Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Modelowanie i analiza systemów informatycznych – Projekt

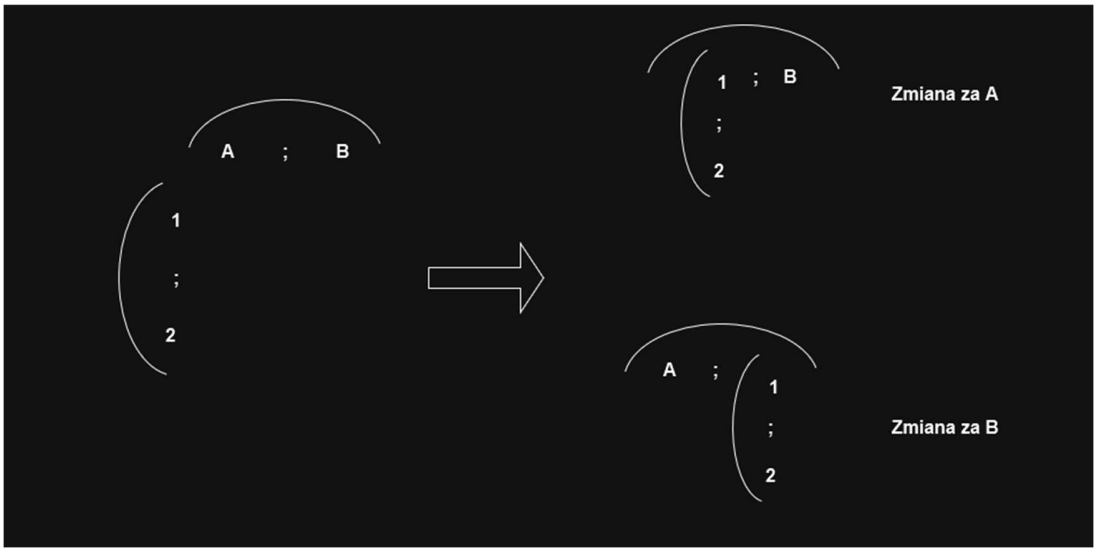
Temat: Zmiana unitermu poziomej operacji sekwencjonowania unitermów na pionową operację sekwencjonowania unitermów	**Opracowanie:** Michał Kaczor (91268) 1IZ22B **Data:** 01.01.2025

Spis treści

3

1. Wstęp

Celem projektu było stworzenie programu komputerowego realizującego operację zmiany unitermu poziomej operacji sekwencjonowania unitermów na pionową operację sekwencjonowania unitermów. Diagram poniżej przedstawia graficzną reprezentację przebiegu tej operacji.



Ilustracja graficzna zmiany unitermów według tematu

2. Technologia

Do realizacji projektu wybrałem język C# wraz z biblioteką graficzną WPF. Korzystałem ze środowiska Visual Studio 2022 oraz Blend for Visual Studio 2022, co umożliwiło mi łatwe zarządzanie kodem programu oraz tworzenie interfejsu graficznego za pomocą funkcji „drag & drop”.

Pomimo że wybrałem tę samą technologię co Profesor Ovsyak, nie korzystałem z jego projektu. Zdecydowałem się na samodzielną realizację, co skutkowało brakiem integracji z bazą danych. Program stanowi moją własną interpretację tematu, a nie przeróbkę projektu Profesora.

3. Założenia

Program ma być aplikacją okienkową, której interfejs graficzny umożliwia łatwą i intuicyjną obsługę operacji zmiany unitermów według tematu projektu. Po uruchomieniu programu użytkownik zobaczy na górze strony tytuł aplikacji oraz krótką instrukcję opisującą zasadę jej działania. Dodatkowo, w programie znajdzie się ikona umożliwiająca wyświetlenie szczegółowego przebiegu operacji zmiany unitermów w formie graficznej, co ułatwi zrozumienie funkcji programu (ten sam diagram co zaprezentowany we wstępie).

Po lewej stronie głównego ekranu umieszczone będzie menu, w którym znajdą się wszystkie przyciski do obsługi programu. Część z tych przycisków będzie dostępna tylko w określonych etapach działania programu, a ich dostępność będzie dynamicznie blokowana lub odblokowywana w zależności od postępu użytkownika.

W głównej części okna programu użytkownik zobaczy dwa unitermy – jeden pionowy i jeden poziomy – z polami tekstowymi, w które będzie mógł wprowadzić odpowiednie wartości. Każdy uniterm musi zawierać dokładnie dwie wartości. Pola tekstowe oraz linie unitermów będą dynamicznie dostosowywać się do długości wprowadzonych danych.

Po wprowadzeniu wartości, użytkownik będzie mógł wybrać, który uniterm chce zamienić: lewy lub prawy. Po kliknięciu przycisku do zamiany, wartości zostaną zamienione, a pola tekstowe staną się nieedytowalne. Użytkownik będzie miał możliwość cofnięcia zmiany, co pozwoli na ponowne edytowanie wartości, lub zresetowania całego ekranu, co usunie wszystkie dane.

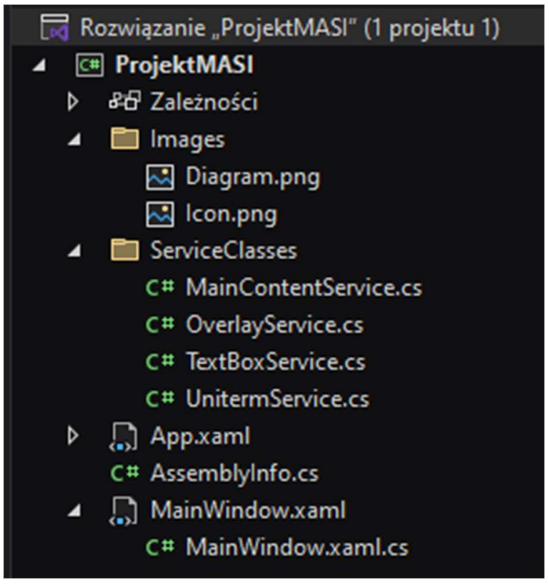
Dodatkowo, użytkownik będzie mógł wyczyścić wprowadzone dane unitermów przed ich zamianą. Program będzie wyposażony w suwak, który umożliwi skalowanie wielkości wyświetlanych unitermów, a główne okno aplikacji będzie w pełni skalowalne, automatycznie dostosowując rozmiar elementów interfejsu do nowego rozmiaru okna.

4. Najważniejsze części kodu

Projekt został podzielony na mniejsze fragmenty, co pozwoliło na rozdzielenie poszczególnych operacji i zwiększenie przejrzystości oraz czytelności kodu. Główna logika programu znajduje się w klasie ***MainWindow.xaml.cs,*** która odpowiada za obsługę zdarzeń w głównym oknie aplikacji (***MainWindow***). Klasa ta jest bezpośrednio powiązana z plikiem ***MainWindow.xaml***, który określa wygląd i strukturę interfejsu użytkownika.

Dodatkowo, w projekcie znajdują się cztery klasy pomocnicze, umieszczone w folderze ***ServiceClasses***. Każda z tych klas realizuje określoną funkcję, wspierającą działanie głównego okna aplikacji. Klasa ***MainWindow.xaml.cs*** korzysta z tych klas pomocniczych, aby obsługiwać zdarzenia oraz realizować odpowiednie operacje.

Pliki graficzne wykorzystywane w projekcie zostały umieszczone w folderze ***Images***, co pozwala na ich łatwe zarządzanie i dostępność w trakcie działania aplikacji.



Struktura projektu

5

Klasa MainWindow odpowiada za obsługę głównego okna aplikacji i interfejsu użytkownika. Zawiera kilka kluczowych pól, w tym:

- ***isSwapped:*** zmienną statyczną, która przechowuje informację o tym, czy unitermy zostały zamienione miejscami.
- Pola do przechowywania instancji klas pomocniczych: ***overlayService, textBoxService, unitermService, mainContentService***, które realizują różne funkcje aplikacji.
- ***textFields:*** tablicę, która zawiera odwołania do pól tekstowych, w których użytkownik wprowadza wartości unitermów

Konstruktor klasy inicjalizuje obiekty klas pomocniczych, a następnie wywołuje metodę ***InitializeComponent()***, która odpowiedzialna jest za załadowanie komponentów interfejsu użytkownika. Po inicjalizacji komponentów, konstruktor przypisuje obsługę zdarzenia ***ValueChanged*** dla suwaka ***ScaleSlider***, umożliwiając reagowanie na zmiany jego wartości. Na końcu, konstruktor wypełnia tablicę ***textFields*** odwołaniami do odpowiednich pól tekstowych interfejsu.

```
public partial class MainWindow : Window
{
    public static bool isSwapped = false; // zmienna globalna określająca czy
aktualnie unitermy są zamienione miejscami

    private OverlayService overlayService;
    private TextBoxService textBoxService;
    private UnitermService unitermService;
    private MainContentService mainContentService;
    private TextBox[] textFields;

    public MainWindow()
    {
        overlayService = new OverlayService();
        textBoxService = new TextBoxService();
        unitermService = new UnitermService();
        mainContentService = new MainContentService();


        InitializeComponent();

        // Przypisujemy obsługę zdarzenia ValueChanged do suwaka ScaleSlider po
inicjalizacji komponentów okna
        ScaleSlider.ValueChanged += ScaleSlider_ValueChanged;

        textFields = new TextBox[] { HUValue1TextField, HUValue2TextField,
VUValue1TextField, VUValue2TextField };
    }

...
```

6

Metoda ***SwapUniterms*** jest wywoływana po kliknięciu przycisku „Zamień” w głównym oknie aplikacji. Jej zadaniem jest wywołanie odpowiedniej funkcji z klasy pomocniczej ***unitermService***, która realizuje całą operację zamiany unitermów. Metoda przekazuje do tej funkcji różne elementy interfejsu użytkownika, takie jak przyciski, pola tekstowe i radio buttony, które są niezbędne do wykonania operacji. Dzięki temu, po kliknięciu przycisku, aplikacja przeprowadza zamianę wartości unitermów oraz aktualizuje stan interfejsu zgodnie z wynikami operacji.

```
        // Metoda zamieniająca miejscami unitermy
        private void SwapUniterms(object sender, RoutedEventArgs e)
        {
            unitermService.Swap(sender, e, LeftRadioButton, RightRadioButton,
HUValue1TextField, HUValue2TextField, VerticalUniterm, HorizontalUniterm,
SwapButton, UndoButton, ClearFieldsButton, textFields);
        }
```

Metoda ***Swap*** odpowiada za przeprowadzenie całej operacji zamiany unitermów. Na początku sprawdza, czy wszystkie pola tekstowe zostały wypełnione, wywołując metodę ***IsNotEmpty*** z klasy pomocniczej ***TextBoxService***. Jeśli któreś pole jest puste, użytkownik otrzymuje komunikat o błędzie i operacja zostaje przerwana.

Następnie, w zależności od tego, który z radio buttonów (lewy lub prawy) jest zaznaczony, metoda określa, który element (pole tekstowe) ma zostać zamieniony z unitermem (panel pionowy lub poziomy). Po zidentyfikowaniu odpowiednich elementów, metoda przenosi je między panelami (pionowym i poziomym) w interfejsie użytkownika.

Dodatkowo, metoda sprawdza, który przycisk („Zamień”, „Cofnij” lub „Reset”) wywołał operację. Na tej podstawie odpowiednio modyfikuje stan dostępnych przycisków, radio buttonów oraz innych elementów interfejsu, takich jak pola tekstowe. W przypadku zamiany, dezaktywuje przycisk „Zamień”, blokuje radio buttony oraz przycisk „Wyczyść”, a także ukrywa przeniesiony panel. Z kolei po cofnięciu zmian lub ich zresetowaniu, przywracane są wcześniejsze ustawienia interfejsu.

```
// Metoda zamieniająca miejscami unitermy
public void Swap(object sender, RoutedEventArgs e, RadioButton leftRadioButton,
RadioButton rightRadioButton, TextBox hUValue1TextField, TextBox
hUValue2TextField, StackPanel verticalUniterm, StackPanel horizontalUniterm,Button
swapButton, Button undoButton, Button clearFieldsButton, TextBox[] textFields)
{
    TextBoxService textBoxService = new TextBoxService();

    // Sprawdzenie, czy wszystkie pola tekstowe są wypełnione
    if (!textBoxService.IsNotEmpty(textFields))
    {
        MessageBox.Show("Wszystkie pola tekstowe muszą być wypełnione przed
wykonaniem zamiany!", "Błąd", MessageBoxButton.OK, MessageBoxImage.Warning);
        return;
    }

    bool isLeftSelected = leftRadioButton.IsChecked ?? false;

    // Określenie elementów źródłowych i docelowych
    var sourcePanel = isLeftSelected ? hUValue1TextField : hUValue2TextField;
    var targetPanel = verticalUniterm;
```

7

```
    var sourceParent = (Panel)sourcePanel.Parent;
    var targetParent = (Panel)targetPanel.Parent;

    // Przeniesienie elementów
    sourceParent.Children.Remove(sourcePanel);
    targetParent.Children.Remove(targetPanel);

    sourceParent.Children.Add(targetPanel);
    targetParent.Children.Add(sourcePanel);

    // Rzutowanie sender na Button, ponieważ sender to obiekt, który wywołał
metodę
    Button? clickedButton = sender as Button;

    // Sprawdzamy, czy rzutowanie powiodło się
    if (clickedButton != null)
    {
        // Uzyskujemy nazwę przycisku
        string buttonName = clickedButton.Name;

        // Ustawiamy odpowiednie opcje dla pozostałych elementów ekranu w
zależności od tego, któy przycisk był kliknięty
        if (buttonName == "SwapButton")
        {
            MainWindow.isSwapped = true;

            swapButton.IsEnabled = false;
            undoButton.IsEnabled = true;

            leftRadioButton.IsEnabled = false;
            rightRadioButton.IsEnabled = false;

            clearFieldsButton.IsEnabled = false;

            sourcePanel.Visibility = Visibility.Hidden;

            textBoxService.ToggleTextFields(textFields, false);
        }
        else if (buttonName == "UndoButton" || buttonName == "ResetButton")
        {
            MainWindow.isSwapped = false;

            swapButton.IsEnabled = true;
            undoButton.IsEnabled = false;

            leftRadioButton.IsEnabled = true;
            rightRadioButton.IsEnabled = true;

            clearFieldsButton.IsEnabled = true;

            sourcePanel.Visibility = Visibility.Visible;

            textBoxService.ToggleTextFields(textFields, true);
        }
    }
}
```

8

Metoda ***ScaleHUTextPanel*** odpowiada za dynamiczne dopasowanie szerokości linii unitermu poziomego do długości wprowadzonych wartości w pola tekstowe. Jest wywoływana, gdy zmienia się rozmiar panelu tekstowego ***hUTextPanel***. W pierwszej kolejności sprawdzane jest, czy obiekty ***hUTextPanel*** oraz ***topPath*** są poprawnie zainicjalizowane. Następnie, szerokość linii (reprezentowanej przez obiekt ***topPath***) jest ustawiana na wartość odpowiadającą szerokości panelu tekstowego.

Kolejnym krokiem jest dostosowanie danych geometrycznych ścieżki ***topPath***, aby linia odpowiadała nowemu rozmiarowi. Zmienia się kształt ścieżki, co pozwala na jej odpowiednią adaptację do szerokości panelu tekstowego, zapewniając poprawne wyświetlanie w interfejsie aplikacji.

```
        // Metoda obsługująca dopasowanie wielkości lini unitermu poziomego w
zależności od zmiany wielkości panelu tekstowego
        public void ScaleHUTextPanel(object sender, SizeChangedEventArgs e,
StackPanel hUTextPanel, Path topPath)
        {
            if (hUTextPanel != null && topPath != null)
            {
                // Ustawienie szerokości TopPath na szerokość HUTextPanel
                topPath.Width = hUTextPanel.ActualWidth;

                // Dostosowanie danych geometrycznych Path
                topPath.Data = new PathGeometry(new PathFigureCollection
        {
            new PathFigure
            {
                StartPoint = new Point(0, 0),
                Segments = new PathSegmentCollection
                {
                    new QuadraticBezierSegment
                    {
                        Point1 = new Point(topPath.Width / 2, -topPath.Height /
2),
                        Point2 = new Point(topPath.Width, 0)
                    }
                }
            }
        });
            }
        }
```

9

Metoda ***ScaleTextBox*** odpowiada za dynamiczną zmianę szerokości pola tekstowego w zależności od długości wprowadzonego tekstu. Została zaprojektowana w taki sposób, aby zapewnić odpowiednią szerokość dla tekstu, zachowując minimalną szerokość pola tekstowego.

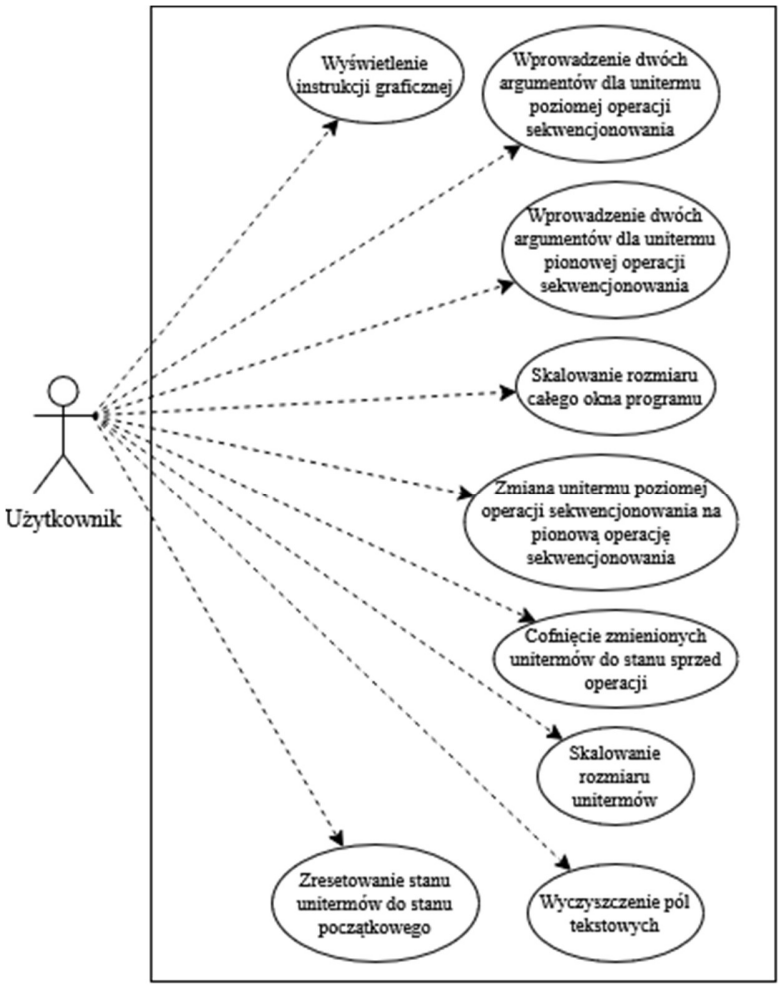
W pierwszym kroku, metoda sprawdza, czy obiekt wywołujący zdarzenie jest typu ***TextBox***. Następnie, oblicza szerokość tekstu wprowadzonego do pola za pomocą klasy ***FormattedText***, która mierzy szerokość tekstu zgodnie z jego czcionką i rozmiarem. Na podstawie tej szerokości, metoda wylicza nową szerokość pola tekstowego, dodając dodatkowy odstęp dla lepszego wyświetlania tekstu.

Na końcu, szerokość pola tekstowego jest ustawiana na wartość większą z obliczonej szerokości i minimalnej szerokości, aby zapewnić, że pole nie będzie miało mniejszej szerokości niż ustalona wartość minimalna.

```
// Metoda obsługująca zmianę szerokości pola tekstowego w zależności od długości
wprowadzonego tekstu
public void ScaleTextBox(object sender, TextChangedEventArgs e)
{
    if (sender is TextBox textBox)
    {
        // Pobranie minimalnej szerokości z kontrolki TextBox
        double minWidth = textBox.MinWidth;

        // Mierzenie szerokości tekstu w TextBox
        var formattedText = new FormattedText(
            textBox.Text,
            System.Globalization.CultureInfo.CurrentCulture,
            FlowDirection.LeftToRight,
            new Typeface(textBox.FontFamily, textBox.FontStyle,
textBox.FontWeight, textBox.FontStretch),
            textBox.FontSize,
            Brushes.Black,
            new NumberSubstitution(),
            1);

        // Obliczanie nowej szerokości na podstawie tekstu
        double newWidth = formattedText.Width + 10; // Dodajemy 10 dla lepszego
odstępu

        // Ustawianie szerokości TextBox z zachowaniem minimalnej szerokości
        textBox.Width = Math.Max(newWidth, minWidth);
    }
}
```

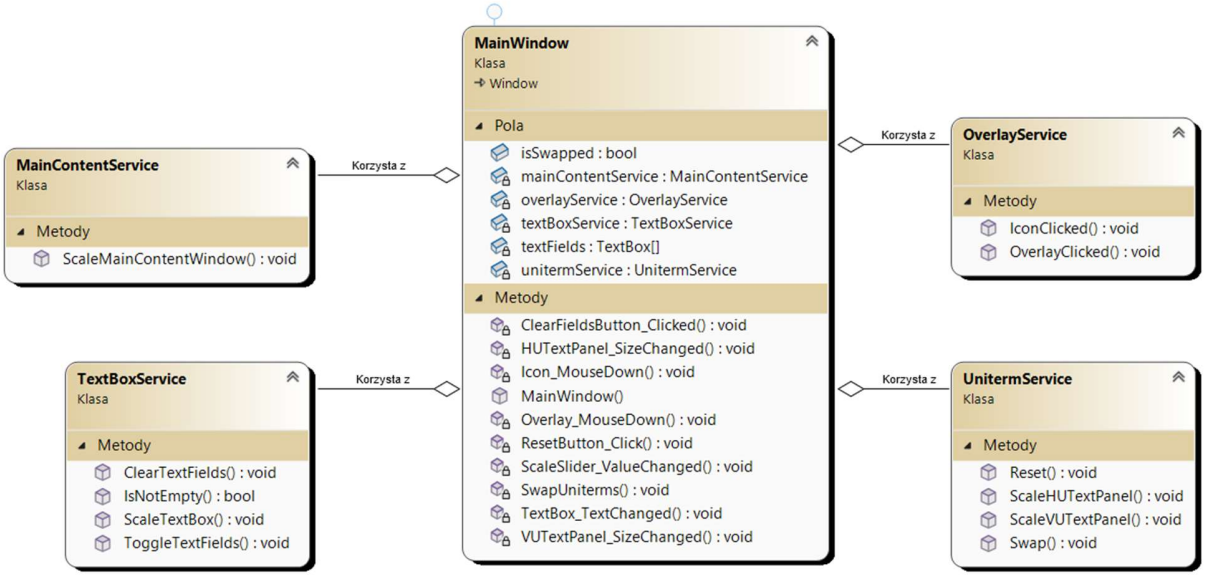
5. Diagram przypadków użycia

Poniżej przedstawiony został diagram przypadków użycia dla mojego projektu.



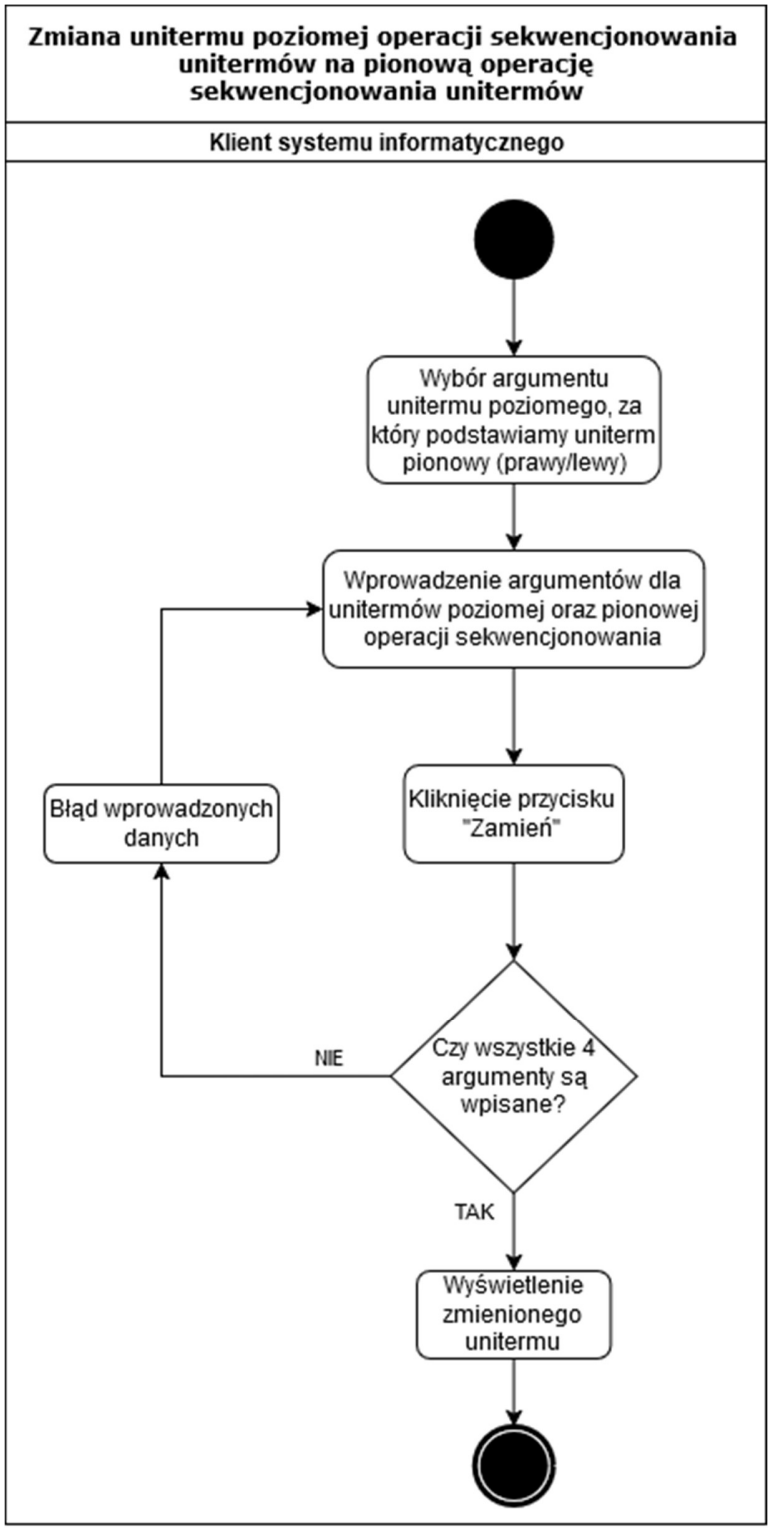
6. Diagram klas

Poniżej przedstawiony został diagram klas dla mojego projektu.



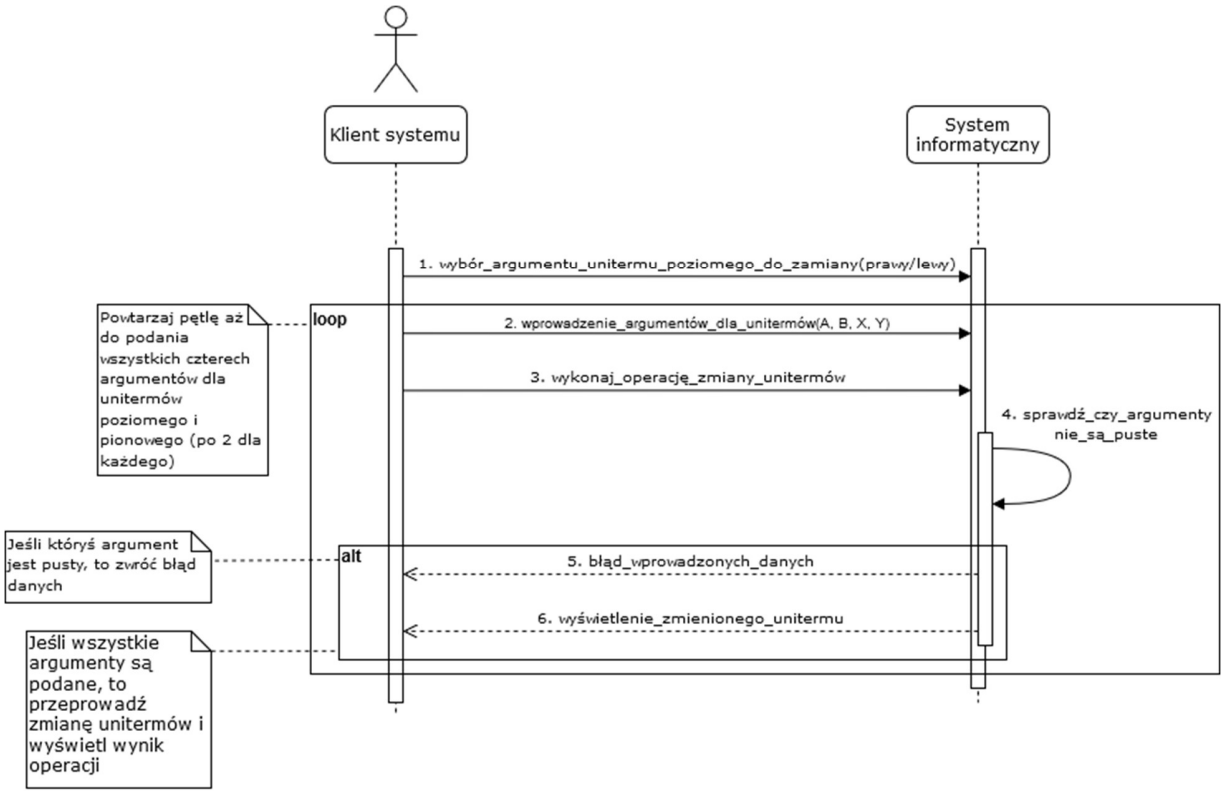
7. Diagram aktywności

Poniżej przedstawiony został diagram aktywności dla procesu zmiany unitermu poziomej operacji sekwencjonowania unitermów na pionową operację sekwencjonowania unitermów.



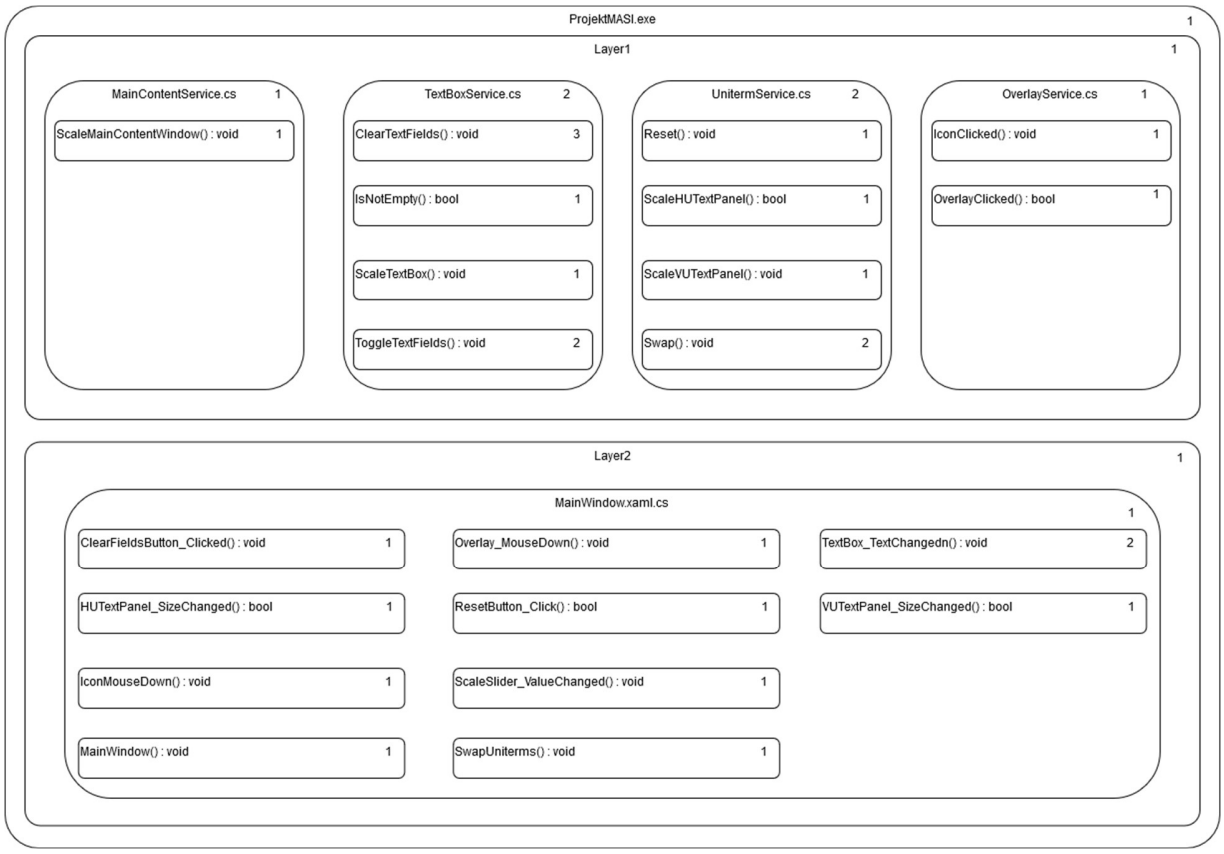
8. Diagram sekwencji

Poniżej przedstawiony został diagram sekwencji dla procesu zmiany unitermu poziomej operacji sekwencjonowania unitermów na pionową operację sekwencjonowania unitermów.



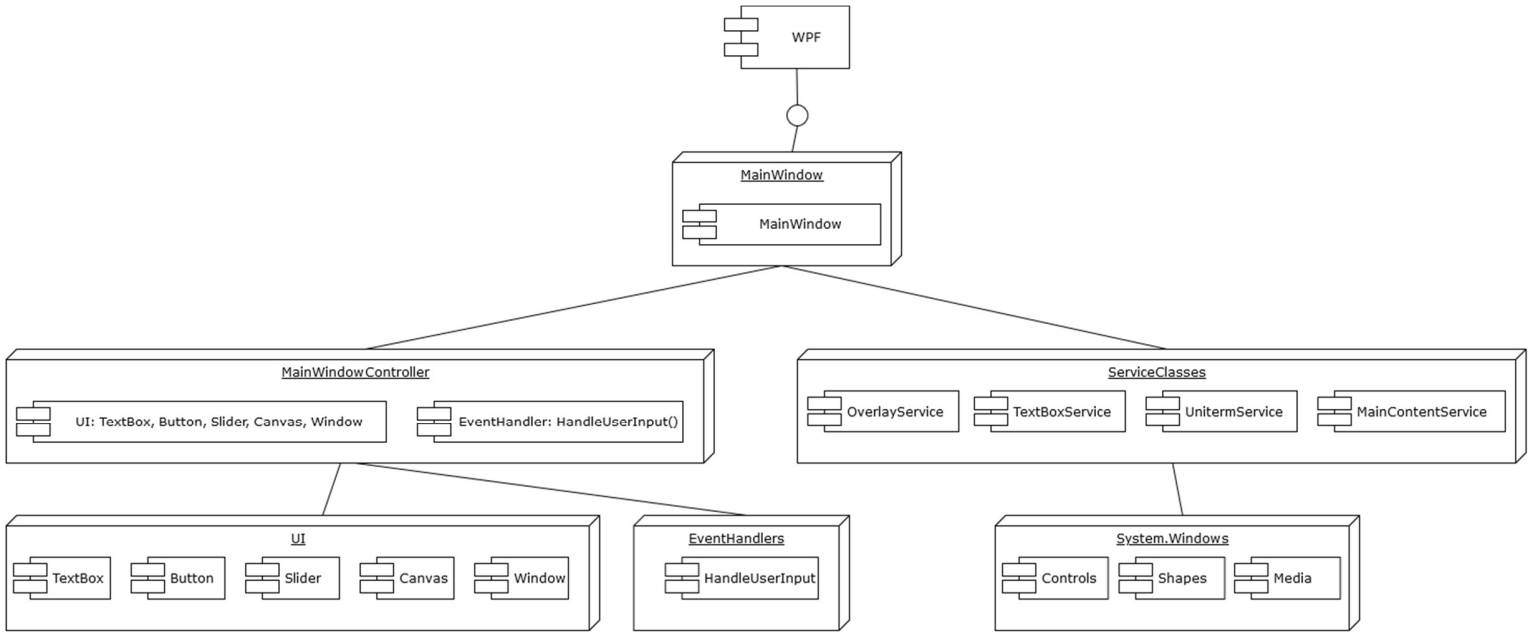
9. Diagram warstw

Poniżej przedstawiony został diagram warstw dla mojego projektu.



10. Diagram komponentów

Poniżej przedstawiony został diagram komponentów dla mojego projektu.

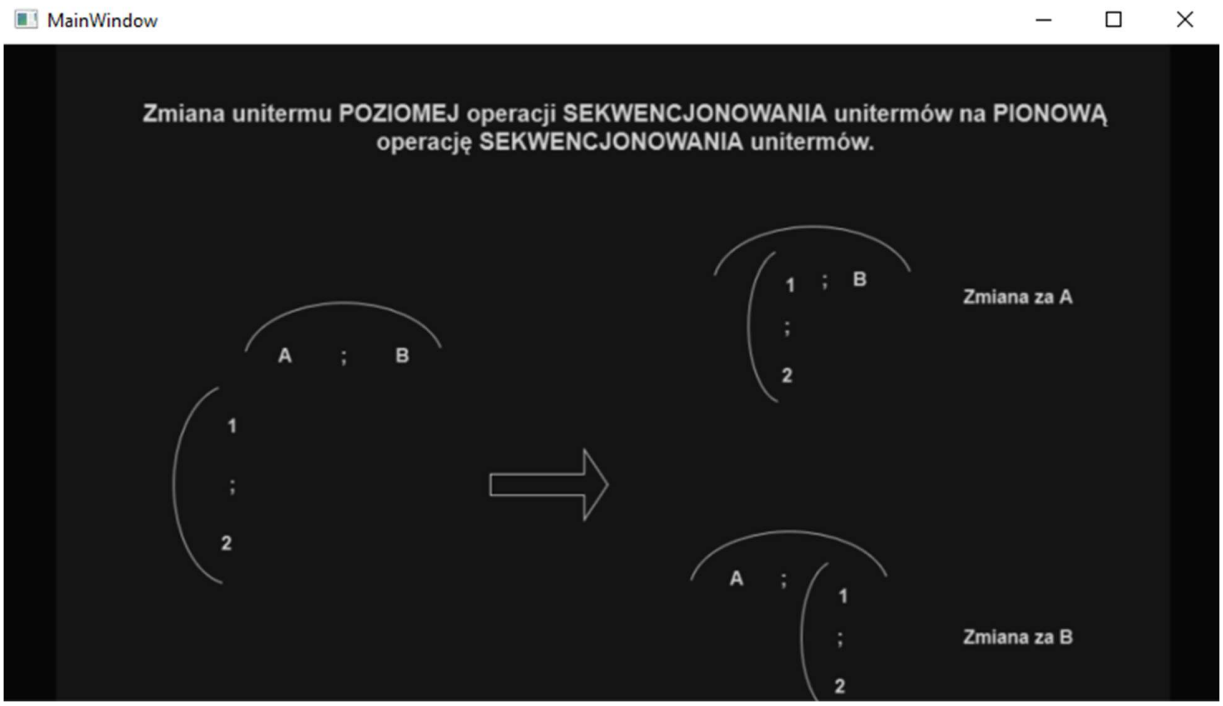


11. Struktura bazy danych

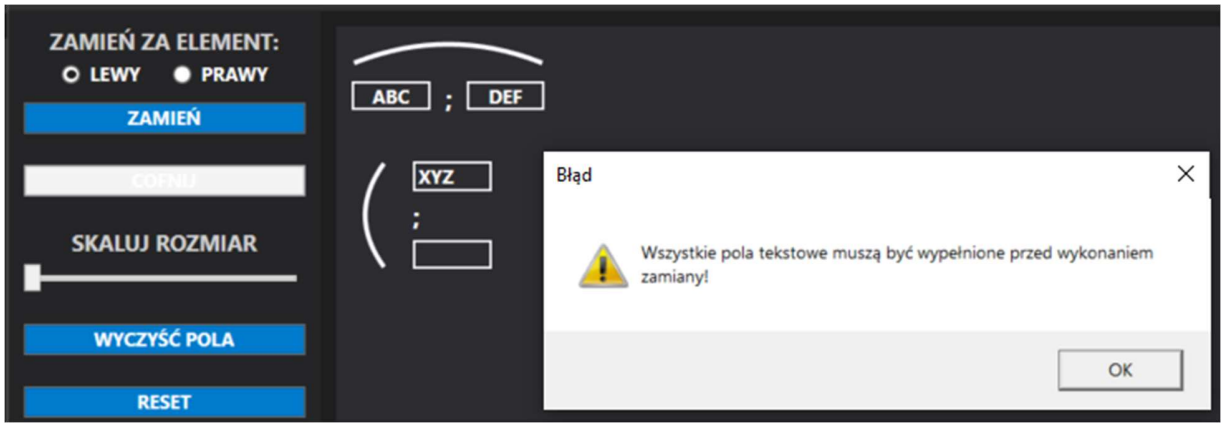
Z racji na zaimplementowanie własnego rozwiązania, bez wzorowania się na projekcie Profesora, to aplikacja nie została zintegrowana z bazą danych.

12. Zdjęcia z działania aplikacji

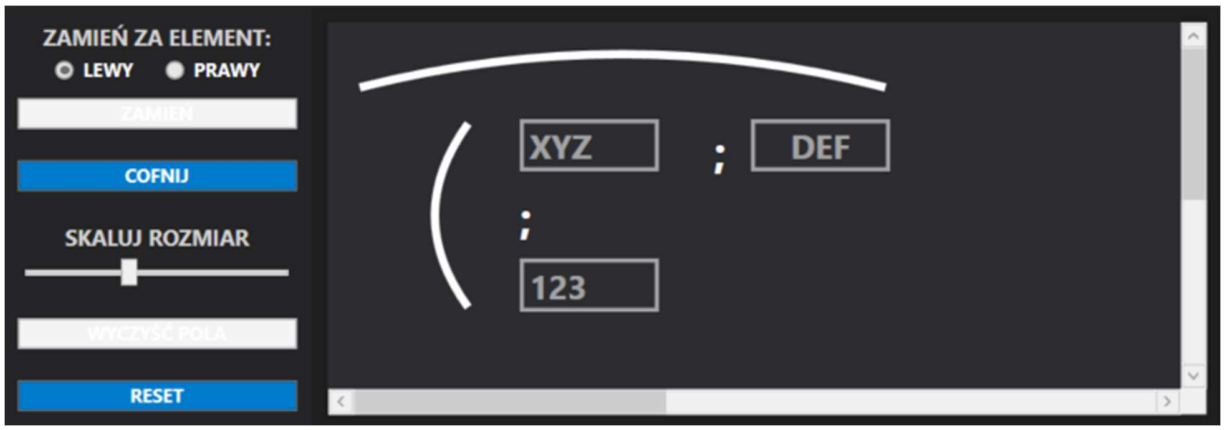
Ekran główny aplikacji projektowej



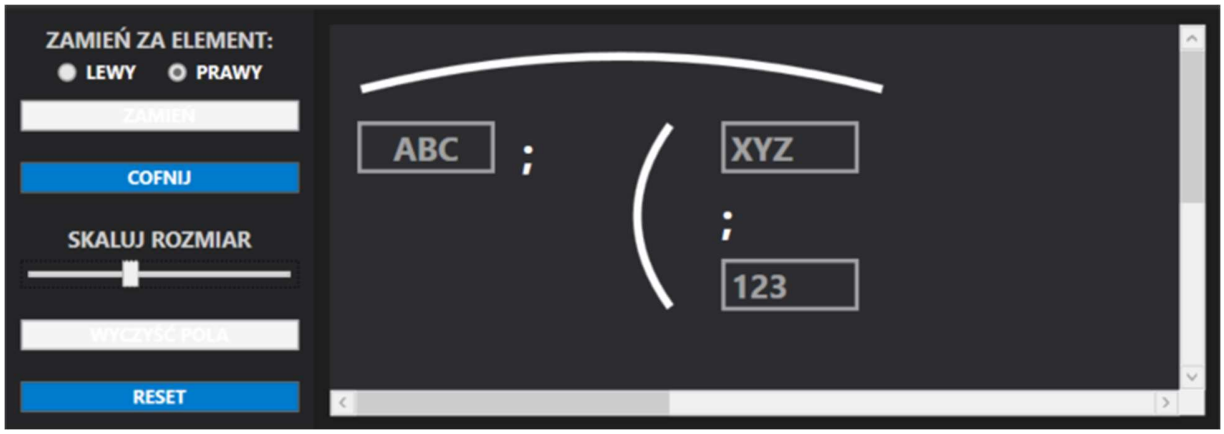
Podgląd graficzny przebiegu zmiany unitermów pokazany po kliknięciu ikony w instrukcji



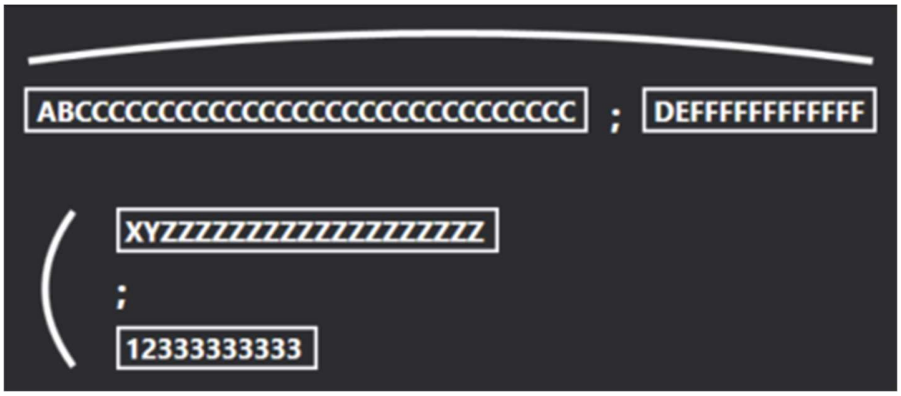
Okno dialogowe wyświetlane, gdy nie jest spełniony warunek - wszystkie pola wypełnione



Działanie zmiany unitermów (zmiana za prawy) oraz skalowania



Działanie zmiany unitermów (zmiana za lewy) oraz skalowania



Działanie skalowania linii unitermów oraz wielkości pól tekstowych

13. Wnioski

Projekt został pomyślnie zrealizowany zgodnie z początkowymi założeniami. Program okienkowy, który umożliwia zamianę unitermów, został zaimplementowany i przetestowany. Wszystkie funkcjonalności, takie jak zamiana unitermów, walidacja danych, dynamiczne dopasowywanie szerokości elementów, oraz interaktywność z użytkownikiem, działają zgodnie z wymaganiami. Program spełnia założenia projektowe i realizuje wymagania przedstawione w zasadach zaliczenia przedmiotu. Bieżące sprawozdanie również zawiera wszystkie niezbędne elementy wymagane do zaliczenia przedmiotu.