



POLITECHNIKA ŚLĄSKA

WYDZIAŁ ELEKTRYCZNY

Katedra Energoelektroniki, Napędu Elektrycznego i Robotyki

PROJEKT INŻYNIERSKI

Aplikacja służąca do generowania harmonogramów pracy

Application used to generate work schedules

Student: **Bartosz Ireneusz GIEROŃ**
Nr albumu: 265649
Studia: Stacjonarne I stopnia
Kierunek: Informatyka
Specjalność: Informatyka w systemach elektrycznych
Prowadzący: dr inż. Maciej SAJKOWSKI

GLIWICE 2020

SPIS TREŚCI

1. WSTĘP	2
2. CEL I ZAKRES PROJEKTU	3
2.1 Cel projektu	3
2.2 Zakres projektu	3
2.3 Założenia	3
3. OPIS PROGRAMU	4
3.1 Zasada działania programu	4
3.2 Schematy blokowe działania programu.	12
4. DOKUMENTACJA TECHNICZNA	14
4.1 Obiektowa architektura projektu	14
4.2 Schematy klas UML	18
4.3 Algorytmy	20
4.4 Zabezpieczenia	25
5. PODSUMOWANIE	27
5.1 Minimalne wymagania sprzętowe	27
5.2 Instalacja	27
5.3 Podsumowanie	28
5.4 Wnioski	28
6. LITERATURA	30
7. ŹRÓDŁA	30

1. WSTĘP

Przedmiotem pracy jest program tworzący harmonogramy pracy. Program pozwala tworzyć miesięczne harmonogramy dla wybranego stanowiska. Ponadto program pozwala na tworzenie, edytowanie i usuwanie stanowisk, pracowników i zasobów.

Poza możliwościami pozwalającymi tworzyć harmonogram, program oferuje narzędzia do przechowywania szczegółowych danych o zasobach, stanowiskach i pracownikach w firmie.

Inspiracją do wykonania programu była dorywcza praca zawodowa w małej firmie z potrzebą ciągłości zapewnienia stanowisk pracy i wynikające z tego komplikacje przy tworzeniu harmonogramu. Program, który jest przedmiotem tej pracy został stworzony z myślą automatyzacji tego powszechnego problemu.

Prowadzenie starań do realizacji zagadnienia niesie ze sobą wiele rozwinięć problemu jakim jest zarządzanie zasobami ludzkimi i materialnymi w obrębie firmy. Rozwiązania jakie znajdują się w aplikacji mają sprawić łatwą i intuicyjną obsługę programu. Niniejsza opis jest stworzona na podstawie w pełni funkcjonującej aplikacji Windows Forms. Wszelkie opisy oraz schematy zostały zaprojektowane z myślą przybliżenia czytelnikowi obiektu pracy.

2. CEL I ZAKRES PROJEKTU

2.1 Cel projektu

Celem projektu jest stworzenie działającego programu i opisu jego pracy.

Motyacją do opracowania tego programu jest potrzeba do tworzenia oprogramowania ułatwiającego organizację pracy przy pomocy dostępnych rozwiązań informatycznych.

Końcowym efektem realizacji pracy powinien być program, przy użyciu którego będą tworzone harmonogramy na wybrany miesiąc. Program będzie umożliwiał przechowywanie informacji o zasobach, stanowiskach pracy i pracownikach.

2.2 Zakres projektu

Zakres projektu obejmuje opracowanie algorytmu tworzącego harmonogramy, stworzenie aplikacji okienkowej dla systemów Windows, pozwalającej na intuicyjną obsługę, zaprojektowanie struktury programu, implementacji algorytmu w języku C# oraz zapytań w języku SQL przy pomocy środowiska Visual Studio, testy programu oraz wizualizację jego działania.

2.3 Założenia

Założono, że program powstanie przy pomocy programu Microsoft Visual Studio 2019 w wersji Community. Interfejs graficzny będzie zrealizowany w Windows Forms, a zaprogramowany zostanie w języku C#.

Ponadto przyjęto założenie, że program będzie realizował następujące działania użytkownika:

- ❖ dodawanie, usuwanie i edytowanie pracowników, stanowisk i zasobów oraz przechowywanie ich w lokalnej bazie danych,
- ❖ tworzenie harmonogramów na podstawie jednej przyjętej metodyki i ich przechowywanie w lokalnej bazie danych wraz z możliwością ich usunięcia.

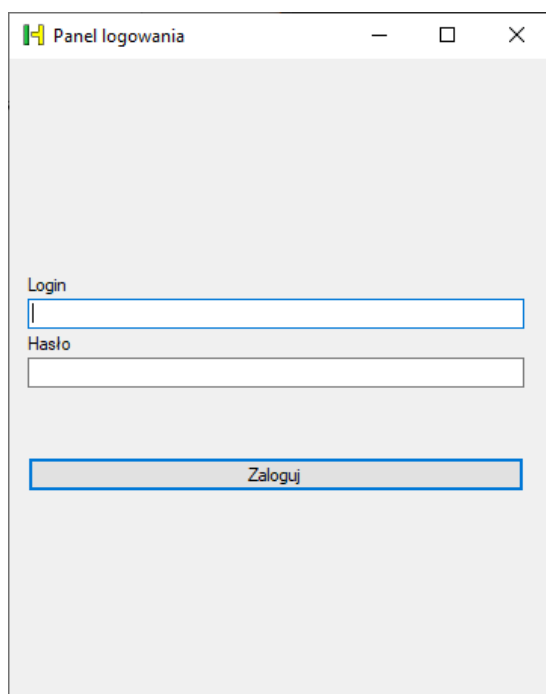
3. OPIS PROGRAMU

W tym rozdziale opisano wygląd i zasady działania programu od strony użytkownika, oraz opisy i schematy dotyczące spójności interfejsu z oprogramowaniem.

3.1 Zasada działania programu

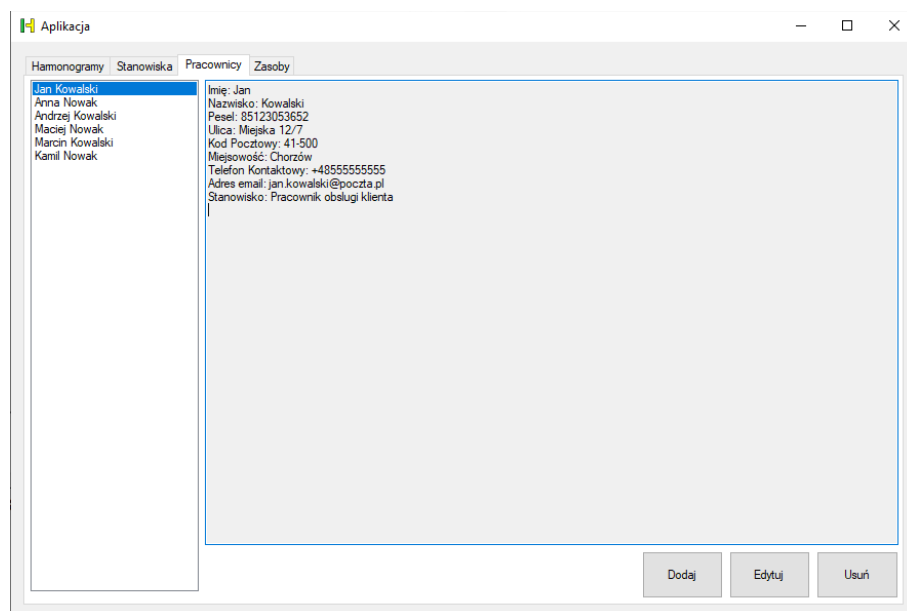
Po uruchomieniu programu ukazuje się Panel Logowania (Rysunek 3.1.1), jak sama nazwa wskazuje jest to okno w którym należy się zalogować aby przejść do następnego ekranu. Dla potrzeb projektu login i hasło zostały ustawione następująco:

- ❖ LOGIN: admin
- ❖ HASŁO: pass

The image shows a screenshot of a Windows-style application window titled "Panel logowania". The window has a standard title bar with minimize, maximize, and close buttons. The main area of the window is light gray. It contains two text input fields: the first is labeled "Login" and the second is labeled "Hasło". Below these fields is a button labeled "Zaloguj".

rys 3.1.1 Panel logowania aplikacji.

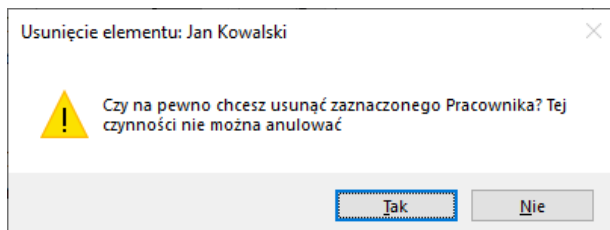
Po prawidłowym wprowadzeniu loginu i hasła wyświetlone zostaje menu główne aplikacji, które jest złożone z 4 widoków oferujących przegląd po elementach umieszczonych w bazie danych. Rys.3.1.2 przedstawia jedną z zakładek widoku głównego aplikacji, czyli „Pracownicy”.



rys.3.1.2: Główny widok aplikacji (widok zakładki „Pracownicy”).

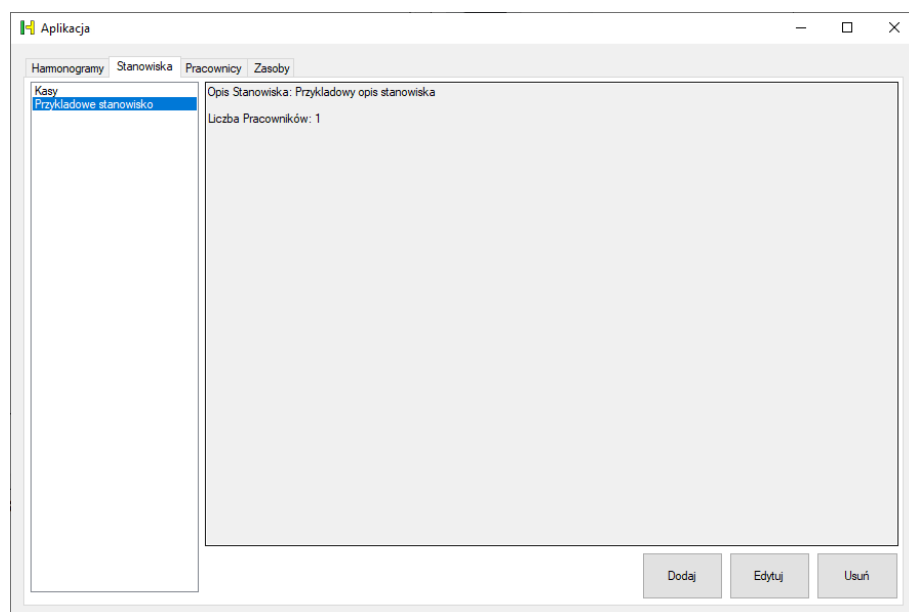
Każdy z widoków posiada przyciski pozwalające na akcje w obrębie danego widoku:

- ❖ Przycisk „Dodaj”, który znajduje się na każdym widoku otwiera nowe okno, w którym można dodać konkretny element (w zależności, z którego widoku został uruchomiony) za pomocą odpowiedniego formularza,
- ❖ Przycisk „Edytuj”, znajduje się w 3 widokach: „Stanowiska”, „Pracownicy”, „Zasoby” i pozwala na załadowanie danych z powrotem do okna z formularzem i edycję danych,
- ❖ Przycisk „Usuń”, znajduje się w każdym widoku i pozwala na usunięcie wybranego elementu z bazy danych uprzedzając użytkownika o nieodwracalności wykonanej akcji (rys.3.1.3).



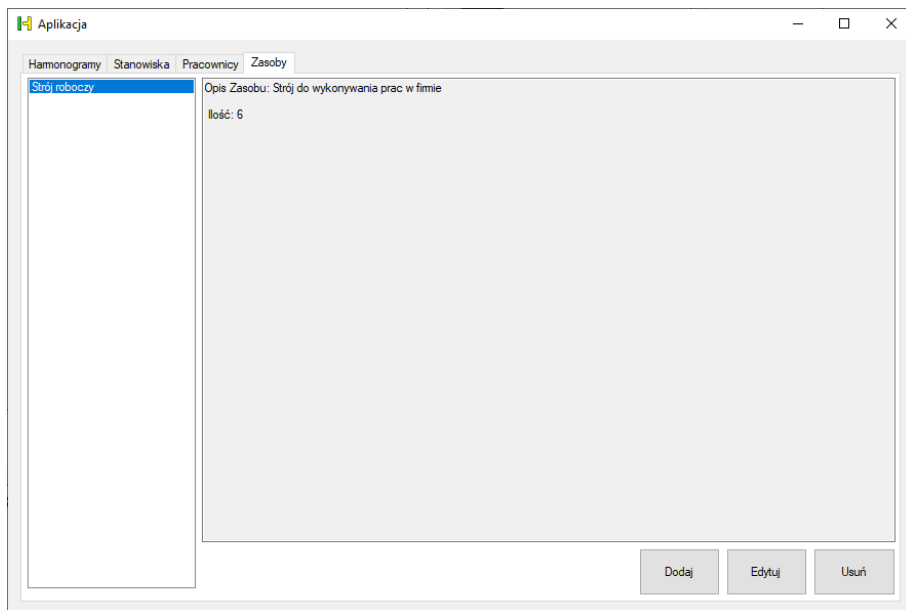
rys.3.1.3 Okno informacyjne wywołane przyciskiem “Usuń”.

Wszystkie widoki oferują podgląd danych w następującym porządku: po lewej znajduje się lista elementów, a po prawej okno podglądu aktualnie zaznaczonego elementu tak jak na rys.3.1.2.



rys.3.1.4 Podgląd wybranego z listy Stanowiska w zakładce “Stanowiska”.

Na rysunkach 3.1.4 i 3.1.5 przedstawiono podobny sposób wyświetlania szczegółowych danych na temat aktualnie zaznaczonego elementu listy.



rys.3.1.5: Podgląd wybranego z listy Zasobu w zakładce “Zasoby”.

W zakładce “Harmonogramy” (rys.3.1.6) podgląd elementu z listy realizowany jest przy pomocy wyświetlanej tabeli, w której kolumny to Imię i Nazwisko i kolejne dni miesiąca którego dotyczy harmonogram. W kolejnych wierszach prezentowani są kolejni pracownicy przypisani do konkretnego harmonogramu oraz symboliczny opis zmian dopasowany do dni miesiąca. Symbole reprezentują:

- ❖ N - nocna zmiana,
- ❖ R - poranna zmiana,
- ❖ P - popołudniowa zmiana,
- ❖ U - urlop,
- ❖ Sb - sobota,
- ❖ Nd - niedziela.

Aplikacja

Hamonogramy Stanowiska Pracownicy Zasoby

2775230_2020_Syczen

Nazwisko	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Jan Kowalski	R	R	R	Sb	Nd	P	P	P	P	P	Sb	Nd	N	N	N	N
Anna Nowak	N	N	N	Sb	Nd	R	R	R	R	R	Sb	Nd	P	P	P	P
Marcin Kowalski	N	N	N	Sb	Nd	R	R	R	R	R	Sb	Nd	P	P	P	P
Maciej Nowak	R	R	R	Sb	Nd	P	P	P	P	P	Sb	Nd	N	N	N	N
Andrzej Kowalski	P	P	P	Sb	Nd	N	N	N	N	N	Sb	Nd	R	R	R	R
Kamil Nowak	P	P	P	Sb	Nd	N	N	N	N	N	Sb	Nd	R	R	R	R

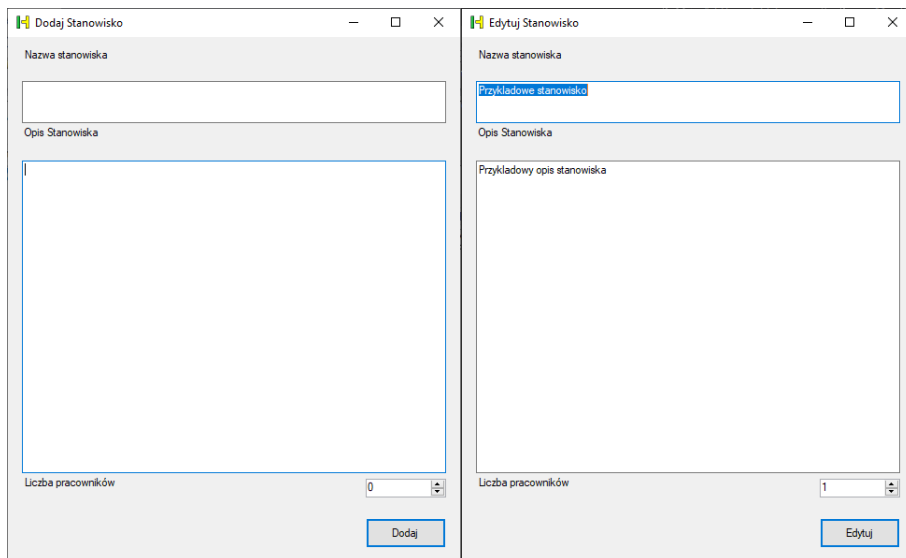
Dodaj Usun

rys.3.1.6 Podgląd Harmonogramu reprezentowanego tabelą w zakładce “Harmonogramy”.

Formularze uruchamiane przy pomocy przycisków “Dodaj” i “Edytuj” w przypadkach zakładek, które posiadają obie z nich, bazują na tym samym widoku, który można zaobserwować na rysunku 3.1.7. Zmianie podlega jedynie opis okna oraz przycisku, co można również zauważyć na rysunku 3.1.7. Okna służące do dodawania zasobu i stanowiska wyglądają podobnie, ponieważ ustalono dla tych wartości tylko 3 dane:

- ❖ w przypadku Stanowiska są to Nazwa Stanowiska, Opis Stanowiska i Liczba pracowników,
- ❖ w przypadku Zasobu są to Nazwa Zasobu, Opis Zasobu i Ilość Zasobu.

Przyjęto, że taka ilość informacji wystarczy do potrzeb realizowanego zadania.



rys.3.1.7 Porównanie okien dodawania i edycji stanowiska.

Okno “Dodaj Pracownika” (rys.3.1.8) wyróżnia się znaczną ilością danych do wprowadzenia.

The image shows a single window titled 'Dodaj Pracownika'. It contains several input fields for employee data: 'Imię' (First Name), 'Nazwisko' (Last Name), 'Pesel', 'Ulica i numer domu' (Street and house number), 'Kod pocztowy' (Postal code), 'Miejscowość' (Location), 'Telefon Kontaktowy' (Contact phone), 'email', and 'Stanowisko' (Position). A 'Dodaj' button is located at the bottom right of the window.

rys.3.1.8 Reprezentacja okna “Dodaj Pracownika”.

Podczas tworzenia programu przyjęto założenie, że za jego pomocą można przechowywać szczegółowe dane na temat Pracowników, które nie mają znaczenia przy dalszym działaniu programu, ale są przechowywane. Edytowanie i podgląd tych danych wypełnia kolejną funkcję programu.

Ostatnim omawianym oknem jest okno “Utwórz Harmonogram” przedstawione na rysunku 3.1.9, wybierane z zakładki “Harmonogramy” przyciskiem “Dodaj”.

rys.3.1.9 Widok okna “Utwórz Harmonogram”.

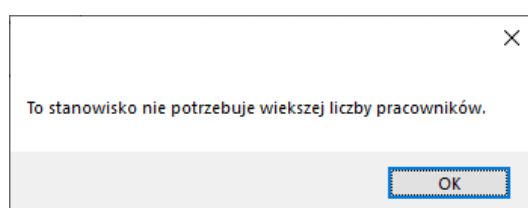
Użytkowanie formularza tworzącego Harmonogram jest w pewnym sensie proceduralne. Na początku należy wybrać miesiąc i rok, którego ma dotyczyć tworzony harmonogram. Potem odblokowany zostanie wybór stanowiska. Kiedy miesiąc, rok i stanowisko będą wybrane można przejść do zablokowanej części programu służącej do dodawania poszczególnych pracowników.

Aby dodać pracownika należy wybrać go z listy po lewej stronie (patrz rys.3.1.9), następnie opcjonalnie można wybrać zasób, który chcemy dodać do danego pracownika (można wybrać tylko jeden) oraz również opcjonalnie dni urlopu w formacie liczbowym odnoszącym się do dnia miesiąca, które będą oddzielone przecinkami. Następnie należy użyć przycisku “Dodaj Pracownika”.

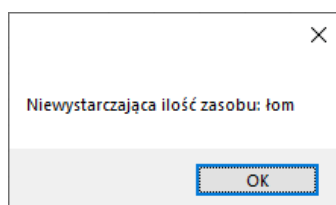
Wybrany pracownik pojawi się na tabeli z prawej wraz z wygenerowanymi zmianami i zniknie z listy po lewej.

W polu “Id Harmonogramu” można podać Id które zostanie wykorzystane przy tworzeniu nazwy harmonogramu w formacie [Id_rok_miesiąc]. Jeżeli tego pola nie zmienimy to Id zostanie wygenerowane z generatora liczb pseudolosowych.

Dla potrzeb projektu przyjęto 8-godzinny tryb 3-zmianowy pracujący od poniedziałku do piątku przez całą dobę. Czyli aby obsadzić na przykład 2 osobowe stanowisko należy do harmonogramu dodać maximum 6 osób. Ostrzeżenie że w harmonogramie jest już wystarczająca ilość osób, które pokazano na rys.3.1.10, pojawi się gdy mimo widniejącego kompletu osób dla stanowiska podjęto próby dodania kolejnego.



rys.3.1.10 Okno informujące o próbie dodania zbyt dużej liczby pracowników.



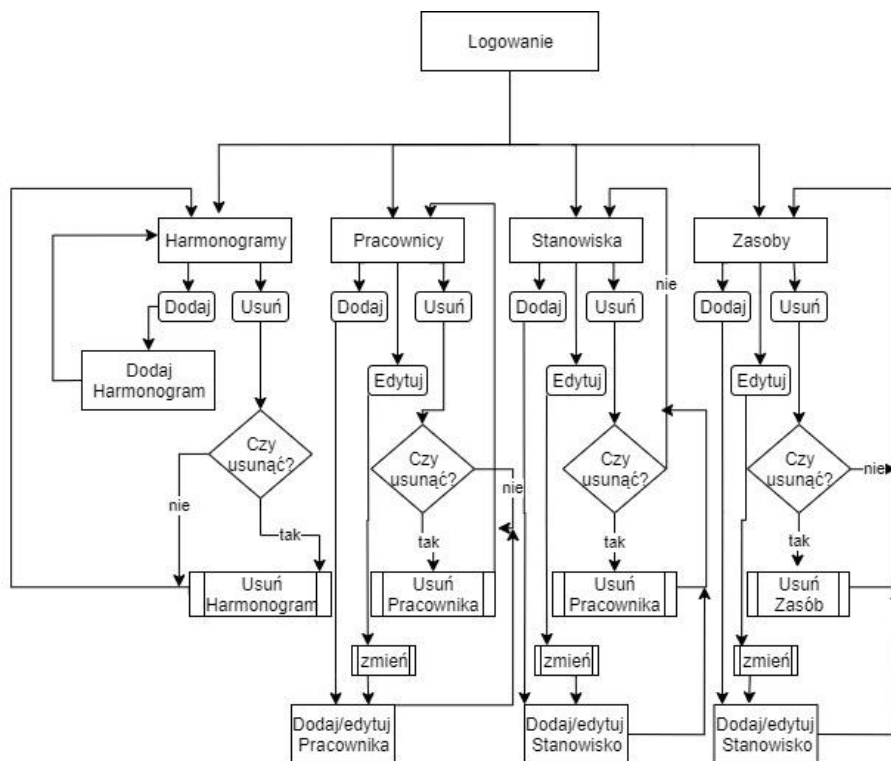
rys.3.1.11 Okno informujące o przekroczeniu ilości próśb o zasób.

Na rys.3.1.11 przedstawiono komunikat, który pojawia się, gdy do pracownika próbowano dodać zasób, którego ilość na to nie pozwala.

Przycisk “Dodaj Harmonogram” zamyka dotychczasowe okno i wygenerowany harmonogram dodaje do bazy. Gotowy harmonogram można przeglądać w zakładce “Harmonogramy” w głównym oknie aplikacji.

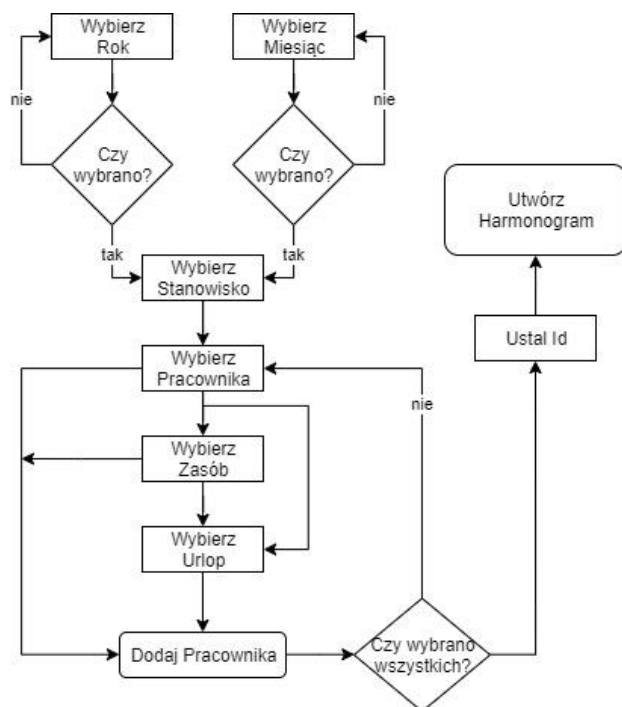
3.2 Schematy blokowe działania programu.

Aby zobrazować działanie programu stworzono schematy blokowe. Schematy zostały przygotowane za pośrednictwem portalu <https://www.draw.io>.



rys.3.2.1 Schemat blokowy działania programu.

Na schemacie przedstawionym na rys.3.2.1 pokazano połączenie okien i decyzji użytkownika przekazywanych przez okna dialogowe (wyboru) oraz przyciski dostępne z poszczególnych części programu. Na rys.3.2.1 można również wyszczególnić 4 zakładki (Harmonogramy, Zasoby, Pracownicy, Stanowiska) a działania na nich przeprowadzone odnoszą się bezpośrednio do działań przeprowadzanych na bieżąco na bazie danych posiadającej przypisane do nich tabele o takich samych nazwach.



rys.3.2.2 Schemat blokowy działania okna dodawania harmonogramu.

Rys.3.2.2 opracowano, aby uporządkować opis proceduralny dodawania harmonogramu. W tym schemacie prostokątne bloki reprezentują kolejne wybory z list rozwijanych, pól tekstowych i numerycznych.

4. DOKUMENTACJA TECHNICZNA

W tym rozdziale opisano klasy i ich udział w programie, przedstawiono mechanizmy w tym algorytmy stworzone do obsługi wymaganych zdarzeń oraz zabezpieczenia aplikacji przed skutkami działań niepożądanych.

4.1 Obiektowa architektura projektu

LogForm

Klasa odpowiadająca za reprezentację i implementację procesu logowania.

Najważniejszym elementem tej klasy jest metoda *LoginBtnClick* która przekazuje dane wprowadzone z pól *passField* i *loginField* do bazy danych i sprawdza czy się w niej znajdują. Przejście do głównej części programu jest możliwe tylko z tej klasy i tylko po podaniu właściwego loginu i hasła.

MainView

Główna klasa, która łączy wszystkie pozostałe w spójny program.

W metodzie *MainView_Load* definiowane są listy (*pesels*, *resources*, *stations*, *tables*, *workers*), które następnie wypełniane są danymi z bazy danych i przekazywane do kontrolerek *ListBox* umieszczonych w zakładkach widoku głównego aplikacji.

Wszystkie kontrolki *ListBox* posiadają metody, które wywoływane są podczas zmiany zaznaczonego w nich elementu. Na przykład metoda *ListBoxWorkers_SelectedIndexChanged* przekazuje do bazy danych zapytanie o szczegółowe dane zaznaczonego elementu i wpisuje je w odpowiednim formacie do kontrolki odpowiedzialnej za reprezentację tych danych, pozostałe metody kończące się na *SelectedIndexChanged* działają analogicznie.

Klasa *MainView* zawiera 11 metod, które są wywołane przez przyciski znajdujące się w oknie głównym programu, wszystkie kończą się słowem *Click* i dzielą się na trzy kategorie:

- ❖ *Add* - metody te uruchamiają okna dodawania elementu,
- ❖ *Delete* - metody, które usuwają kolejny element po uprzednim upewnieniu się czy użytkownik zdaje sobie sprawę o swoim czynnie,
- ❖ *Edit* - metody, które uruchamiają okna dodawania elementu, ale zmieniając je tak, aby było opisane jako okno edycji, poza tym metody te przekazują wartości aktualnie zaznaczonego elementu do konstruktora formularzy edycyjnych.

W tej klasie znajduje się również metoda *MainView_FormClosing*, która jest wywoływana przez akcję zamykania głównego okna i ma za zadanie zamknąć wszystkie pozostałe okna (jeżeli istnieją). Jest to forma zabezpieczenia program przed utraceniem spójności, ponieważ okno główne łączy wszystkie okna w całość programu.

AddResourcesForm

Ta klasa reprezentuje dwie funkcje: dodawania i edytowania zasobu zmieniająca swoje przeznaczenie w zależności od pola *IsForEditing* typu bool, które jest ustawione w zależności od przesłanych do konstruktora parametrów. Jeżeli przesłano do konstruktora dane zasobu (zadaniem ich zmiany) to tytuł okna oraz napis na przycisku zostaną zmienione na odpowiednio "Edytuj" w miejscu "Dodaj". Zmieni się również zastosowanie metody *AddResourcesBtnClick*, która domyślnie uruchamia procedurę dodawania zasobu do odpowiedniej tabeli w bazie danych. Ta metoda zamiast tego uruchomi procedurę odszukania danych korzystając z przekazanego do konstruktora pola *Id* oraz ich edycję.

AddStatForm

Klasa działająca bardzo podobnie do klasy *AddResourcesForm*. W niej również zaprogramowano metodę *AddStationButton* w sposób żeby służyła dwóm celom w tym przypadku: dodawaniu i edytowaniu stanowiska, a mechanizm jest taki sam. Klasy te różnią się widokiem okna.

AddWorkerForm

W tej klasie tak jak w poprzednich dwóch zastosowano rozwiązanie zapewniające podwójne zastosowanie, w tym przypadku: dodawanie i edytowanie pracownika. Oprócz tego znajdują się tu metody, które usprawniają wpisywanie danych. Metoda *OnlyDigit* pobiera dane z kontrolki typu *TextBox* i sprawia, że w niej można wpisywać tylko liczby. Metoda *TextBoxCode_TextChanged* pozwala na wprowadzenie tylko cyfr i myślnika w polu *TextBoxCode*. Metoda *TextBoxPesel_TextChanged* używa bezpośrednio metody *OnlyDigit* przekazując do niej wartość pola tekstowego *TextBoxPesel*. Podobnie do poprzedniej metody działa *TextBoxPhone_TextChanged* z wyjątkiem, że dodano fragment kodu, który umożliwia użytkownikowi na wprowadzenie symbolu "+", mogącego być elementem numeru telefonu.

FormIntoDatabase

Jest to jedna z dwóch (nie licząc głównej klasy *Program*) klas w tym projekcie które nie są klasami typu Windows Forms a służą ułatwieniu pracy. Ta konkretna klasa została opracowana w celu ułatwienia przekazywania danych z formularzy do bazy danych. Posiada konstruktor, który przyjmuje dwa parametry:

- ❖ *sqlQuery* - do tego pola typu string należy wpisać odpowiednio zmodyfikowane polecenie w języku SQL, które w miejscach których normalnie znajdują się parametry są wpisane symbole "\$",

- ❖ *values* - jest to tablica elementów string, których zamierzano użyć w poleceniu języka SQL, powinna być takiej ilości ile symboli “\$” jest w *sqlQuery*.

Pole *valuesNames* to tablica nazw generowana w konstruktorze przy pomocy metody *CreateValuesNames*. Nazwy te generowane są jako kolejne liczby naturalne poprzedzone symbolem “@”. Taka konstrukcja nazw w ilości odpowiadającej ilości elementów tablicy *values* jest potrzebna do kolejnej metody *InsertNamesIntoQuery*, która w każde miejsce “\$” wstawi kolejną wygenerowaną nazwę z tabeli *valuesNames*. Taka konstrukcja wyrażenia oraz kolekcja parametrów pozwala na użycie publicznej metody *Query* na obiekcie typu *FormIntoDatabase*. Metoda *Query* łączy się z bazą danych i w sposób bezpieczny przekazuje parametry do zapytania, w ten sposób program jest zabezpieczony przed niepożądanym skutkiem wprowadzenia szkodliwych dla zapytania symboli.

DataBaseHelper

Jest to klasa statyczna udostępniająca 3 statyczne metody:

- ❖ *ConnectionString* - metoda zwraca obiekt typu string potrzebny do połączenia z bazą danych,
- ❖ *DeleteRow* - przyjmuje 3 parametry, czyli kolumnę, tabelę i daną. Ma za zadanie usunąć wiersz w podanej tabeli. Odpowiedni wiersz jest odszukiwany na podstawie podanej danej w podanej kolumnie
- ❖ *FillDataset* - przyjmuje zmienną typu string *DataBaseQuery* i opcjonalnie nazwę obiektu typu string *name* i obiekt *obj*. Metoda ta zwraca obiekt typu *DataSet* zapełniony danymi otrzymanymi z bazy danych na odpowiedź zapytania *DataBaseQuery*, do którego w sposób bezpieczny jest wstawiany opcjonalny obiekt *obj*.

AddTimeTableForm

Klasa ta służy do tworzenia i dodawania do bazy danych harmonogramu pracy i jest najważniejszym pod względem użyteczności elementem programu.

Znajdują się w niej pola:

- ❖ 4 listy typu string (*workers, pesels, stations, resources*) i jedna typu int (*resourcesCount*), które służą do przechowywania informacji pobranych z bazy danych.
- ❖ 2 listy typu int (*sundays, saturdays*) służące do przechowywania numerów dni w miesiącu wypadających w soboty i niedziele.
- ❖ 3 obiekty typu int (*year, month, days*) które kolejno przechowują informacje na temat aktualnie wybranego roku, miesiąca i liczby dni w tymże miesiącu danego roku.

- ❖ dwuwymiarowa tablica obiektów typu string *shiftsTable*, która przechowuje symbole zmian w liczbie potrzebnych pracowników zawartych w zmiennej typu int *workersNeeded*.
- ❖ zmienna robocza typu int *workerNeededCount* służąca do operacji, w których potrzebny jest odnośnik do pozostałej ilości potrzebnych pracowników do umieszczenia w danym harmonogramie, co jest uwarunkowane wybranym stanowiskiem.

Zdefiniowana w tej klasie metoda *AddTimeTableDorm_Load* odpowiada na zdarzenie załadowania okna i jej zadanie to wypełnienie list danymi z bazy danych, oraz przekazanie imion, nazwisk, stanowisk i zasobów do odpowiednich miejsc w oknie reprezentowanych na rys.3.1.9.

Metoda *ComboBox_year_SelectedIndexChanged* reaguje na dwa zdarzenia: wybór roku i wybór miesiąca i wykonywana jest tylko wtedy gdy i rok i miesiąc są uzupełnione. W tej metodzie zaprogramowano dodawanie kolumn do tabeli widocznej w kontrolce *listView*. Ich ilość oraz numeracja odpowiadają dniom wybranego miesiąca w danym roku. Pierwsza kolumna zawsze posiada opis "Nazwisko" i jest zauważalnie większa od pozostałych. Metoda ta również uzupełnia listy *sundays* i *saturdays* przy pomocy metody *FindDayOfWeek*.

FindDayOfWeek to metoda, która przyjmuje rok, miesiąc, ilość w nim dni oraz dzień, który ma być znaleziony oraz zwraca listę poszukiwanych dni.

W metodzie *ComboBox_station_SelectedIndexChanged* reagującej na zmianę wybranego stanowiska ustawiono zmienną *workersNeeded* na trzykrotność liczby pracowników pobranej z bazy danych (zasadność tej trzykrotności opisano na stronie 13) oraz uzupełnia tabelę *shiftsTable* za pomocą metody *CreateShifts*.

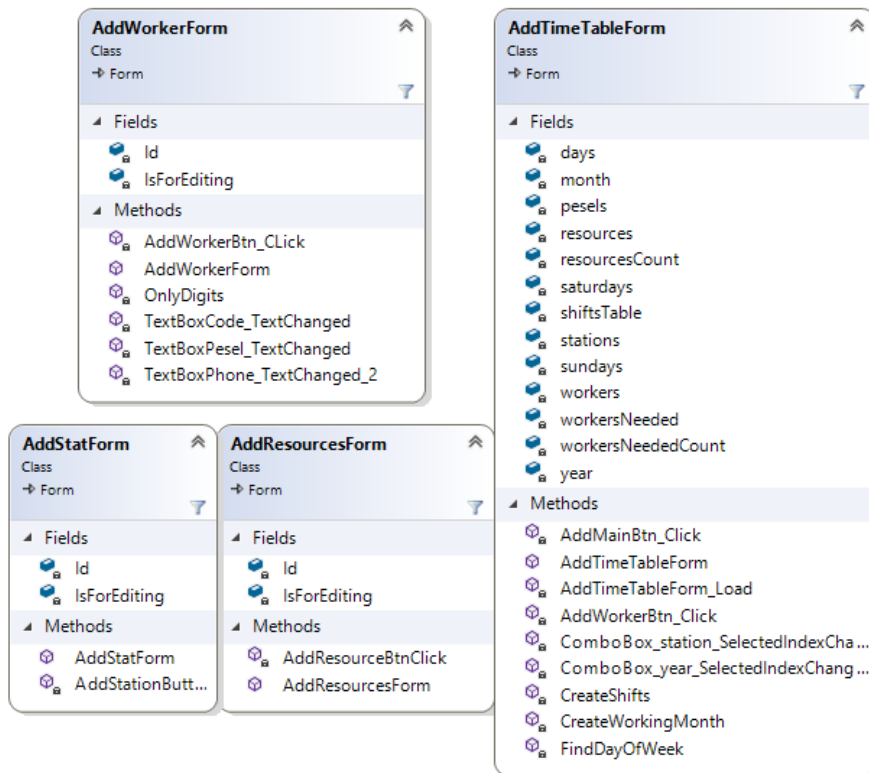
CreateShifts jest metodą stworzoną do wypełniania dwuwymiarowej tabeli *shifts* symbolami zmian w ilości związanej ze zmienną *workersNeeded*.

Metodą, która odpowiada za stworzenie listy zmian dla danego pracownika dopasowanej do danego miesiąca jest metoda *CreateWorkingMonth*. Jest ona wykorzystywana przy każdym wywołaniu metody *AddWorkerBtn_Click*, która z kolei wywoływana jest akcją przycisku dodawania pracownika. Poza tym Metoda *AddWorkerBtn_Click* wykonuje szereg operacji związanych ze sprawdzaniem wystarczającej liczby pracowników, obsługi dni wolnych od pracy oraz zasobów.

Ostatnią metodą w tej klasie jest metoda *AddMainBtn_Click* kończąca wszystkie poprzednie działania zwracając je tworzeniem harmonogramu na podstawie ustawionych danych i zapisaniu go w bazie danych. Jeżeli wszystko przebiegło zgodnie z wytycznymi to powodzenie tej funkcji kończy się zamknięciem aktualnego okna.

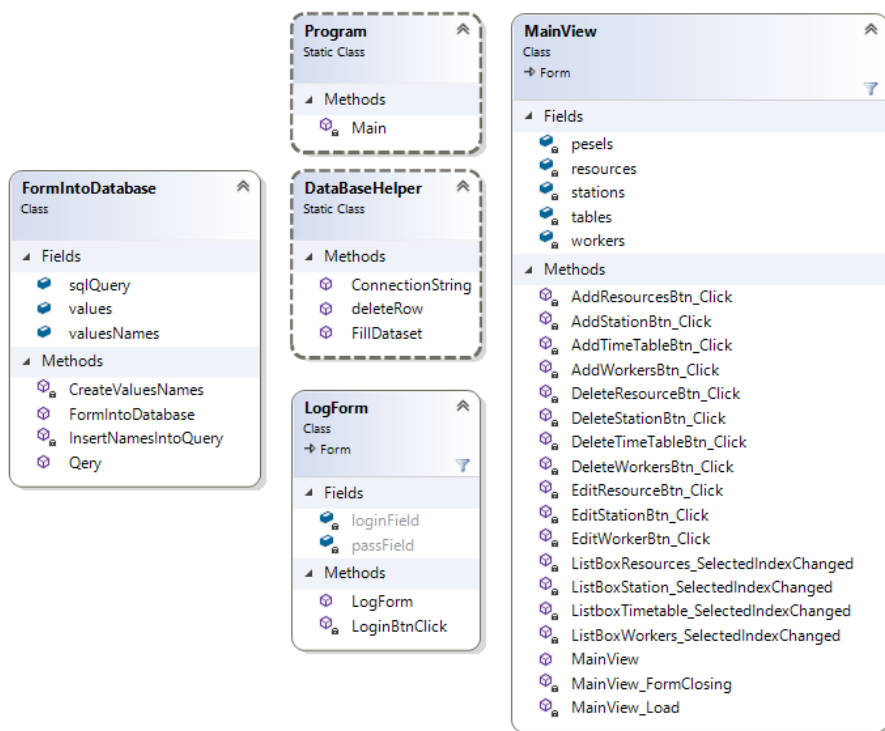
4.2 Schematy klas UML

Schematy klas zawartych w tym projekcie zostały wygenerowane przy pomocy rozszerzenia programu Visual Studio 2019 o nazwie Class Designer.



rys.4.2.1 Część 1 schematu klas zawierający metody dodawania elementu.

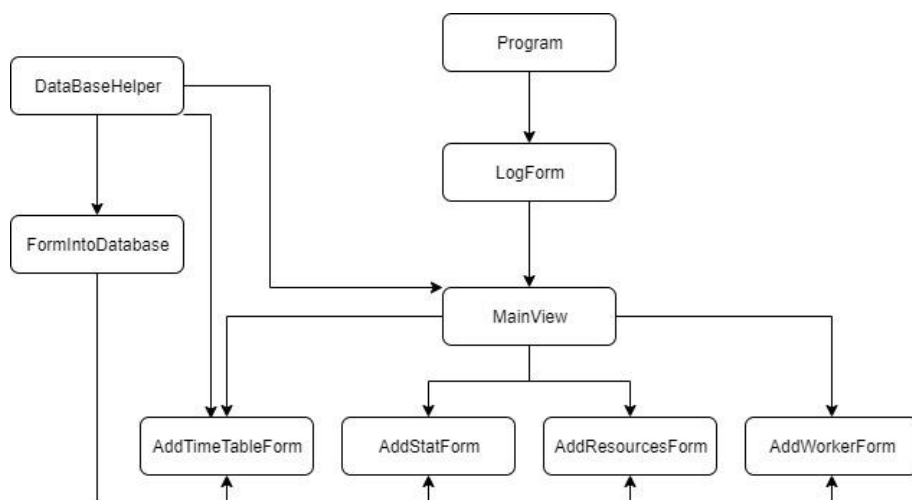
Na rys.4.2.1 przedstawiono fragment schematu klas, który podzielono na 2 części, aby ułatwić jego czytelność. Na tym umieszczono zgrupowane klasy Windows Forms o podobnym przeznaczeniu, mianowicie wszystkie one otwierają okna dodawania (lub edycji w przypadku klas *AddWorkerForm*, *AddStatForm* i *AddResourcesForm*). Należy zaznaczyć, że oprócz opisanych zawartości kodu tych klas (przybliżonych w rozdziale 4.1) każda klasa Windows Form posiada starannie stworzony widok oraz akcje łączące stronę kodu ze stroną interfejsu aplikacji.



rys.4.2.2 Część 2 Schematu klas przedstawiająca pozostałe klasy programu.

Klasę *MainView*, oraz klasę Logowania *LogForm* typu Windows Form wraz z klasami pomocniczymi *FormIntoDatabase* i *DataBaseHelper* (klasa statyczna) oraz klasa *Program*, która jest użyta do wywołania całości aplikacji zestawiono w rys.4.2.2.

W programie nie zastosowano mechanizmów dziedziczenia, więc schematy na rys.4.2.1 i rys.4.2.2 nie ukazują połączeń między klasami a jedynie skróconą reprezentację ich zawartości. Na rysunku 4.2.3 zilustrowano relacje, jakie zachodzą między klasami zawartymi w projekcie.



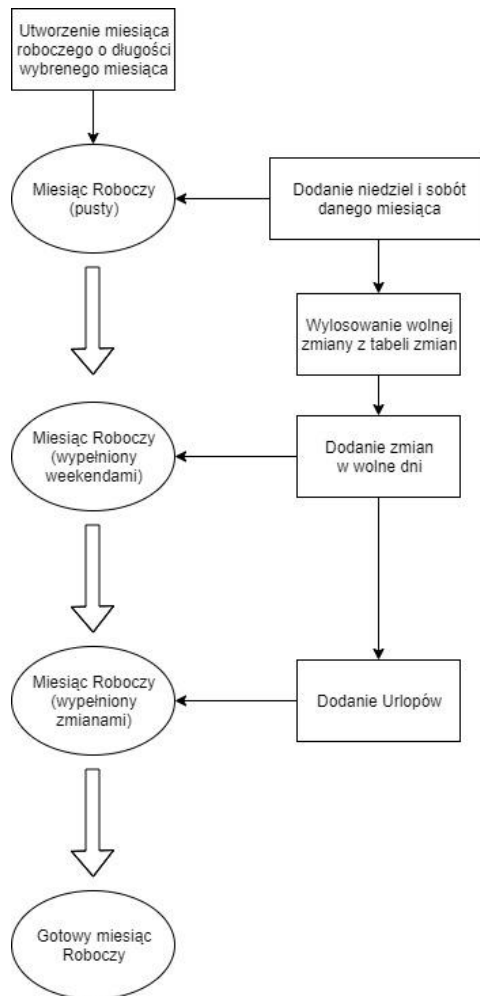
rys.4.2.3 Schemat relacji klas w projekcie.

4.3 Algorytmy

Algorytmem najważniejszym ze względu na jego zadanie w projekcie jest algorytm układania harmonogramu. Algorytm ten przewija się przez kilka metod klasy *AddTimeTableForm* i uznano, że jego działanie zostanie objaśnione w osobnym rozdziale.

Przedstawiony schemat na rys.4.3.1 obrazuje etapy powstawania gotowego miesiąca roboczego dla jednego pracownika.

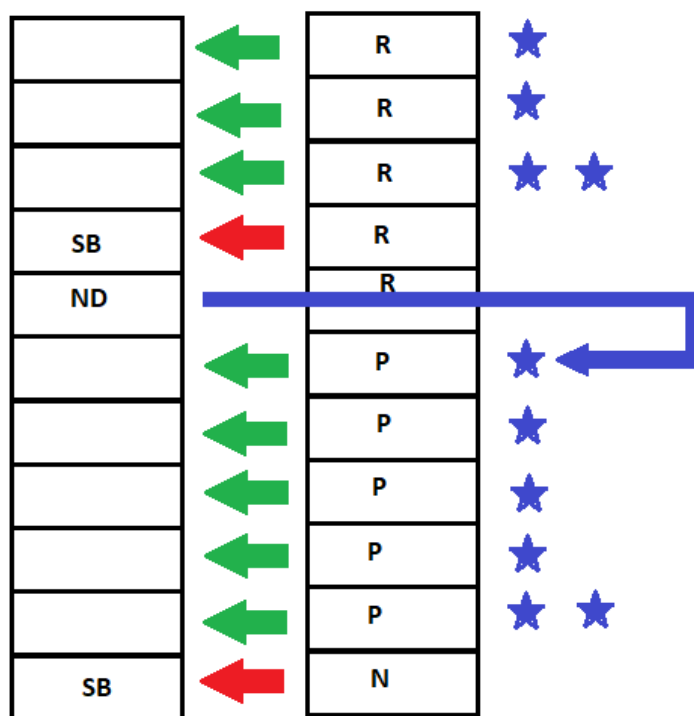
Najpierw jest tworzony miesiąc o długości odpowiadającej wybranemu, następnie wypełniany jest sobotami i niedzielami również ustalonymi dla wybranego miesiąca, następnie zmiany są losowane z tabeli zmian, która posiada odpowiadającą liczbie potrzebnych pracowników kolekcję zmian. Dzięki temu, że zaprogramowano losowość zmiany unika się przypadku, gdy pracownik miałby co miesiąc taki sam rozkład zmian co można uważać za niesprawiedliwe. Zmiany te są dodawane w puste dni (nie w weekendy), a na koniec dodawane są urlopy (jeżeli istnieją).



rys.4.3.1 Schemat powstawania miesiąca roboczego.

Szczegóły samego dodawania zmian przedstawiono przy pomocy rys.4.3.2. Głównym problemem do zrealizowania jest nieregularność miesięcy. Różna długość tygodni, miesiące zaczynające lub kończące się sobotą albo niedzielą spowodowały, że zdecydowano się na stworzenie wydajnego algorytmu, działającego dla każdego miesiąca.

Tabela po lewej obrazuje fragment tworzonego miesiąca roboczego, a tabela po prawej jest reprezentacją fragmentu jednego wiersza (losowo wybranego) z tabeli ze zmianami.



rys.4.3.2 Mechanizm zmiany tygodnia

Rys.4.3.2 uwzględnia następujące uproszczenia:

- ❖ zielone strzałki to “przejścia”, czyli momenty, kiedy dzień jest “pusty” i wpisywana jest kolejna zmiana,
- ❖ strzałki czerwone to “brak przejścia” następuje wtedy, gdy algorytm napotka dzień “pełny”,
- ❖ strzałka niebieska reprezentuje “skok” do następnego tygodnia w tabeli zmian,
- ❖ niebieskie gwiazdy reprezentują gdzie znajduje się wskaźnik tabeli przy kolejnych iteracjach.

Na rysunku 4.3.2 można zauważyć, że w pierwszym tygodniu dwa dni nie zostały wpisane z tabeli zmian tylko algorytm wykonał “skok” do następnego tygodnia (niebieska strzałka). Te “skoki” są obliczane na podstawie reszty z dzielenia wstawionych już dni przez 5.

Na listingu 4.3.1 pokazano fragment kodu odpowiadający za realizację opisanego algorytmu.

```
do
{
    rndNum = random.Next(0, workersNeeded);
} while (shiftsTable[rndNum,x] == null);

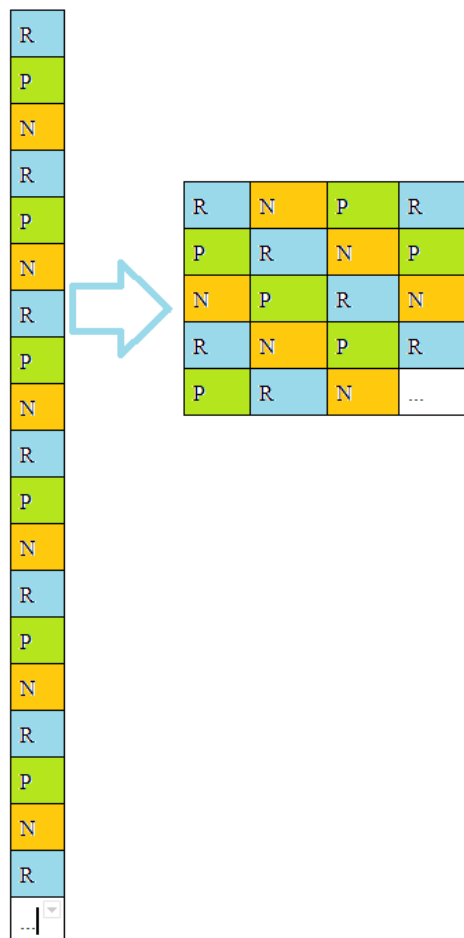
for (int i = 0; i < days; i++)
{
    if (daysTable[i] == null)
    {
        daysTable[i] = shiftsTable[rndNum, x];
        shiftsTable[rndNum, x] = null;
        x++;
    }
    if (daysTable[i] == "Nd")
    {
        int rest = x % 5;
        if (rest == 0) continue;
        for (int j = x; j < (x - rest + 5); j++)
        {
            shiftsTable[rndNum, j] = null;
        }
        x += (5 - rest);
    }
    if (x >= 26) break;
}
```

Listing 4.3.1 fragment metody *CreateWorkingMonth*.

Należy zauważyć, że na Listingu 4.3.1 znajduje się fragment losujący z tabeli zmian (*shiftsTable*) wiersz, który nie jest pusty, a przy “skoku” do następnego tygodnia algorytm zapobiega pozostawieniu niewykorzystanych symboli zmian. Bez tego zabezpieczenia fragment losujący z tabeli zmian mógłby błędnie losować wiersz, w którym częściowo znajdują się symbole a częściowo obiekty *null*.

Wypełnianie tabeli *shiftsTable* odbywa się w sposób, który korzysta z technologii stosu. Tworzony jest stos zmian bazujący na zmiennej *workersNeeded* pomnożonej razy 25 (jest to liczba maksymalnej ilości tygodni czyli 5 bez sobót i niedziel) i wypełniony naprzemiennie symbolami zmian porannych, popołudniowych i nocnych (“R”, “P”, “N”). Następnie utworzony stos “przesypuje” się do tablicy dwuwymiarowej, zmieniając wiersz co 25 elementów. Takie “przesypywanie” zastosowano, ponieważ obliczono, że dla dowolnej liczby reprezentowanej przez zmienną *workersNeeded*, która jest wielokrotnością liczby 3 elementy w tablicy ułożą się w regularny sposób pokazany na rys.4.3.3.

Z komentarzem [MS1]:



rys.4.3.3 Przemiana stosu w macierz.

Na rys.4.3.3 przedstawiono graficznie jak stos zamieniony zostaje w macierz zapisywaną w dwuwymiarowej tabeli. Każdy symbol zmiany na tym rysunku w rzeczywistości występuje powtórzony 5 razy (tydzień roboczy) ale widok ten został uproszczony.

Na Listingu 4.3.2 pokazano fragment kodu, który zaprogramowano w oparciu o opisane wyżej obliczenia. Jest to fragment metody *CreateShifts*.

```

string[] shiftSymbols = new string[workersNeeded * 25];
for (int i = 0; i < shiftSymbols.Length; i += 15)
{
    if (i + 1 == shiftSymbols.Length) break;
    for (int j = 0; j < 5; j++)
    {
        shiftSymbols[i + j] = "R";
    }
    if (i + 1 == shiftSymbols.Length) break;
    for (int j = 5; j < 10; j++)
    {
        shiftSymbols[i + j] = "P";
    }
    if (i + 1 == shiftSymbols.Length) break;
    for (int j = 10; j < 15; j++)
    {
        shiftSymbols[i + j] = "N";
    }
}
string[,] shiftsTable = new string[workersNeeded, 25];
for (int i = 0; i < workersNeeded; i++)
{
    for (int j = 0; j < 25; j++)
    {
        shiftsTable[i, j] = shiftSymbols[j + (25 * i)];
    }
}

```

Listing 4.3.2: Fragment kodu implementujący wypełnianie tablicy z symbolami zmian.

Jak można zauważyć na Listingu 4.3.2 omawiany stos jest reprezentowany tablicą *shiftSymbols* a macierz dwuwymiarową tablicą *shiftsTable*.

4.4 Zabezpieczenia

Zabezpieczenia stworzone w opisywanym programie można podzielić na następujące grupy:

- ❖ zabezpieczenia dostępu użytkownika (strona logowania),
- ❖ zabezpieczenia przed wpisaniem niepoprawnych danych (np. tekstu w polu pesel),
- ❖ zabezpieczenia bazy danych (przed zmienieniem lub utraceniem danych),
- ❖ zabezpieczenia spójności programu (np. zamknięciu wszystkich okien),
- ❖ zabezpieczenia przed niespodziewanym zamknięciem programu (bloki *try* i *catch*),
- ❖ zabezpieczenia przed wyciekiem danych.

Zabezpieczenie, które jest warte uwagi to zabezpieczenie bazy danych. Lokalna baza danych jest zabezpieczona przed wprowadzeniem danych, które przez swoją składnię mogą zaszkodzić poleceniom w języku SQL, np. użycie apostrofu w imieniu "D'artagnan" lub inne znaki interpunkcyjne używane w komendach SQL mogą zaburzyć pracę algorytmu i zakończyć się błędem lub złym zapisem informacji. Poza tym jest chroniona przed atakami typu SQL Injection, polegającym na wykorzystywaniu składni języka do wykonywania dodatkowych zapytań w ramach przesyłanego do bazy danych zapytania. Ochronę to zapewnia przekazywanie wszystkich danych wprowadzanych przez użytkownika jako parametrów używanej komendy, która jest typu *SqlCommand* pokazane na listingu 4.4.1.

```
public void Query()
{
    using (SqlConnection connection = new SqlConnection(DataBaseHelper.ConnectionString()))
    {
        try
        {
            SqlDataAdapter adapter = new SqlDataAdapter();
            connection.Open();
            SqlCommand command = new SqlCommand(sqlQuery);
            int i = 0;
            foreach (string s in valuesNames)
            {
                command.Parameters.AddWithValue(s, values[i]);
                i++;
            }
            command.CommandType = CommandType.Text;
            command.Connection = connection;
            adapter.SelectCommand = command;
            DataTable datatable = new DataTable();
            adapter.Fill(datatable);
            connection.Close();
        }
        catch (SystemException ex)
        {
            MessageBox.Show(string.Format("An error occurred: {0}", ex.Message));
        }
    }
}
```

Listing 4.4.1: Metoda Query wysyłająca zapytanie do bazy danych i odbierająca wynik.

Na listingu 4.4.1 pokazano jak w większości przypadków, za pośrednictwem właśnie tej metody odbywa się komunikacja z bazą danych. Za każdym razem użyto bloku *try i catch* a parametry dodaje się za pomocą metody *AddWithValue*, która jest zawartością biblioteki *System.Data.SqlClient*.

5. PODSUMOWANIE

W tym rozdziale opisano wymagania sprzętowe, instrukcję instalacji oraz podsumowano zgodność końcowej pracy z założeniami i sformułowano wnioski na temat projektu.

5.1 Minimalne wymagania sprzętowe

Do działania oprogramowania należy zapewnić sprzęt o następujących parametrach:

- ❖ System Operacyjny Windows 7 SP1 : Home Premium, Professional, Enterprise, Ultimate lub wyższy,
- ❖ Procesor 1,8 GHz lub szybszy. Zalecany jest co najmniej czterordzeniowy,
- ❖ 2 GB pamięci RAM,
- ❖ Miejsce na dysku twardym: minimum 1GB,
- ❖ Stację dysków CD/DVD,
- ❖ Karta graficzna obsługująca minimalną rozdzielczość ekranu 720p (1280 na 720).

5.2 Instalacja

Przed uruchomieniem programu należy zainstalować program Visual Studio 2019 wraz ze składnikiem SQL Server Express 2016 LocalDB.

Zaleca się skopiowanie danych z nośnika na lokalny dysk.

Program uruchamia się za pomocą kompilatora Visual Studio 2019.

Login i hasło do strony logowania to

- ❖ Login: admin
- ❖ Hasło: pass

5.3 Podsumowanie

Celem projektu było opracowanie programu do tworzenia Harmonogramów pracy wraz z opisem jego działania.

Zakres projektu obejmował analizę tematu, stworzenie aplikacji użytkowej w tym interfejsu użytkownika, algorytmu połączonego z lokalną bazą danych oraz przeprowadzenie testów aplikacji i implementację wyczerpującego opisu wraz z wizualizacją pracy programu.

Cele przyjęte w projekcie zostały zrealizowane zgodnie z zamiarem a efektem jest w pełni działający program pozwalający na zarządzanie zasobami ludzkimi i materialnymi oraz tworzenia harmonogramów pracy.

W początkowej fazie realizacji projektu zebrano informację na temat możliwości implementacji założeń, zdecydowano, że program będzie zrealizowany jako aplikacja okienkowa na systemy Windows realizowana przy pomocy programu Visual Studio 2019 w interfejsie Windows Forms. Zbadano potrzeby jakie przewiduje aplikacja i zaprojektowano interfejs użytkownika. Następnie podjęto decyzję o stworzeniu bazy danych, która okazała się niezbędna przy projekcie. Wybrano lokalną bazę danych, którą zapewnia składnik programu Visual Studio 2019, czyli SQL Server Express 2016 LocalDB. Potem podzielono odpowiednio rolę programu i zaimplementowano klasy pomocnicze oraz wypełniono metody w klasach wygenerowanych przy tworzeniu okien uzupełniając je metodami ułatwiającymi i organizującymi pracę. Przed końcowym efektem program wiele razy testowano pod względem spójności, funkcjonalności i bezpieczeństwa jednocześnie wprowadzając wiele poprawek, aby wyeliminować wszystkie wychwycone błędy, aby ostateczny projekt był w jak największym stopniu wolny od wad.

5.4 Wnioski

Na podstawie stworzonego rozwiązania można stwierdzić, że programowanie może usprawnić każdą dziedzinę życia i upraszczać mechanizmy, które z powodu zakorzenienia w naszym umyśle wykonujemy w sposób niewydajny. Analiza tematu pokazała jak stosunkowo proste zagadnienie polegające na opracowaniu harmonogramu może zostać skomplikowane przy zwiększonej ilości ludzi lub zasobów, oraz jak wiele trzeba zaplanować, aby poprawnie i wydajnie prowadzić firmę. Program ten stwarza możliwość zautomatyzowania niektórych aspektów, co może prowadzić do zaoszczędzenia czasu, lepszej organizacji pracy i wynikającej z tego redukcji kosztów prowadzenia firmy.

Podczas pracy nad projektem stwierdzono, że zdecydowanie największym problemem było opracowanie koncepcji a potem zaimplementowanie mechanizmu tworzącego harmonogramy. Zróżnicowanie liczby dni każdego miesiąca, zmieniająca się ilość pracowników, oraz różnorodność zmian w organizacji pracy skomplikowały analizowany problem, jednak rozwiązywanie wymienionych kwestii wniosło wiele w obręb całego programu.

Z komentarzem [MS2]:

Kierunki rozwoju aplikacji wynikają jasno z ograniczeń, które można ominąć. Jako główne z nich można wskazać:

- ❖ rozwiniecie liczby użytkowników poprzez przeniesienie bazy na serwer,
- ❖ dodanie szablonów układania zmian odpowiadających wymogom klientów,
- ❖ eksportowanie harmonogramów do plików lub wysyłanie ich do pracowników za pośrednictwem skrzynki pocztowej przypisanej do pracownika,
- ❖ tworzenie kont ze zróżnicowanym poziomem dostępu dzieląc pracę przy zarządzaniu,
- ❖ dodawanie próśb i ich rozstrzyganie,
- ❖ edytowanie ręczne harmonogramów na zaawansowane potrzeby,
- ❖ tworzenie grup lub map stanowisk oraz ich widoków umożliwiających ich uzupełnianie,
- ❖ prowadzenie kontroli godzinowej pracowników.

Jak widać możliwych kierunków do rozwinięcia jest wiele i każdy kolejny otwiera możliwości na następne mechanizmy automatyzacji procesu organizacji pracy.

6. LITERATURA

- [1] Sells C.: Windows Forms Programming in C#, Pearson Education, 2003
- [2] Chłosta P.: Aplikacje Windows Forms .NET w C#, Wydawnictwo Naukowe PWN, Mikom, 2006

7. ŹRÓDŁA

- [1] Wikipedia - wolna encyklopedia
https://pl.wikipedia.org/wiki/Wikipedia:Strona_główna, dostęp dnia: 27.01.2020.
- [2] Stack Overflow - portal zrzeszający programistów i ich problemy
<https://stackoverflow.com>, dostęp dnia: 27.01.2020.
- [3] Dokumentacja Microsoftu
<https://docs.microsoft.com/pl-pl/dotnet/framework/>, dostęp dnia: 27.01.2020.
- [4] w3schools - portal gromadzący proste polecenia w językach programowania
https://www.w3schools.com/sql/sql_alter.asp , dostęp dnia 27.01.2020.
- [5] Code project - portal gromadzący wiedzę programistyczną
<https://www.codeproject.com>, dostęp dnia: 27.01.202.