOrderArchive_ButtonClick(wxCommandEvent& event)
{
    long itemNumber=Orders_ListCtrl->GetNextItem(-1, wxLIST_NEXT_ALL,
    wxLIST_STATE_SELECTED);
    wxString Address=Orders_ListCtrl->GetItemText(itemNumber, 1);
    wxString Price=Orders_ListCtrl->GetItemText(itemNumber, 2);
    wxString Payment=Orders_ListCtrl->GetItemText(itemNumber, 3);
}

```

```

wxString Telephone=Orders_ListCtrl->GetItemText(itemNumber, 4);
wxString Date=Orders_ListCtrl->GetItemText(itemNumber, 5);
wxString Dishes=Orders_ListCtrl->GetItemText(itemNumber, 6);
wxTextFile file("data/archives_list.txt");
if(file.Exists())
{
    file.Open("data/archives_list.txt", wxConvUTF8);
    file.AddLine(Address);
    file.AddLine(Price);
    file.AddLine(Payment);
    file.AddLine(Telephone);
    file.AddLine(Date);
    file.AddLine(Dishes);
    file.Write();
    file.Close();
    Orders_ListCtrl->DeleteItem(itemNumber);
    SetTotalOrdersNumber();
    SaveOrdersFile();
}
}

```

Listing 8 – metoda `void OnOrderArchive_ButtonClick(wxCommandEvent &event)`, wywoływana po naciśnięciu przycisku „Archiwizuj” – zapisuje zaznaczony na liście element do pliku `archives_list.txt`, przechowującego dane o archiwum zamówień.

```

void eStaurantFrame::SaveOrdersFile()
{
    ClearFile("data/orders_list.txt");
    if(Orders_ListCtrl->GetItemCount()>0)
    {
        wxFileOutputStream output("data/orders_list.txt");
        if(!output.IsOk())
        {
            wxLogError("Nie można zapisać pliku '%s'.", "data/orders_list.txt");
            return;
        }
        else
        {
            wxTextOutputStream file(output);
            int numItems=Orders_ListCtrl->GetItemCount();
            int numCols=Orders_ListCtrl->GetColumnCount();
            for(int i=0; i<numItems; i++)
            {
                for(int j=0; j<numCols; j++)
                {
                    file << Orders_ListCtrl->GetItemText(i, j);
                    if(!(i+1==numItems && j+1==numCols))
                        file << '\n';
                }
            }
        }
    }
}

```

Listing 9 – metoda `void SaveOrdersFile()`, zapisująca dane z listy zamówień do pliku `orders_list.txt`.

Do dyspozycji użytkownika jest również pasek menu (`wxMenuBar`), umieszczony w górnej części głównego okna. Zawiera on 3 etykiety:

- „Plik”:
 - „Wczytaj ostatnią listę” – wywołuje metodę `OnOrders_ReadClick(wxCommandEvent& event)` - umożliwia wczytanie ostatniej listy zamówień
 - „Lista klientów...” (skrót: Ctrl+K) – wywołuje tę samą metodę, co przycisk „Klienci” – wyświetla okno dialogowe z listą klientów

- „Menu potraw...” (skrót: Ctrl+M) – wywołuje tę samą metodę, co przycisk „Menu” – wyświetla okno dialogowe z listą potraw
- „Archiwum zamówień...” (skrót klawiszowy: Ctrl+N) – wywołuje tę samą metodę, co przycisk „Archiwum” – wyświetla okno dialogowe z archiwum zamówień
- „Zakończ” (skrót: Alt+F4) – wywołuje tę samą metodę *OnClose(wxCloseEvent& event)*, co przycisk zamknięcia w prawym, górnym rogu. Wyświetla okno dialogowe z potwierdzeniem wyjścia z programu i zamyka aplikację, jeśli wybrano „OK”
- „Edycja”:
 - „Archiwizuj zamówienie” – wywołuje tę samą metodę, co przycisk „Archiwizuj” – usuwa wybrane zamówienie z listy i przenosi je do *Archiwum*. Opcja ta jest aktywna jedynie wtedy, gdy wybrano dowolne zamówienie z listy
- „Pomoc”:
 - „eStaurant – informacje” (skrót: Ctrl+I) – wyświetla okno dialogowe *Informacje*, zawierające informacje o projekcie

```
void eStaurantFrame::OnOrders_ReadClick(wxCommandEvent& event)
{
    if(Orders_ListCtrl->GetItemCount()==0)
    {
        int columnIterator=0, lineIterator=0;
        wxFileInputStream input("data/orders_list.txt");
        if(!input.IsOk())
        {
            wxLogError("Nie można otworzyć pliku '%s'.", "data/orders_list.txt");
            return;
        }
    }
    else
    {
        if(input.Eof())
            return;
        wxTextInputStream text(input, wxT("\n"), wxConvUTF8);
        wxString line;
        long itemIndex=Orders_ListCtrl->InsertItem(0, line);
        while(input.IsOk() && !input.Eof())
        {
            text >> line;
            if(columnIterator==0)
            {
                currNum << line;
                FillVector();
            }
            Orders_ListCtrl->SetItem(itemIndex, columnIterator, line);
            columnIterator++;
            if(columnIterator>6)
            {
                lineIterator++;
                columnIterator=0;
                if(!input.Eof())
                    itemIndex=Orders_ListCtrl->InsertItem(lineIterator, line);
            }
        }
        sort(OrdersNumbers.begin(), OrdersNumbers.end());
        SetOrderNumber();
    }
}
```

```

        SetTotalOrdersNumber();
    }
}

```

Listing 10 – metoda `void OnOrders_ReadClick(wxCommandEvent& event)`, odpowiedzialna za wczytanie listy zamówień z pliku. Po pomyślnej operacji, wywoływana jest metoda `FillVector()`, odpowiedzialna za wypełnienie wektora `OrdersNumbers` numerami zamówień, odczytanych z pliku. Numery te następnie są sortowane rosnąco, dzięki czemu metoda `SetOrderNumber()` może w prosty sposób ustalić numer dla nowego zamówienia – wynosi on wartość ostatniego elementu wektora powiększoną o 1.

```

void eStaurantFrame::FillVector()
{
    currNumInt=wxAtoi(currNum);
    OrdersNumbers.push_back(currNumInt);
    currNum=wxT("");
}

int eStaurantFrame::SetOrderNumber()
{
    int n=OrdersNumbers.size()-1;
    if(Orders_ListCtrl->GetItemCount()>0)
        OrderNum=OrdersNumbers[n]+1;
    return OrderNum;
}

```

Listing 11 – metody `void FillVector()` oraz `int SetOrderNumber()`, odpowiedzialne za ustalanie numeru dla nowego zamówienia.

```

void eStaurantFrame::SetTotalOrdersNumber()
{
    wxString OrdersNumber=wxString::Format(wxT("%i"), Orders_ListCtrl->GetItemCount());
    ActiveOrders_TextCtrl->ChangeValue(OrdersNumber);
}

```

Listing 12 – metoda `void SetTotalOrdersNumber()`, aktualizująca licznik aktywnych zamówień.

```

void eStaurantFrame::OnClose(wxCloseEvent& event)
{
    if(ExitConfirmation->ShowModal()==wxID_CANCEL)
        return;
    else
        Destroy();
}

```

Listing 13 – metoda `void OnClose(wxCloseEvent& event)`, wywoływana przy próbie zamknięcia programu.

3.3. Okno dialogowe „Klienci”

Wszystkie okna dialogowe w aplikacji inicjowane są jako okna modalne. Tryb ten zapewnia oknu pierwszorzędną rolę w programie, aż do momentu zamknięcia tego okna. W przeciwieństwie do zwykłych okien, okna modalne tworzone są na stosie, zamiast na stercku. Gwarantuje to większą szybkość oraz płynność działania programu.

Kod źródłowy okna „Klienci” zawarty jest w pliku `Customers_Dialog.cpp`, korzysta przy tym z własnego pliku nagłówkowego `Customers_Dialog.h`.

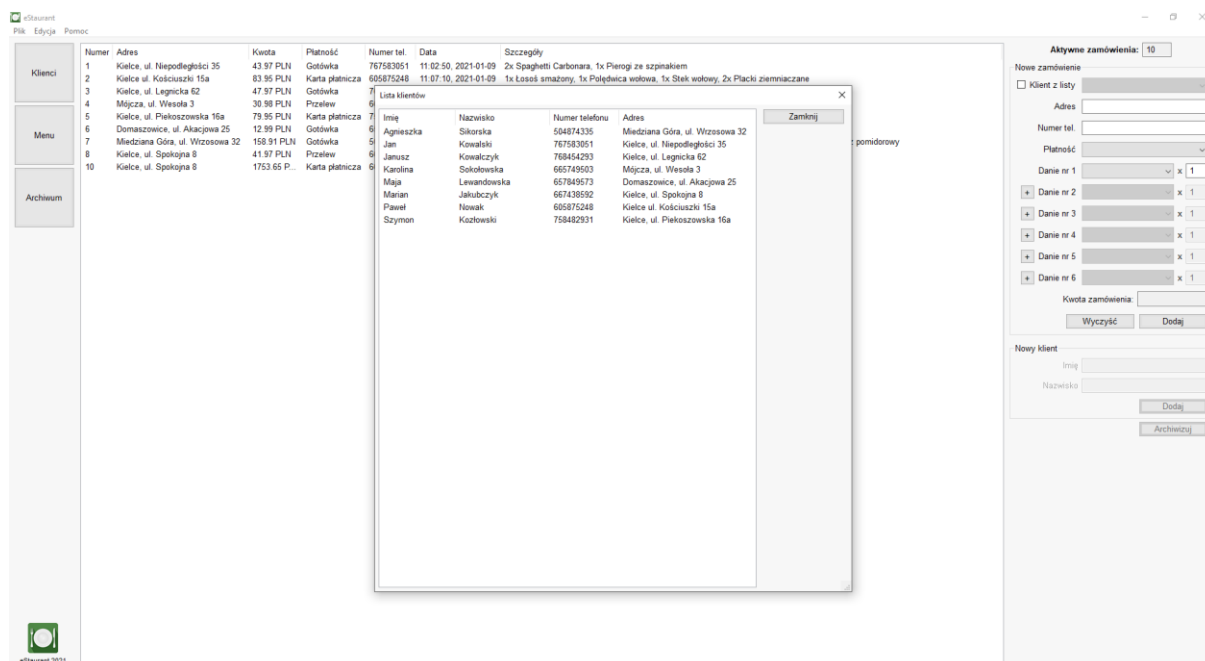
Okno „Klienci” (`Customers_Dialog`) ma bardzo minimalistyczną budowę, składa się bowiem jedynie z listy klientów (`wxListCtrl`) oraz przycisku „Zamknij”, zamykającego okno dialogowe. Lista klientów składa się z 4 kolumn („Imię”, „Nazwisko”, „Numer telefonu” oraz „Adres”), a dane odczytywane są z pliku `customers_list.txt` automatycznie podczas otwarcia okna i natychmiastowo sortowane wg. kolumny o indeksie 0 („Imię”) – od A do Z (metoda `SortItems(int columnNumber)`).

```

Create(parent, wxID_ANY, _T("Lista klientów"), wxDefaultPosition, wxDefaultSize,
wxCAPTION|wxRESIZE_BORDER|wxCLOSE_BOX, _T("wxID_ANY"));
SetBackgroundColour(wxColour(247,247,247));
wxFont
thisFont(10,wxFONTFAMILY_SWISS,wxFONTSTYLE_NORMAL,wxFONTWEIGHT_NORMAL,false,_T("Arial"),wxFO
NTENCODING_DEFAULT);
SetFont(thisFont);

```

Listing 14 – konstruowanie okna dialogowego „Klienci” wraz z ustawieniami domyślnego fontu oraz koloru tła.



Zrzut 3 – okno dialogowe „Klienci” z przykładową listą klientów.

```

void Customers_Dialog::OnInit(wxInitDialogEvent& event)
{
    int columnIterator=0, lineIterator=0;
    wxFileInputStream input("data/customers_list.txt");
    if(!input.IsOk())
    {
        wxLogError("Nie mozna otworzyc pliku '%s'.", "data/customers_list.txt");
        return;
    }
    else
    {
        wxTextInputStream text(input, wxT("\n"), wxConvUTF8);
        wxString line;
        long itemIndex=Customers_ListCtrl->InsertItem(0, line);
        while(input.IsOk() && !input.Eof())
        {
            text >> line;
            Customers_ListCtrl->SetItem(itemIndex, columnIterator, line);
            columnIterator++;
            if(columnIterator>3)
            {
                lineIterator++;
                columnIterator=0;
                if(!input.Eof())
                {
                    itemIndex=Customers_ListCtrl->InsertItem(lineIterator, line);
                }
            }
        }
        SortItems(0);
    }
}

```

Listing 15 – metoda `void OnInit(wxInitDialogEvent& event)`, wywoływana przy otwarciu okna dialogowego.


```

void Customers_Dialog::SortItems(int columnNumber)
{
    int numItems;
    long item1, item2;
    numItems=Customers_ListCtrl->GetItemCount();
    if(columnNumber<0)
        return;
    else
    {
        for(item1=1; item1<numItems; item1++)
        {
            item2=item1;
            while(item2>0 && Customers_ListCtrl->GetItemText(item2-1, columnNumber) >
Customers_ListCtrl->GetItemText(item2, columnNumber))
            {
                SortData(item2, item2-1);
                item2=item2-1;
            }
        }
        return;
    }
}

void Customers_Dialog::SortData(long item1, long item2)
{
    int numCols;
    long tempData;
    wxString tempString;
    numCols=Customers_ListCtrl->GetColumnCount();
    for(int i=0; i<numCols; i++)
    {
        tempString=Customers_ListCtrl->GetItemText(item1, i);
        Customers_ListCtrl->SetItem(item1, i, Customers_ListCtrl->GetItemText(item2, i));
        Customers_ListCtrl->SetItem(item2, i, tempString);
    }
    tempData=Customers_ListCtrl->GetItemData(item1);
    Customers_ListCtrl->SetItemData(item1, Customers_ListCtrl->GetItemData(item2));
    Customers_ListCtrl->SetItemData(item2, tempData);
    return;
}

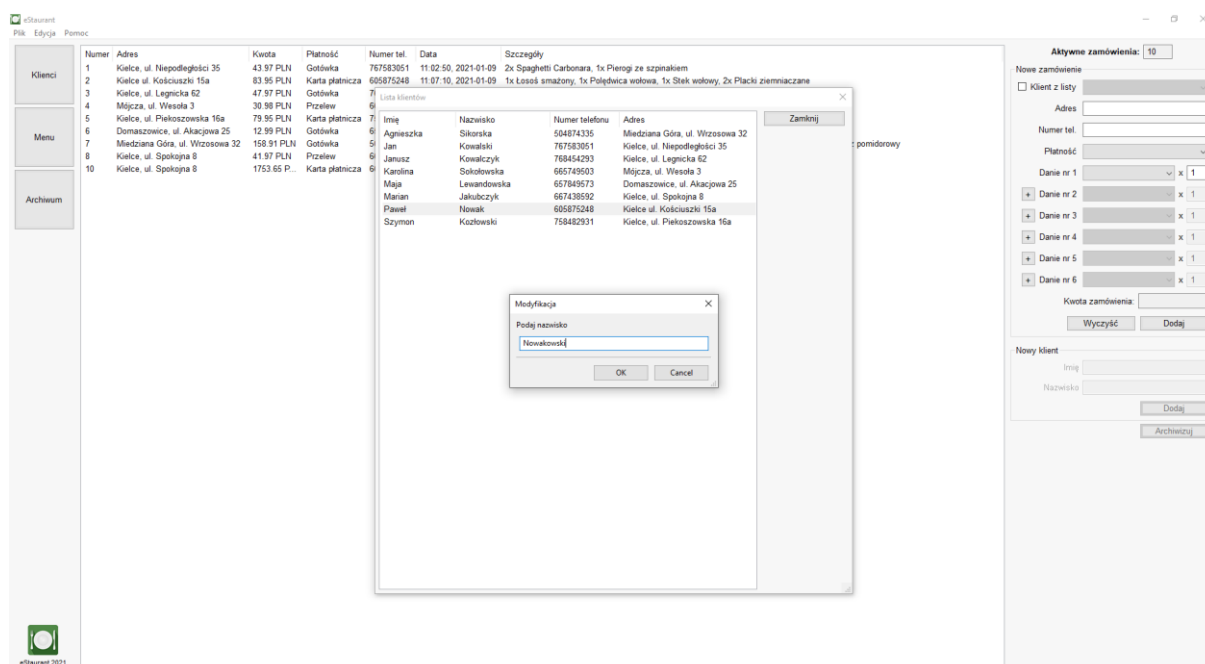
```

Listing 16 – metody `void SortItems(int columnNumber)` i `void SortData(long item1, long item2)`, odpowiedzialne za posortowanie listy w porządku rosnącym (A-Z).

Dane na liście można modyfikować – odbywa się to przy pomocy klasy `wxMenu`. Aby wprowadzić zmiany bądź usunąć dany element z listy, należy zaznaczyć go i wcisnąć prawy przycisk myszy. Wywołuje to metodę `PopupMenu(&CustomersDialog_Menu)`, odpowiedzialną za wyświetlenie menu. Do dyspozycji użytkownika są następujące opcje:

- „Modyfikuj”:
 - „Zmień imię...”
 - „Zmień nazwisko...”
 - „Zmień numer telefonu...”
 - „Zmień adres...”
- „Usuń”

Przy wybraniu dowolnej opcji modyfikującej, wyświetlone zostaje modalne okno dialogowe (`wxTextEntryDialog`) z polem do wprowadzenia nowej wartości dla modyfikowanego elementu. Przy wybraniu opcji „Usuń”, wyświetlone zostaje okno dialogowe (`wxMessageDialog`) z potwierdzeniem operacji. Jeśli wybrano „OK”, element zostaje usunięty z listy.



Zrzut 4 – przykład modyfikacji klienta „Paweł Nowak” – zmiana nazwiska na liście.

```
void Customers_Dialog::OnModifyLastNameSelected(wxCommandEvent& event)
{
    if(Modify_LastNameCtrl->ShowModal()==wxID_CANCEL)
        return;
    else
    {
        itemNumber=Customers_ListCtrl->GetNextItem(-1, wxLIST_NEXT_ALL,
        wxLIST_STATE_SELECTED);
        wxString FirstName=Customers_ListCtrl->GetItemText(itemNumber, 0);
        wxString LastName=Modify_LastNameCtrl->GetValue();
        wxString Telephone=Customers_ListCtrl->GetItemText(itemNumber, 2);
        wxString Address=Customers_ListCtrl->GetItemText(itemNumber, 3);
        Customers_ListCtrl->DeleteItem(itemNumber);
        SetItemDetails(FirstName, LastName, Telephone, Address);
        ClearModifyCtrl();
    }
}
```

Listing 17 – metoda `void OnModifyLastNameSelected(wxCommandEvent& event)`, wywoływana przy próbie modyfikacji nazwiska wybranego klienta. Analogiczne metody występują dla modyfikacji imienia, numeru telefonu oraz adresu. Do zmiennej `wxString LastName` przypisywana jest nowa wartość, pobrana z okna dialogowego (`wxTextEntryDialog`), natomiast do pozostałych zmiennych tego typu przypisywane są dotychczasowe wartości znajdujące się w poszczególnych kolumnach dla danego elementu. Po przypisaniu wartości zmiennym, modyfikowany element jest usuwany, po czym wywoływana jest metoda odpowiedzialna za ponowne przypisanie zapisanych wartości do elementu o indeksie zmodyfikowanego elementu (Listing 13). Na końcu wszystkie kontrolki wejściowe okien dialogowych odpowiedzialnych za wprowadzanie zmodyfikowanych danych są czyszczone (metoda `ClearModifyCtrl()`).

```
void Customers_Dialog::SetItemDetails(wxString FirstName, wxString LastName, wxString Telephone, wxString Address)
{
    long itemIndex=Customers_ListCtrl->InsertItem(itemNumber, FirstName);
    Customers_ListCtrl->SetItem(itemIndex, 0, FirstName);
    Customers_ListCtrl->SetItem(itemIndex, 1, LastName);
    Customers_ListCtrl->SetItem(itemIndex, 2, Telephone);
    Customers_ListCtrl->SetItem(itemIndex, 3, Address);
    SaveCustomersFile();
}
```

Listing 18 – metoda `void SetItemDetails(wxString FirstName, wxString LastName, wxString Telephone, wxString Address)`, wywoływana podczas modyfikowania elementów listy. Jako argumenty przyjmuje ona wartości zmiennych `wxString`, ustawione podczas wykonywania funkcji `OnModifyLastNameSelected(wxCommandEvent& event)`. Po wykonanej operacji, plik z listą

klientów jest zapisywany przy pomocy metody *SaveCustomersFile()*, realizującej bardzo zbliżone instrukcje do tych z funkcji *SaveOrdersFile()* (Listing 5).

3.4. Okno dialogowe „Menu”

Kod źródłowy okna „Menu” zawarty jest w pliku *Menu_Dialog.cpp*, natomiast plik nagłówkowy dotyczący tego okna to *Menu_Dialog.h*.

Modalne okno dialogowe „Menu” (*Menu_Dialog*) można, podobnie jak główne okno aplikacji, podzielić na dwie części:

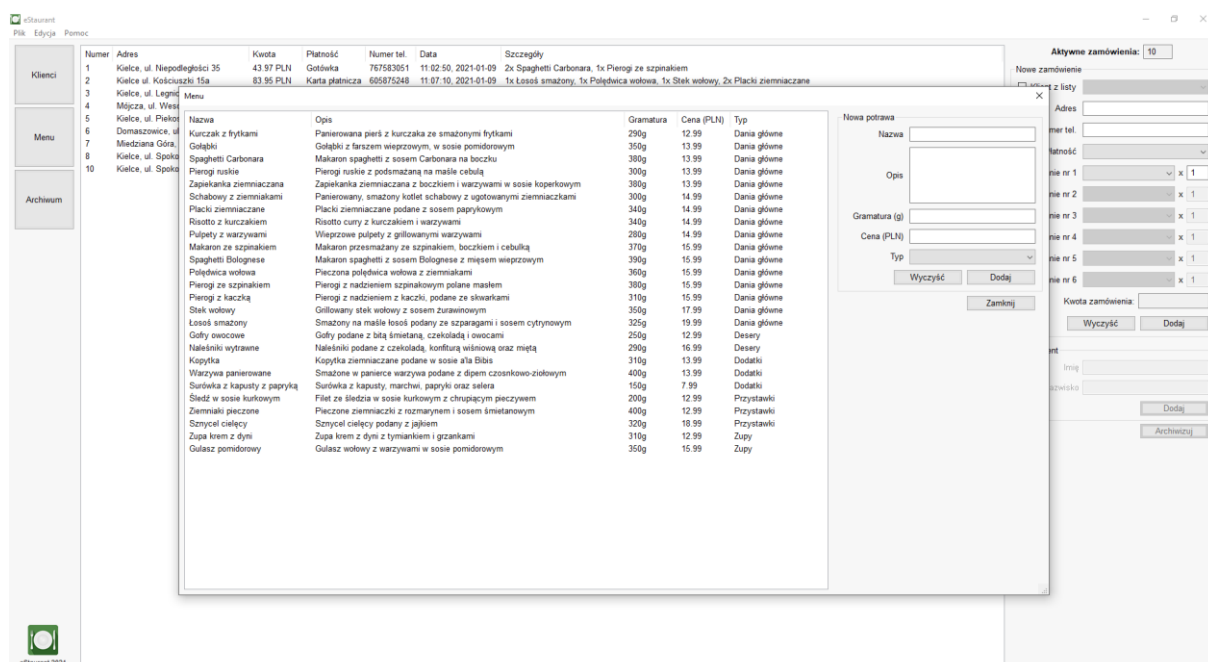
- I część (lewa, środkowa) – zawiera listę potraw (*wxListCtrl*) podzieloną na 5 kolumn:
 - „Nazwa” – nazwa potrawy
 - „Opis” – szczegóły dotyczące danej potrawy
 - „Gramatura” – waga jednej porcji danej potrawy (wyrażona w gramach)
 - „Cena (PLN)” – cena za porcję
 - „Typ” – typ danej potrawy

Zawartość listy jest automatycznie wczytywana z pliku *dishes_list.txt* podczas inicjowania okna dialogowego (metoda *OnInit(wxInitDialogEvent& event)*).

- II część (prawa) – zawiera panel umożliwiający dodanie nowej potrawy do listy. Zawiera pola wejściowe (*wxTextCtrl*) o nazwach identycznych, jak nazwy w/w kolumn listy, a także pole wyboru (*wxComboBox*) „Typ” z opcjami: „Przystawki”, „Dania główne”, „Zupy”, „Desery”, „Dodatki”, „Napoje” oraz „Inne”. Ponadto w panelu zawarte są przyciski „Dodaj”, „Wyczyść” i poniżej „Zamknij”. Odpowiadają one kolejno za dodanie nowej potrawy do listy (*OnDishAdd_ButtonClick(wxCommandEvent& event)*), wyczyszczenie pól wejściowych (*OnDishClear_ButtonClick(wxCommandEvent& event)*), zamknięcie okna dialogowego (*OnClose_ButtonClick(wxCommandEvent& event)*).

```
Create(parent, wxID_ANY, _T("Menu"), wxDefaultPosition, wxDefaultSize,
wxCAPTION|wxRESIZE_BORDER|wxCLOSE_BOX, _T("wxID_ANY"));
SetBackgroundColour(wxColour(247,247,247));
wxFont
thisFont(10,wxFONTFAMILY_SWISS,wxFONTSTYLE_NORMAL,wxFONTWEIGHT_NORMAL,false,_T("Arial"),wxFO
NTENCODING_DEFAULT);
SetFont(thisFont);
```

Listing 19 – konstruowanie okna dialogowego „Menu” wraz z ustawieniami domyślnego fontu oraz koloru tła.



Zrzut 5 – okno dialogowe „Menu” z przykładową listą potraw.

```
void Menu_Dialog::OnInit(wxInitDialogEvent& event)
{
    int columnIterator=0, lineIterator=0;
    wxFileInputStream input("data/dishes_list.txt");
    if(!input.IsOk())
    {
        wxLogError("Nie mozna otworzyc pliku '%s'.", "data/dishes_list.txt");
        return;
    }
    else
    {
        wxTextInputStream text(input, wxT("\n"), wxConvUTF8);
        wxString line;
        long itemIndex=Menu_ListCtrl->InsertItem(0, line);
        while(input.IsOk() && !input.Eof())
        {
            text >> line;
            Menu_ListCtrl->SetItem(itemIndex, columnIterator, line);
            columnIterator++;
            if(columnIterator>4)
            {
                lineIterator++;
                columnIterator=0;
                if(!input.Eof())
                    itemIndex=Menu_ListCtrl->InsertItem(lineIterator, line);
            }
        }
    }
}
```

Listing 20 – metoda `void OnInit(wxInitDialogEvent& event)`, inicjowana podczas uruchamiania okna dialogowego. Wczytuje dane z pliku TXT do listy potraw.

```
void Menu_Dialog::OnDishAdd_ButtonClick(wxCommandEvent& event)
{
    long maxIndex=Menu_ListCtrl->GetItemCount();
    wxString Name=Name_TextCtrl->GetValue();
    wxString Description=Description_TextCtrl->GetValue();
    wxString Weight=Weight_TextCtrl->GetValue();
    Weight << wxT("g");
    wxString Price=Price_TextCtrl->GetValue();
    wxString Type=Type_ComboBox->GetValue();
```

```

if(!(Name==wxT("") || Description==wxT("") || Weight==wxT("") || Price==wxT("") ||
Type==wxT("")))
{
    long itemIndex=Menu_ListCtrl->InsertItem(maxIndex, Name);
    Menu_ListCtrl->SetItem(itemIndex, 0, Name);
    Menu_ListCtrl->SetItem(itemIndex, 1, Description);
    Menu_ListCtrl->SetItem(itemIndex, 2, Weight);
    Menu_ListCtrl->SetItem(itemIndex, 3, Price);
    Menu_ListCtrl->SetItem(itemIndex, 4, Type);
    SaveMenuFile();
    ClearDishInputData();
}
else
    return;
}

```

Listing 21 – metoda `void OnDishAdd_ButtonClick(wxCommandEvent& event)`, dodająca do listy nową potrawę na podstawie wprowadzonych przez użytkownika danych. Potrawa zostanie dodana tylko wtedy, kiedy wszystkie pola wejściowe zostały wypełnione i został wybrany jej typ. Po zakończonej operacji, lista zapisywana jest do pliku a pola wejściowe zostają wyczyszczone.

```

void Menu_Dialog::OnDishClear_ButtonClick(wxCommandEvent& event)
{
    ClearDishInputData();
}

void Menu_Dialog::ClearDishInputData()
{
    Name_TextCtrl->Clear();
    Description_TextCtrl->Clear();
    Weight_TextCtrl->Clear();
    Price_TextCtrl->Clear();
    Type_ComboBox->SetSelection(-1);
}

```

Listing 22 – metoda `void OnDishClear_ButtonClick(wxCommandEvent &event)`, wykonywana po wciśnięciu przycisku „Wyczyść”. Wywołuje funkcję `void ClearDishInputData()`, odpowiedzialną za czyszczenie kontrolki wejściowych (funkcja ta jest również wywoływana automatycznie po dodaniu nowej potrawy).

```

void Menu_Dialog::OnClose_ButtonClick(wxCommandEvent& event)
{
    SaveMenuFile();
    EndModal(0);
}

```

Listing 23 – metoda `void OnClose_ButtonClick(wxCommandEvent& event)`, wywoływana przy próbie zamknięcia okna dialogowego. Zapisuje listę potraw do pliku i zamyka okno.

Podobnie jak w przypadku okna „Klienci”, elementy znajdujące się na liście potraw można modyfikować oraz usuwać. Odbywa się to przy pomocy klasy `wxMenu`, a sposoby implementacji poszczególnych operacji są niemal identyczne jak dla listy klientów.

3.5. Okno dialogowe „Archiwum”

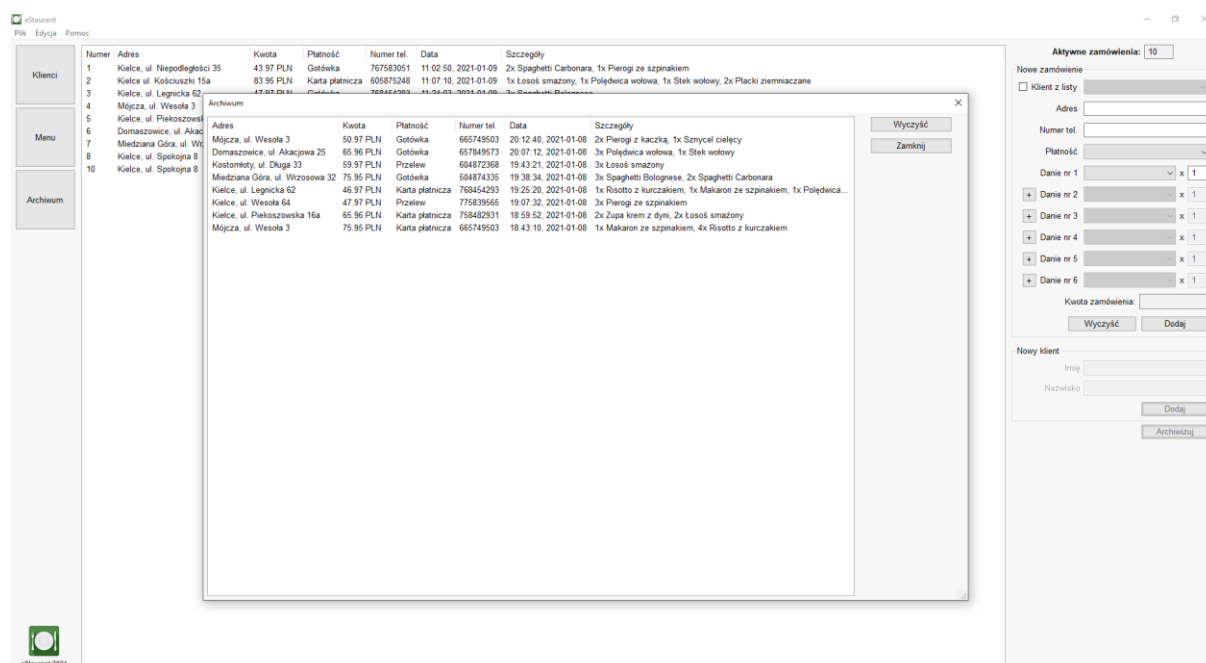
Kod okna „Archiwum” zawarty jest w pliku źródłowym `Archive_Dialog.cpp`, przy czym korzysta on z własnego pliku nagłówkowego `Archive_Dialog.h`.

„Archiwum” (`Archive_Dialog`) jest kolejnym modalnym oknem dialogowym, zawierającym listę (`wxListCtrl`). Jej zawartość jest automatycznie wczytywana z pliku `archives_list.txt` podczas uruchamiania okna (metoda `OnInit(wxInitDialogEvent& event)`). Przechowuje wszystkie zarchiwizowane zamówienia i posiada prawie wszystkie kolumny identyczne jak w przypadku listy zamówień, za wyjątkiem kolumny „Numer”. Ponadto w prawej części okna

umieszczono dwa przyciski: „Wyczyść” służący do usunięcia wszystkich elementów z listy (metoda `OnClearArchive_ButtonClick(wxCommandEvent& event)`) oraz „Zamknij” do zamknięcia okna dialogowego (`OnCloseArchive_ButtonClick(wxCommandEvent &event)`).

```
Create(parent, wxID_ANY, _T("Archiwum"), wxDefaultPosition, wxDefaultSize,
wxCAPTION|wxRESIZE_BORDER|wxCLOSE_BOX, _T("wxID_ANY"));
SetBackgroundColour(wxColour(247,247,247));
wxFont
thisFont(10,wxFONTFAMILY_SWISS,wxFONTSTYLE_NORMAL,wxFONTWEIGHT_NORMAL,false,_T("Arial"),wxFONTENCODING_DEFAULT);
SetFont(thisFont);
```

Listing 24 – konstruowanie okna dialogowego „Archiwum” wraz z ustawieniami domyślnego fontu oraz koloru tła.



Zrzut 6 – okno dialogowe „Archiwum” z przykładową listą zarchiwizowanych zamówień.

```
void Archive_Dialog::OnInit(wxInitDialogEvent& event)
{
    int columnIterator=0, lineIterator=0;
    wxFileInputStream input("data/archives_list.txt");
    if(!input.IsOk())
    {
        wxLogError("Nie mozna otworzyc pliku '%s'.", "data/archives_list.txt");
        return;
    }
    else
    {
        if(input.Eof())
            return;
        wxTextInputStream text(input, wxT("\n"), wxConvUTF8);
        wxString line;
        long itemIndex=Archive_ListCtrl->InsertItem(0, line);
        while(input.IsOk() && !input.Eof())
        {
            text >> line;
            Archive_ListCtrl->SetItem(itemIndex, columnIterator, line);
            columnIterator++;
            if(columnIterator>5)
            {
                lineIterator++;
                columnIterator=0;
                if(!input.Eof())
                    itemIndex=Archive_ListCtrl->InsertItem(lineIterator, line);
            }
        }
    }
}
```

```

    }
}
}

```

Listing 25 – metoda `void OnInit(wxInitDialogEvent& event)` wywoływana podczas otwierania okna dialogowego. Wczytuje do listy archiwum zamówień z pliku TXT.

```

void Archive_Dialog::OnClearArchive_ButtonClick(wxCommandEvent& event)
{
    if(ArchiveConfirmClear_Dialog->ShowModal()==wxID_CANCEL)
        return;
    else
    {
        if(!Archive_ListCtrl->DeleteAllItems())
        {
            wxLogError("Nie można wyczyścić archiwum.");
            return;
        }
    }
    wxTextFile file("data/archives_list.txt");
    if(file.Exists())
    {
        file.Open("data/archives_list.txt", wxConvUTF8);
        file.Clear();
        file.Write();
        file.Close();
    }
}

```

Listing 26 – metoda `void OnClearArchive_ButtonClick(wxCommandEvent& event)`, wywoływana po wciśnięciu przycisku „Wyczyść” – najpierw usuwa wszystkie elementy z listy, a następnie czyści plik TXT przechowujący dane dla listy.

3.6. Okno dialogowe „eStaurant - Informacje”

Kod źródłowy okna „eStaurant – Informacje” zawarty jest w pliku *Info_Dialog.cpp*, natomiast wykorzystywany plik nagłówkowy to *Info_Dialog.h*.

Modalne okno dialogowe „eStaurant - Informacje” (*Info_Dialog*) jest oknem zawierającym najważniejsze z punktu widzenia użytkownika informacje o programie oraz jego twórcy. Składa się z umieszczonego w górnej części logo aplikacji o wymiarach 128x128 pikseli (*wxStaticBitmap*), części tekstowej (*wxStaticText*) oraz przycisku „Zamknij”, będącego jedynym przyciskiem umożliwiającym zamknięcie okna dialogowego (okno nie posiada paska tytułowego, więc nie możliwe jest jego zamknięcie poprzez wciśnięcie „krzyżyka”).

```

Create(parent, wxID_ANY, _T("eStaurant"), wxDefaultPosition, wxDefaultSize, 0,
_T("wxID_ANY"));
SetBackgroundColour(wxColour(247,247,247));
wxFont
thisFont(10,wxFONTFAMILY_SWISS,wxFONTSTYLE_NORMAL,wxFONTWEIGHT_NORMAL,false,_T("Arial"),wxFONTENCODING_DEFAULT);
SetFont(thisFont);

```

Listing 27 – konstruowanie okna dialogowego „eStaurant – informacje” wraz z ustawieniami domyślnego fontu oraz koloru tła.

eStaurant

Plik Edycja Pomoc

Klienci

Menu

Archiwum

Numer	Adres	Kwota	Płatność	Numer tel.	Data	Szczegóły
1	Kielce, ul. Niepodległości 35	43.97 PLN	Gotówka	767592651	11.02.50, 2021-01-09	2x Spaghetti Carbonara, 1x Pierogi ze szpinakiem
2	Kielce, ul. Koszusiaka 15a	83.95 PLN	Karta płatnicza	665675248	11.07.10, 2021-01-09	1x Losos smażony, 1x Polędwica wołowa, 1x Stek wołowy, 2x Placki ziemniaczane
3	Kielce, ul. Legnicka 62	47.97 PLN	Gotówka	768454293	11.24.03, 2021-01-09	3x Spaghetti Bolognese
4	Mójca, ul. Wesoła 3	30.98 PLN	Pizzelew	665749503	11.34.30, 2021-01-09	1x Risotto z kurczakiem, 1x Makaron ze szpinakiem
5	Kielce, ul. Piękoszowska 16a	79.95 PLN	Karta płatnicza	758482931	11.51.46, 2021-01-09	2x Pierogi ze szpinakiem, 3x Pierogi z kaczką
6	Domaszowice, ul. Alakjowa 25	12.99 PLN	Gotówka	657849573	11.57.32, 2021-01-09	1x Zupa krem z dyni
7	Miedziana Góra, ul. Wiczosowa 32	159.91 PLN	Gotówka	504874335	12.03.56, 2021-01-09	3x Losos smażony, 2x Stek wołowy, 1x Risotto z kurczakiem, 1x Polędwica wołowa, 2x Gulasz pomidorowy
8	Kielce, ul. Spokojna 8	41.97 PLN	Pizzelew	667438592	12.10.23, 2021-01-09	2x Golby owocowe, 1x Polędwica wołowa
10	Kielce, ul. Spokojna 8	1753.65 P...	Karta płatnicza	667438592	12.12.40, 2021-01-09	135x Kurczak z frytkami

eStaurant

Kontakt: bukowski@gmail.com

© 2020-2021 Bartosz Bukowski. Wszelkie prawa zastrzeżone.

Zamknij

Aktywne zamówienia: 19

Nowe zamówienie

Klient z listy

Adres

Numer tel.

Płatność

Danie nr 1

Danie nr 2

Danie nr 3

Danie nr 4

Danie nr 5

Danie nr 6

Kwota zamówienia:

Wyczyść

Dodaj

Nowy klient

Imię

Nazwisko

Dodaj

Archiwizuj

Zrzut 7 – okno dialogowe „eStaurant – informacje”.

4. Testowanie projektu

Wykonano testy funkcjonalne („testy czarnej skrzynki”) aplikacji. Dokonywano przy tym analizy jedynie zewnętrznej części projektu, nie odnosząc się do jego wewnętrznej budowy. Zakładano więc, że testy odbywają się bez jakiejkolwiek wiedzy nt. użytych technologii i sposobów implementacji poszczególnych funkcji programu.

Testy przeprowadzano osobno na trzech ekranach: o rozdzielczości 1920x1080, 1440x900 oraz 1366x768 pikseli. Specyfikacja urządzenia:

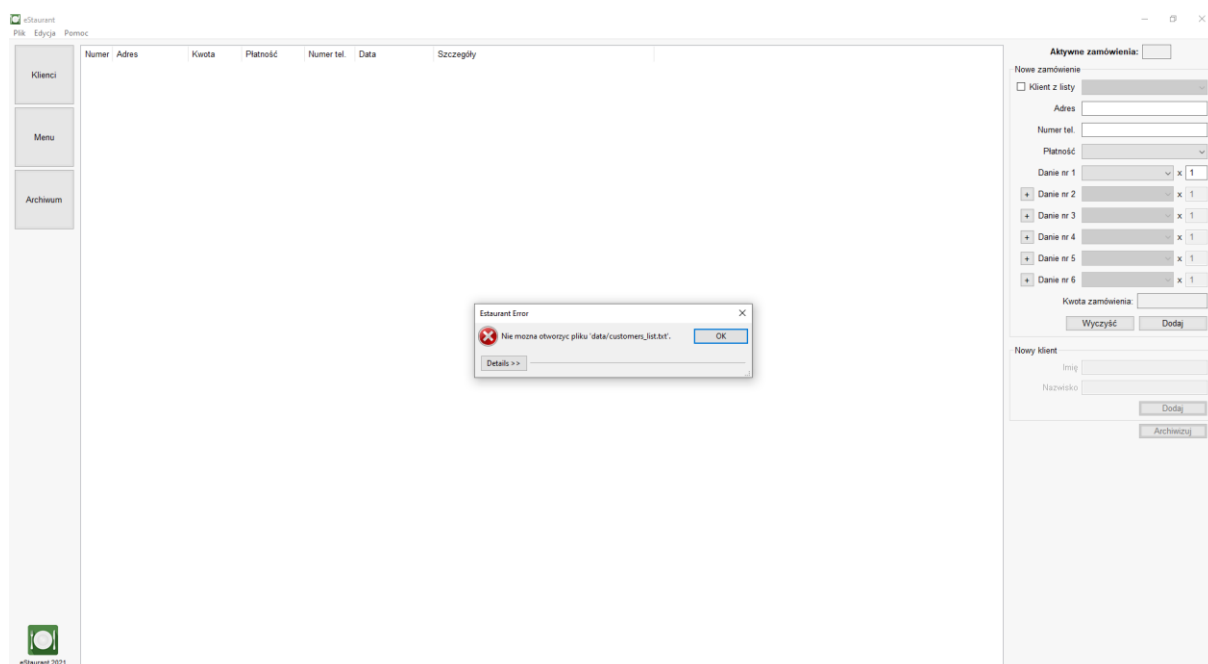
- Procesor Intel Core i7-4700HQ 2,40GHz, 4 rdzenie, 8 wątków, 6 MB pamięci Cache
- 8 GB RAM (SO-DIMM DDR3, 1600MHz)
- Karta graficzna NVIDIA GeForce GTX 850M, 2 GB DDR3
- 64-bitowy system operacyjny Windows 10 Home w wersji 1909

Przeprowadzono kilkadziesiąt prób uruchomienia oraz zamknięcia programu – wszystkie przebiegły pomyślnie. Ekran powitalny wyświetlany jest przez ok. 1,5 sekundy, natomiast czas od uruchomienia, do wyświetlenia głównego okna aplikacji nie wyniósł w żadnej próbie więcej niż 1 sekundę.

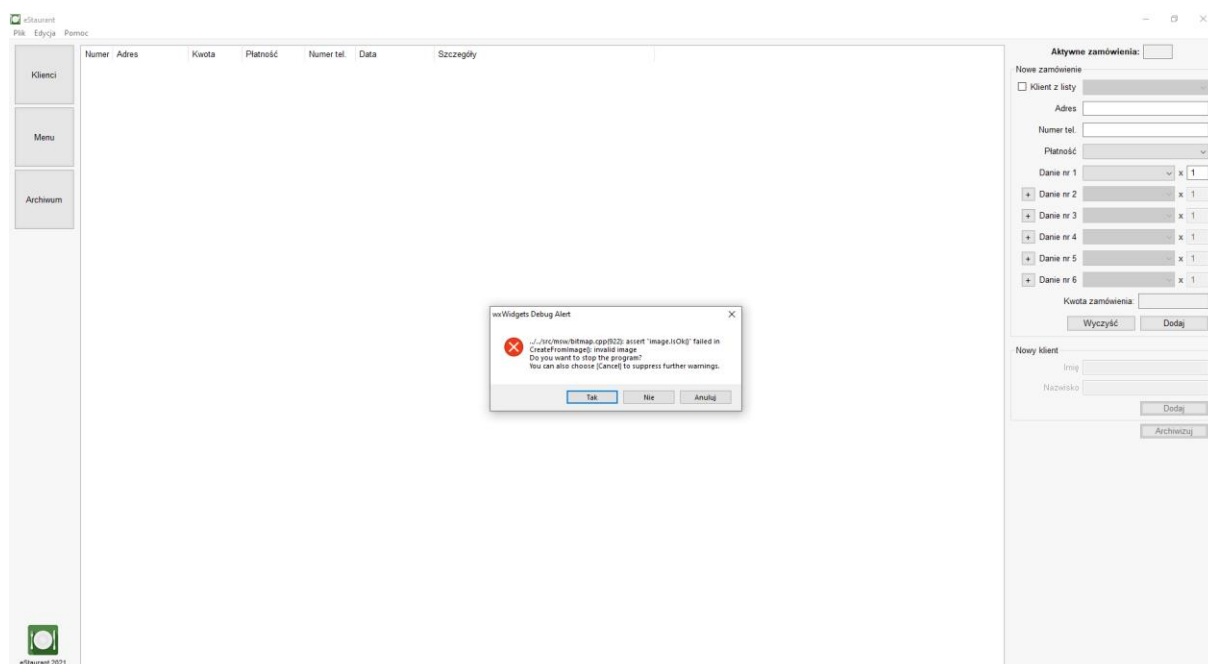
Zauważono chwilowy spadek wydajności aplikacji przy zamykaniu okna dialogowego „Menu” przy ok. 30 pozycjach na liście potraw. Spadek ten rósł wraz ze wzrostem liczby pozycji na liście. Ten sam problem zauważono w przypadku zamykania okna „Klienci” przy podobnej liczbie elementów listy wyświetlanej w tym oknie. Nie stwierdzono występowania tego problemu dla okna „Archiwum”.

Sprawdzono obsługę błędów przez aplikację. Program poprawnie wyświetla komunikat o wystąpieniu błędu w przypadku braku któregoś z plików TXT, podając przy tym nazwę brakującego pliku. Nie przerywa to działania programu, jednak może nie być możliwe korzystanie w pełni ze wszystkich funkcji programu. Podobnie zachowuje się w przypadku braku któregoś z plików graficznych – aplikacja informuje użytkownika o pojawieniu się błędu, natomiast to użytkownik może zdecydować co chce w takim wypadku zrobić. Może zamknąć program lub zdecydować się na dalsze z niego korzystanie - większość funkcji aplikacji będzie działać normalnie, jednak nie zostaną wyświetlone brakujące pliki graficzne. Prawidłowo wyświetlany jest również komunikat o błędzie w przypadku problemów z konwersją zmiennej tekstowej na liczbową (jeśli podczas ustalania ceny dla nowej potrawy, część dziesiętną od całkowitej oddzielono przecinkiem, zamiast kropką).

Problemem aplikacji może okazać się dosyć słabe wsparcie dla niskich rozdzielczości ekranu. Aplikacja poprawnie wyświetla się dla rozdzielczości 1920x1080, 1440x900 oraz 1366x768, natomiast uruchomienie jej na ekranach o niższych parametrach nie gwarantuje w pełni prawidłowego wyświetlenia okna głównego oraz okien dialogowych. Nie wiadomo jak aplikacja wyświetla się na ekranach o większych rozdzielczościach, takich jak 2K, 4K, 8K itd. Wykonanie takich testów jest niemożliwe ze względu na ograniczenia sprzętowe.



Zrzut 8 – błąd wynikający z braku pliku *customers_list.txt*.



Zrzut 9 – błąd podczas próby wyświetlenia okna dialogowego „eStaurant – informacje”, wynikający z braku pliku graficznego *logo.bmp*.

5. Wnioski

Udało się wykonać wszystkie założenia ustalone podczas pierwszych zajęć projektowych. Program jest zaopatrzony we wszelkie funkcje, których potrzebę implementacji określono podczas deklaracji tematu projektu. Pozwala na wygodną i prostą w użytkowaniu obsługę zamówień „na wynos” dla małych lokali gastronomicznych. Nie wymaga połączenia z internetem oraz posiada niskie wymagania sprzętowe. Może stanowić ciekawą alternatywę dla drogich, komercyjnych aplikacji, oferując najpotrzebniejsze do pełnienia swojej roli funkcje.

Ponadto opracowano logo projektu, mając na celu zwiększenie estetyki i prestiżu aplikacji. Samodzielnie stworzono również ekran powitalny w formie pliku graficznego BMP.

Kolejnym krokiem w rozwoju projektu mogłoby być przejście z technologii zapisu i odczytu danych z plików TXT, do obsługi systemów zarządzania bazami danych, takich jak np. Oracle Database. Pozwoliłoby to na korzystanie ze znacznie większych zbiorów danych, a także umożliwiłoby stworzenie bezpiecznego systemu, zapewniającego ochronę danych (np. poprzez wymagane zalogowanie się przed skorzystaniem z aplikacji).

Kolejnym pomysłem na rozwój projektu jest dodanie takich opcji, jak importowanie list do plików PDF oraz drukowanie list bezpośrednio z poziomu programu. Takie rozwiązanie mogłoby znacząco zwiększyć komfort pracy z programem.

Aplikację można by również wyposażyć w opcję doboru czcionek, ich rodzaju oraz rozmiaru. Mogłoby to pomóc np. osobom słabowidzącym, poprzez poprawienie czytelności wyświetlanego tekstu.

W celu wprowadzenia aplikacji na rynek zagraniczny oraz zwiększenia liczby potencjalnych odbiorców, należałoby dodać opcję wyboru języka interfejsu programu. Minimum opcji powinny stanowić język polski oraz angielski.

Ponadto, interfejs aplikacji warto by przebudować tak, aby wspierał on wyświetlanie okna głównego oraz okien dialogowych również na ekranach o rozdzielczościach mniejszych niż 1366x768 pikseli.

Praca nad projektem pozwoliła lepiej zrozumieć mechanizmy języka C++ oraz pomogła w zrozumieniu idei projektowania GUI. Doświadczenie zdobyte podczas budowania aplikacji może okazać się przydatne w tworzeniu programów z interfejsem graficznym w przyszłości.

6. Lista załączników projektowych

Poniżej zaprezentowana zostaje lista wszystkich plików, jakie zostały załączone w ramach oddania projektu zaliczeniowego z przedmiotu „Programowanie w języku C 2”:

- Folder *eStaurant* – skompilowana aplikacja gotowa do uruchomienia
 - Folder *data*
 - *archives_list.txt*
 - *customers_list.txt*
 - *dishes_list.txt*
 - *orders_list.txt*
 - Folder *resources*
 - *icon.ico*
 - *icon2.ico*
 - *logo.bmp*
 - *logo_small.bmp*
 - *splash.bmp*
 - *eStaurant.exe*
 - *libgcc_s_seh-1.dll*
 - *libstdc++-6.dll*
 - *libwinpthread-1.dll*
 - *libwxmsw30u_adv.a*
 - *wxbase30u_gcc810_x64.dll*
 - *wxmsw30u_adv_gcc810_x64.dll*
 - *wxmsw30u_core_gcc810_x64.dll*
- Folder *eStaurant source files* – wszystkie pliki źródłowe utworzone w ramach budowania aplikacji
 - Folder *obj*
 - Folder *Debug*
 - *Archive_Dialog.o*
 - *Customers_Dialog.o*
 - *eStaurantApp.o*
 - *eStaurantMain.o*
 - *Info_Dialog.o*
 - *Menu_Dialog.o*
 - *resource.res*
 - Folder *data*
 - *Archives_list.txt*
 - *Customers_list.txt*

- *Dishes_list.txt*
- *Orders_list.txt*
- Folder *resources*
 - *Icon.ico*
 - *Icon2.ico*
 - *Logo.bmp*
 - *Logo_small.bmp*
 - *Splash.bmp*
- *Archive_Dialog.cpp*
- *Archive_Dialog.h*
- *Customers_Dialog.cpp*
- *Customers_Dialog.h*
- *eStaurant.cbp*
- *eStaurantApp.cpp*
- *eStaurantApp.h*
- *eStaurantMain.cpp*
- *eStaurantMain.h*
- *Info_Dialog.cpp*
- *Info_Dialog.h*
- *Menu_Dialog.cpp*
- *Menu_Dialog.h*
- *version.h*
- *resource.rc*

7. Instrukcja uruchomienia

7.1. Wymagania systemowe

Wymagania sprzętowe aplikacji są bardzo niskie, dlatego większość współczesnych komputerów osobistych z 64-bitowym systemem Windows 7 lub nowszym, powinna bez problemu uruchomić i obsłużyć program.

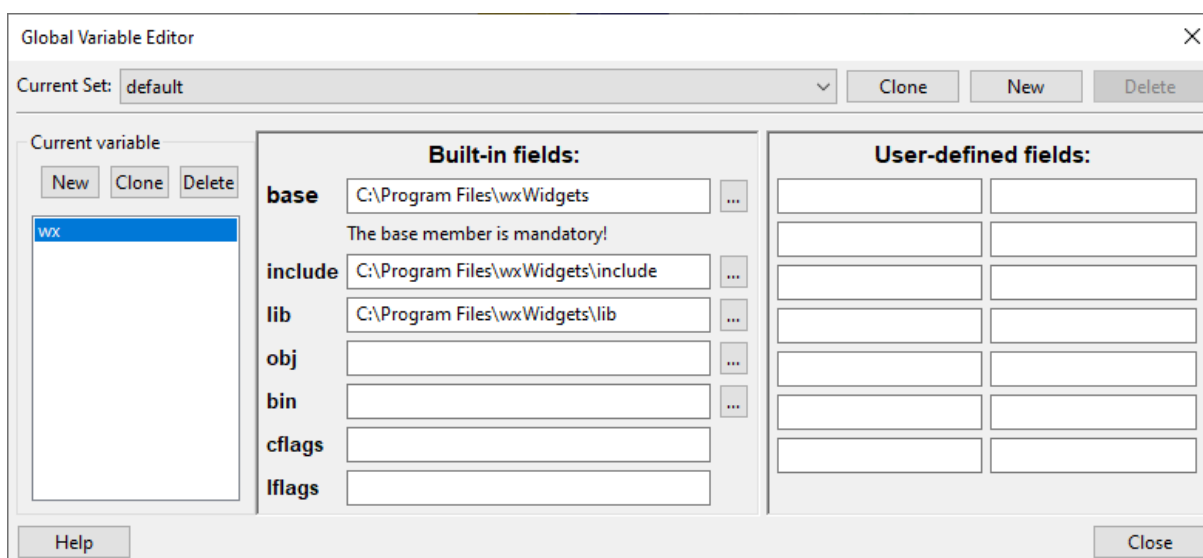
Aplikacja wspiera 64-bitowe systemy operacyjne: Windows 7, Windows 8 oraz Windows 10. Wymagane jest co najmniej 25 MB wolnego miejsca na dysku. Do płynnego działania, program potrzebuje przynajmniej 128MB wolnej pamięci RAM. Sugerowana rozdzielczość wyświetlania nie powinna być mniejsza niż 1366x768 pikseli. Uruchomienie aplikacji na ekranach o mniejszych rozdzielczościach nie gwarantuje wyświetlenia poprawnego i w pełni oddanego realnemu wyglądowi interfejsu.

7.2. Instrukcja kompilacji projektu

Poniższa instrukcja kompilacji dotyczy środowiska Code::Blocks 20.03. Środowisko to można zainstalować przy pomocy pliku *codeblocks-20.03mingw-setup.exe* zawartego w folderze *Installation files*, postępując zgodnie z poleceniami widocznymi na ekranie.

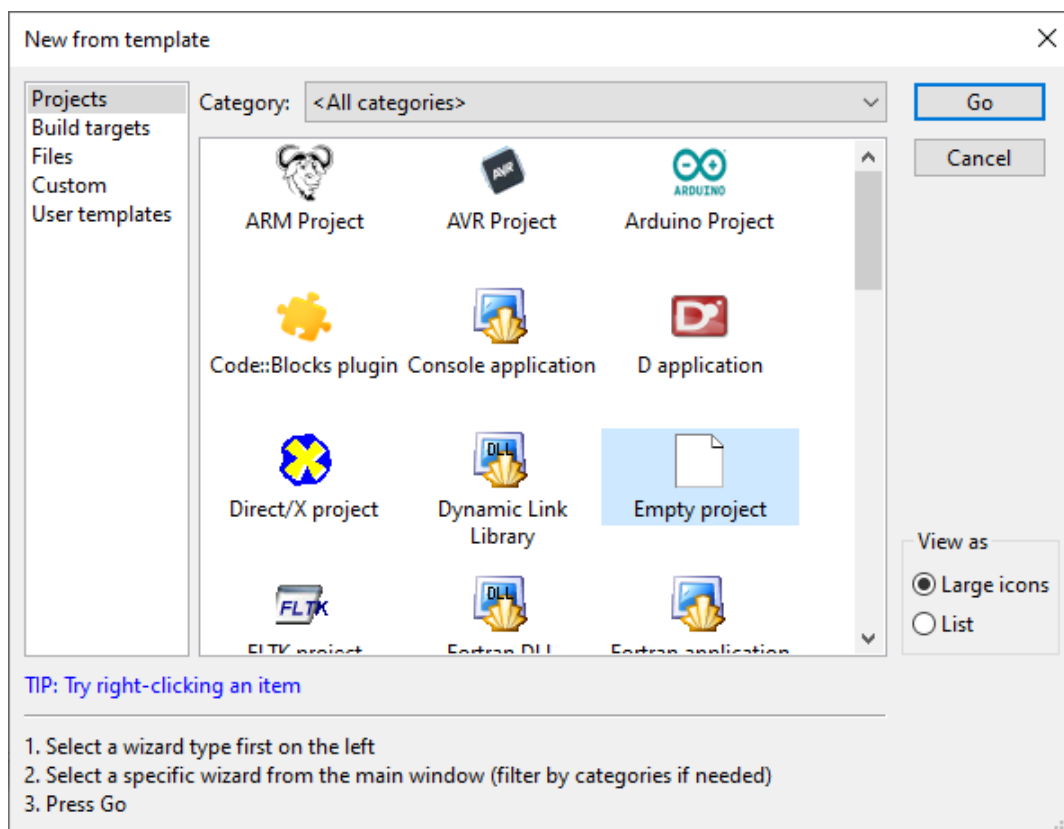
Do skompilowania i uruchomienia programu, konieczne jest posiadanie zainstalowanej biblioteki wxWidgets w wersji 3.0.5. W tym celu należy wypakować znajdujące się w folderze *Installation files* archiwum *wxWidgets-3.0.5.zip*, np. do katalogu *C:\Program Files*.

Po uruchomieniu środowiska Code:Blocks, należy w pierwszej kolejności na pasku menu wejść w „Settings”, a następnie „Global variables...” i dodać nową zmienną o nazwie „wx”. W „Built-in fields:” należy w „base” podać ścieżkę do głównego folderu wxWidgets (np. „C:\Program Files\wxWidgets”), w „include” – ścieżkę do katalogu „include” znajdującego się w folderze wxWidgets (np. „C:\Program Files\wxWidgets\include”), a w „lib” – ścieżkę do katalogu „lib” w folderze wxWidgets (np. „C:\Program Files\wxWidgets\lib”).



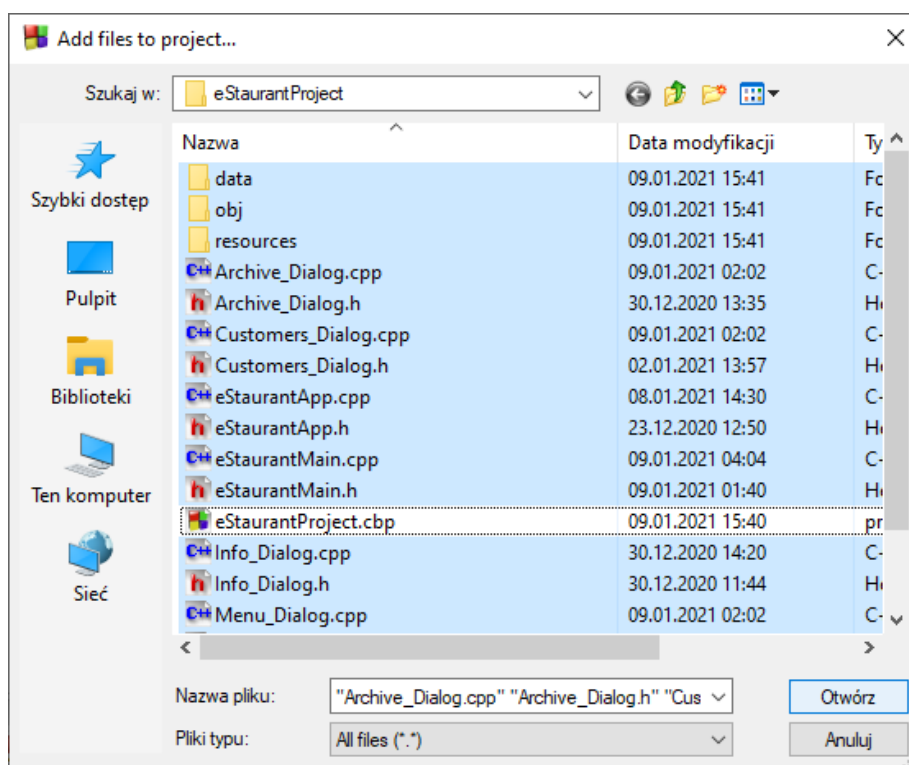
Zrzut 10 – przykładowo wypełnione okno „Global Variable Editor”.

Po wykonaniu powyższych czynności można przystąpić do tworzenia projektu. Należy stworzyć nowy projekt poprzez „File” > „New” > „Project...”, a następnie wybrać opcję „Empty project”. Należy wpisać tytuł dla projektu, np. „eStaurantProject” i utworzyć projekt.



Zrzut 11 – tworzenie nowego, pustego projektu.

Kolejnym krokiem jest skopiowanie wszystkich plików z załączonego katalogu „eStaurant source files” do głównego katalogu nowo utworzonego projektu. Po wykonaniu tej operacji, należy w programie Code::Blocks nacisnąć prawym przyciskiem myszy na znajdującą się w zakładce „Projects” po lewej stronie nazwę utworzonego projektu (np. eStaurantProject), a następnie wybrać opcję „Add files...”. Należy wybrać wszystkie **pliki** znajdujące się w katalogu projektu, **pomijając foldery oraz plik z rozszerzeniem CBP** i potwierdzić przyciskiem „Otwórz”.



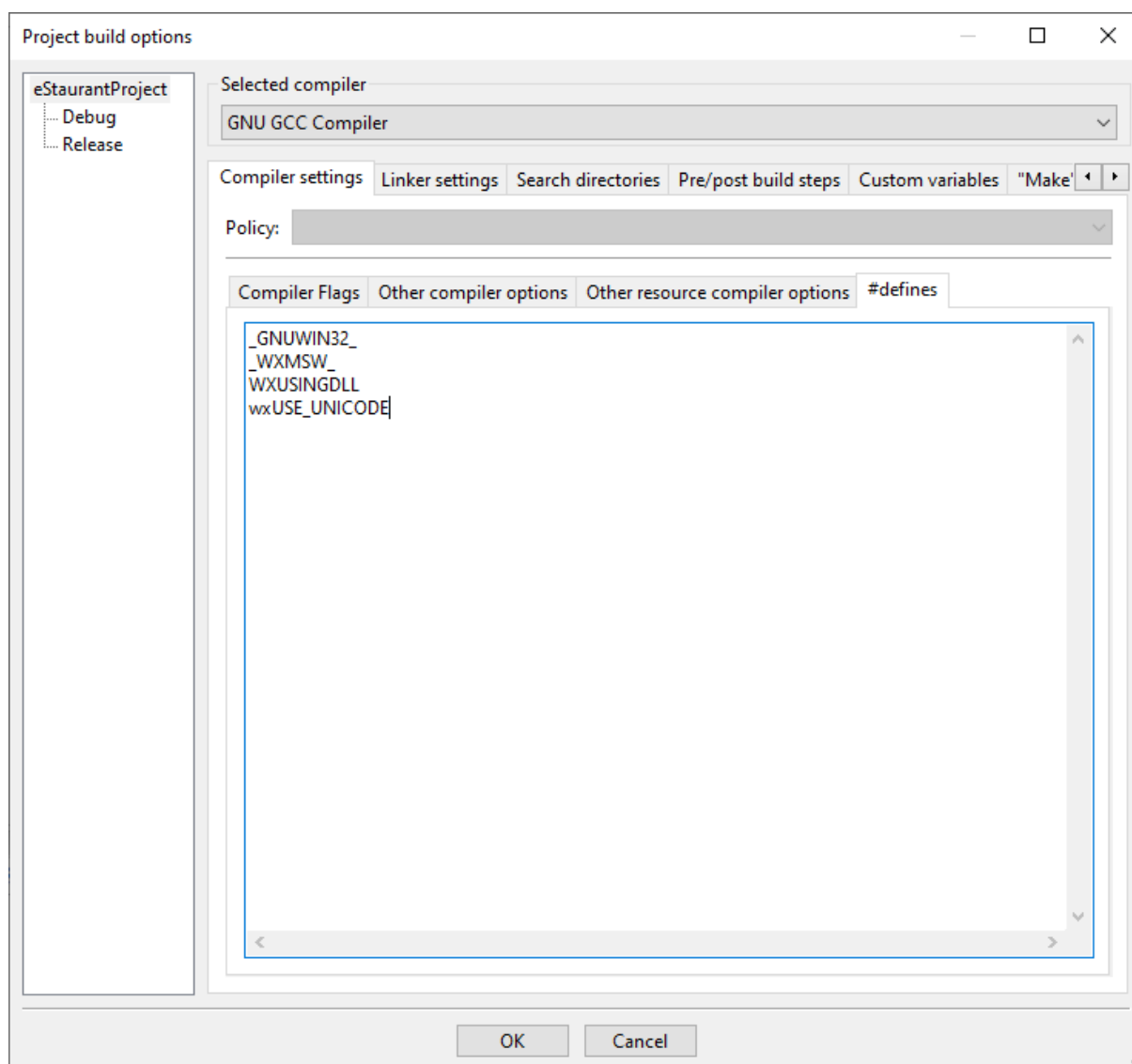
Zrzut 12 – dodawanie plików do projektu.

Jeśli pliki zostały pomyślnie dodane do projektu, należy przystąpić do jego konfiguracji. W tym celu ponownie należy kliknąć prawym przyciskiem myszy na nazwę projektu, a następnie wybrać opcję „Properties...”. Zostanie wyświetlone okno dialogowe „Project/targets options”. W zakładce „Build targets” należy w polu „Type:” wybrać opcję „GUI application”. Następnie, wracając do zakładki „Project settings” należy wybrać „Project’s build options...”, co skutkuje otwarciem kolejnego okna dialogowego. Dla wybranego projektu należy w zakładce „Compiler settings” wybrać podzakładkę „Other compiler options”. Należy wprowadzić następujące opcje:

```
-pipe
-mthreads
-finput-charset=utf-8
```

Po czym wybrać podzakładkę „#defines” i wprowadzić następujące wiersze:

```
_GNUWIN32_
_WXMSW_
WXUSINGDLL
wxUSE_UNICODE
```

Zrzut 13 – konfigurowanie opcji kompilacji projektu

Kolejnym krokiem jest udanie się do zakładki „Linker settings” i w polu „Link libraries:” należy podać ścieżki do plików znajdujących się w katalogu wxWidgets. Przykładowe ścieżki:

C:\Program Files\wxWidgets\lib\gcc_dll\libwxbase30ud.a

C:\Program Files\wxWidgets\lib\gcc_dll\libwxmsw30ud_adv.a

C:\Program Files\wxWidgets\lib\gcc_dll\libwxmsw30ud_core.a

C:\Program Files\wxWidgets\lib\gcc_dll\libwxpngd.a

C:\Program Files\wxWidgets\lib\gcc_dll\libwxzlibd.a

W polu „Other linker options:” należy natomiast wprowadzić polecenie:

-mthreads

Następnie należy wybrać zakładkę „Search directories” i w podzakładkach „Compiler” oraz „Resource compiler” podać tę samą ścieżkę do katalogu wxWidgets\include (np. „C:\Program Files\wxWidgets\include”).

Kolejnym etapem jest wprowadzenie zmian dla „Debug”. Po lewej stronie okna dialogowego, pod zaznaczoną dotychczas nazwą projektu, należy wybrać „Debug”. Następnie, po

przejściu do zakładki „Search directories” i w polach „Compiler” oraz „Resource compiler” podaje się ścieżki do katalogu `wxWidgets\lib\gcc_dll\mswud`, natomiast w polu „Linker” - ścieżkę do katalogu `wxWidgets\lib\gcc_dll`.

Po wprowadzeniu wszystkich zmian, należy zatwierdzić je przyciskiem „OK”, po czym można przystąpić do kompilacji projektu. Należy rozpocząć kompilację przyciskiem „Build” i jeśli projekt został skonfigurowany poprawnie a wszystkie niezbędne pliki zawarte zostały w jego katalogu, projekt powinien skompilować się bez błędów. Jeśli to nastąpiło, można uruchomić program.

7.3. Instrukcja obsługi programu

Program został zaprojektowany tak, aby był intuicyjny i prosty w obsłudze. Bezpośrednio po jego uruchomieniu plikiem `eStaurant.exe`, można przystąpić do pracy. Szczegółowy opis interfejsu okna głównego oraz okien dialogowych został zawarty w rozdziale 3.

Jeśli program został uruchomiony po raz pierwszy, należy w pierwszej kolejności utworzyć listę potraw. W tym celu należy wcisnąć przycisk „Menu” (skrót klawiszowy: Ctrl+M), dzięki któremu otwarte zostanie okno dialogowe umożliwiające utworzenie menu potraw.

Dodawanie potraw odbywa się w prosty sposób. W polu „Nazwa” należy wpisać nazwę potrawy, w polu „Opis” szczegóły dotyczące potrawy, pole „Gramatura (g)” należy wypełnić liczbą całkowitą oznaczającą wagę porcji potrawy mierzoną w gramach. Nie należy dopisywać jednostki, ponieważ program automatycznie doda na końcu „g”, jako skrót od „gram”; pole „Cena (PLN)” służy do podania ceny za porcję potrawy. Liczba ta może być całkowita lub zmiennoprzecinkowa, z tym że **należy pamiętać, aby część dziesiętną od całkowitej oddzielić kropką, a nie przecinkiem** (wpisanie przecinka spowoduje wystąpienie błędu podczas próby dodania potrawy do zamówienia). Ostatnim krokiem jest wybranie typu potrawy – do dyspozycji są następujące rodzaje: „Przystawki”, „Dania główne”, „Zupy”, „Desery”, „Dodatki”, „Napoje” oraz „Inne”. Aby móc dodać potrawę do listy, wszystkie pola muszą być wypełnione. Dodawanie zatwierdza się przyciskiem „Dodaj”. Można również wyczyścić pola wejściowe przyciskiem „Wyczyść” (pola są również automatycznie czyszczone po dodaniu nowej potrawy). Dodana potrawa powinna pojawić się na liście. Okno można zamknąć przyciskiem „Zamknij” lub „krzyżykiem” znajdującym się w prawym, górnym rogu okna. Dane na liście zostaną wtedy automatycznie zapisane do pliku i odczytane przy ponownym otwarciu tego okna dialogowego.

Możliwa jest również edycja potraw znajdujących się na liście – należy w tym celu zaznaczyć daną potrawę lewym przyciskiem myszy, po czym kliknąć prawy przycisk myszy i wybrać dowolną opcję modyfikacji, spośród 4 dostępnych (modyfikacja nazwy, opisu, gramatury oraz ceny). W podobny sposób można usunąć potrawę – wystarczy po wciśnięciu

prawego przycisku myszy na danej potrawie wybrać opcję „Usuń”, a następnie zatwierdzić przyciskiem „OK”.

Dodawanie nowych zamówień przebiega w sposób zbliżony do dodawania nowych potraw. Służy do tego panel „Nowe zamówienie”, umieszczony w prawej części głównego okna aplikacji. Jeśli lista klientów nie jest pusta, można wybrać klienta z bazy, naciskając na przycisk zaznaczenia obok frazy „Klient z listy” i wybierając danego klienta z rozwijanej listy. Po wybraniu, pola „Adres” oraz „Numer tel.” zostaną automatycznie wypełnione danymi dotyczącymi wybranego klienta. Jeśli natomiast nie zaznaczono opcji „Klient z listy”, należy ręcznie wprowadzić adres oraz numer telefonu klienta w odpowiednie pola. Ponadto należy wybrać opcję płatności z pola wyboru „Płatność” – dostępne opcje to: „Gotówka”, „Karta płatnicza”, „Przelew” oraz „Inny”. Następnie należy przejść do wyboru dań. Jeśli lista potraw nie jest pusta, to w polach „Danie nr 1”, „Danie nr 2”, ..., „Danie nr 6” pojawią się do wyboru dostępne potrawy. Wybranie pierwszej potrawy jest obowiązkowe, natomiast kolejne można dodawać do zamówienia przyciskiem plusa, znajdującym się obok każdego dania i wybierając potrawę z listy. Obok każdego pola wyboru dania, znajduje się także pole tekstowe z liczbą zamawianych porcji danej potrawy. Domyślnie wprowadzona wartość to 1. Przy wybieraniu kolejnych dań i modyfikowaniu ilości zamawianych porcji, pole „Kwota zamówienia” będzie automatycznie aktualizowane o łączną kwotę zamówienia, bazując na cenach ustalonych podczas dodawania potraw do listy „Menu”. Zamówienie można dodać tylko wtedy, kiedy wszystkie dostępne pola są wypełnione. Do zatwierdzania służy przycisk „Dodaj” (funkcjonuje identycznie do przycisku „Dodaj” z okna „Menu”), a do czyszczenia pól – przycisk „Wyczyść”. Jeśli poprawnie dodano zamówienie, licznik „Aktywne zamówienia” znajdujący się nad panelem „Nowe zamówienie” zostanie automatycznie zaktualizowany o liczbę zawartych w liście aktywnych zamówień.

Jeśli wykonano dane zamówienie, można zarchiwizować je przyciskiem „Archiwizuj”, znajdującym się pod panelem „Nowy klient”. Zamówienie zostanie usunięte z listy aktywnych zamówień oraz przeniesione do listy „Archiwum”, a licznik aktywnych zamówień zostanie pomniejszony o to zamówienie. Archiwum można przejrzeć wybierając przycisk „Archiwum” (skrót klawiszowy: Ctrl+N). Listę zarchiwizowanych zamówień można wyczyścić przyciskiem „Wyczyść” znajdującym się w oknie dialogowym „Archiwum”. Spowoduje to usunięcie wszystkich zarchiwizowanych zamówień z listy.

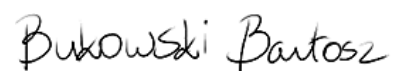
Dodawanie nowych klientów do bazy odbywa się w następujący sposób: Należy kliknąć lewym przyciskiem myszy na aktywne zamówienie, z adresem oraz numerem telefonu klienta, którego chcemy dodać, dzięki czemu pola wejściowe „Imię” oraz „Nazwisko” panelu „Nowy klient” staną się aktywne. Następnie należy wprowadzić imię oraz nazwisko dla nowego klienta w odpowiednie pola oraz zatwierdzić przyciskiem „Dodaj”, znajdującym się w tym panelu. Spowoduje to dodanie do listy klientów osoby o podanym imieniu i nazwisku, przypisując jej adres oraz numer telefonu na podstawie wybranego zamówienia. Listę

klientów można obejrzeć wybierając przycisk „Klienci” (skrót klawiszowy: Ctrl+K). Dane dotyczące każdego klienta można modyfikować, a także usuwać, w sposób identyczny jak w przypadku listy potraw.

Informacje dotyczące aplikacji oraz twórcy, dostępne są po wybraniu opcji „eStaurant – informacje” (skrót klawiszowy: Ctrl+I) w menu „Pomoc” znajdującym się w górnej części programu.

8. Oświadczenie o samodzielności wykonania projektu

Oświadczam, że projekt o tytule „eStaurant – program do obsługi zamówień dla lokali gastronomicznych” wraz z niniejszą dokumentacją, stanowiący podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu „Programowanie w języku C 2” został wykonany przeze mnie samodzielnie.



podpis

Bartosz Bukowski gr. 2ID11B

Nr albumu 090088