

Przedmiot: Inżynieria Oprogramowania

Prowadzący: mgr inż. Przemysław Grabowski

Temat: Prezentacja projektu – To-Do List

Wykonujący projekt: Michał Truszkowski, Konrad Wierzbicki, Rafał Złotkowski

Data: 03.05.2025

Cel projektu	1
Wymagania funkcjonalne	1
Interfejs użytkownika	2
Funkcje.	3
Funkcja Utworzenia nowego zadania	3
Wygląd formularza dodawania nowego zadania do listy	3
Funkcja oznaczenia wykonania zadania	4
Funkcja edycji zadania	4
Wygląd formularza edycji zadania	5
Funkcja usuwania zadania	6
Funkcja eksportu listy zadań	6
Komunikat o udanym eksporcie	7
Funkcja importu	7
Komunikat o udanym imporcie	8
Testy aplikacji	8
Testy Wartości Bezwzględnych	9
Testy dat	10
Testy tekstu	11
Testy opisu	11
Podział obowiązków	12

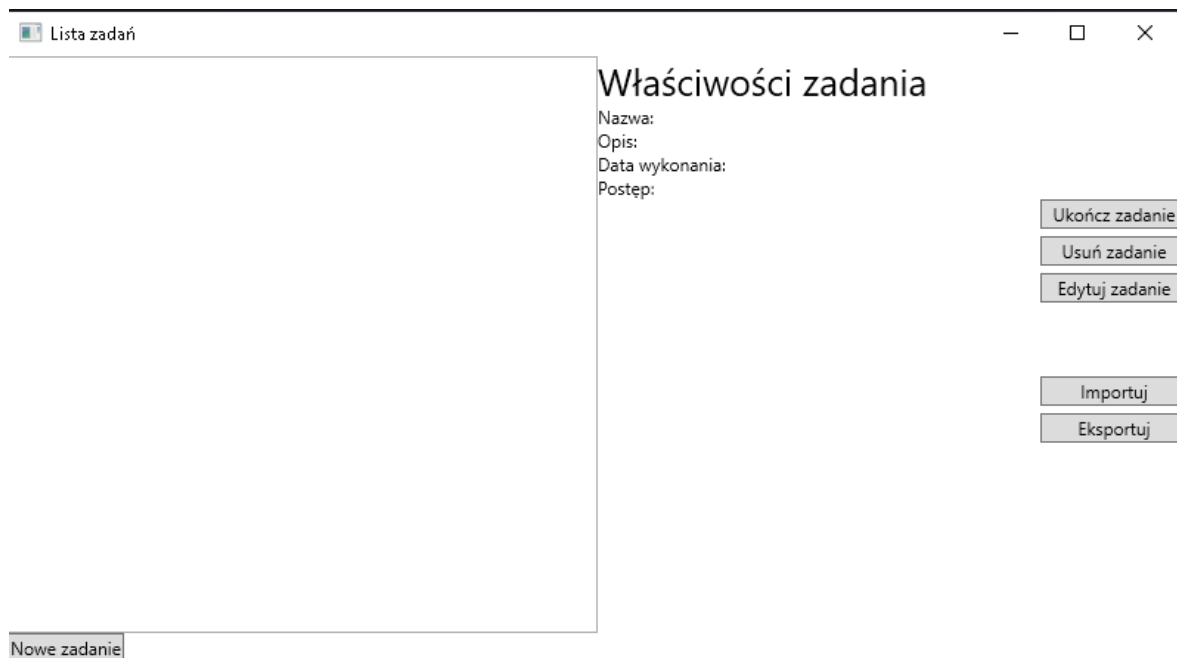
Cel projektu:

- Stworzenie prostej aplikacji typu „lista zadań” (To-Do List) z graficznym interfejsem użytkownika
- Możliwość dodawania, edytowania, usuwania i oznaczania zadań jako wykonane
- Funkcje eksportu i importu listy zadań z pliku tekstowego(.txt)

Wymagania funkcjonalne:

- Dodawanie nowego zadania
- Edycja istniejącego zadania
- Usuwanie zadania
- Oznaczanie zadania jako wykonane
- Eksport/import listy zadań
- Przejrzysty interfejs użytkownika

Interfejs użytkownika.



Zdjęcie nr.1 Interfejs użytkownika

Funkcje:

Funkcja Utworzenia nowego zadania.


```
public void NewTask_Click(object sender, RoutedEventArgs e)
{
    List.Visibility = Visibility.Hidden;
    New_Task.Visibility = Visibility.Visible;
    TaskName.Text = "";
    TaskDescription.Text = "";
}

//Dodanie zadania do listy
public void AddTask_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if(TaskName.Text == null)
        {
            MessageBox.Show("Brak nazwy");
            return;
        }
        string time = Convert.ToString(TaskDate.SelectedDate);
        time = time.Substring(0, 10);
        time = $"{time} {Convert.ToString(TaskHour.SelectedItem).Remove(0, 38)}:{Convert.ToString(TaskMinute.SelectedItem).Remove(0, 38)}:00";
        DateTime? stime = Convert.ToDateTime(time);
        Lista.Items.Add(new Task(TaskName.Text, TaskDescription.Text, stime));
        List.Visibility = Visibility.Visible;
        New_Task.Visibility = Visibility.Hidden;
    }
    catch (Exception)
    {
        MessageBox.Show("Nieprawidłowe lub brakujące dane!");
    }
}
```

Zdjęcie nr.2 Fragment kodu

Kod odpowiada za dodawanie nowego zadania w aplikacji To-Do List. Po kliknięciu „Dodaj nowe”, formularz pokazuje się, a pola są czyszczone. Po kliknięciu „Dodaj”, program sprawdza poprawność danych, tworzy nowe zadanie z nazwą, opisem i datą, a następnie dodaje je do listy. W przypadku błędu wyświetlany jest komunikat.

Wygląd formularza dodawania nowego zadania do listy.



The screenshot shows a Windows application window titled "Lista zadań". Inside, there is a form titled "Kreator zadania". The form contains the following fields and controls:

- Nazwa:** A text input field containing "zrób projekt".
- Opis:** A larger text input field also containing "zrób projekt".
- Data wykonania:** A date picker showing "05.05.2025" with a calendar icon.
- Godzina:** A dropdown menu showing "0".
- Minut:** A dropdown menu showing "00".
- Zapisz:** A button at the bottom of the form.

Zdjęcie nr.3 Interfejs użytkownika

Funkcja oznaczenia wykonania zadania.

```
private void FinishTask_Click(object sender, RoutedEventArgs e)
{
    if (selectedTask != null)
    {
        selectedTask.Done();
        Refresh_Properties();
        Lista.Items.Refresh();
    }
}
```

Zdjęcie nr.4 Fragment kodu

```
        Data.Text = $"Data wykonania: {selectedTask.DueDate}";
        if (selectedTask.IsDone)
        {
            Wykonane.Text = "Postęp: Ukończone";
        }
        else
        {
            Wykonane.Text = "Postęp: Nieukończone";
        }
    }
}
```

Zdjęcie nr.5 Fragment kodu

Kod odpowiada za oznacza wybrane zadanie jako ukończone i aktualizuje jego status w interfejsie.

Funkcja edycji zadania.

```
public void EditTask_Click(object sender, RoutedEventArgs e)
{
    if(selectedTask != null)
    {
        List.Visibility = Visibility.Hidden;
        Editor_Task.Visibility = Visibility.Visible;
        //Odczytanie i wprowadzenie obecnych danych do edytora zadań
        ETaskName.Text = selectedTask.Name;
        ETaskDescription.Text = selectedTask.Description;
        ETaskDate.SelectedDate = selectedTask.DueDate.Date;
        ETaskHour.SelectedItem = selectedTask.DueDate.Hour;
        ETaskMinute.SelectedItem = selectedTask.DueDate.Minute;
    }
}
```

Zdjęcie nr.6 Fragment kodu

```

public void SaveTask_Click(object sender, RoutedEventArgs e)
{
    //Sprawdza czy jest nowa nazwa zadania
    if (selectedTask.Name != ETaskName.Text)
    {
        selectedTask.Name = ETaskName.Text;
    }
    if (selectedTask.Description != ETaskDescription.Text)
    {
        selectedTask.Description = ETaskDescription.Text;
    }
    //Sprawdza czy zmienił się czas
    try
    {
        if (selectedTask.DueDate != Glue_Time(ETaskDate.SelectedDate, ETaskHour.SelectedItem, ETaskMinute.SelectedItem))
        {
            selectedTask.DueDate = Glue_Time(ETaskDate.SelectedDate, ETaskHour.SelectedItem, ETaskMinute.SelectedItem);
        }
    }
    catch(Exception)
    {
    }
    //Odświeżenie listy
    Lista.Items.Refresh();
    Refresh_Properties();
    List.Visibility = Visibility.Visible;
    Editor_Task.Visibility = Visibility.Hidden;
    MessageBox.Show($"Zadanie {selectedTask.Name} zostało zmienione");
}

```

Zdjęcie nr.7 Fragment kodu

Kod umożliwia edycję wybranego zadania. Po kliknięciu „Edytuj” wyświetlany jest formularz z aktualnymi danymi zadania. Po wprowadzeniu zmian i kliknięciu „Zapisz”, dane zadania są aktualizowane, lista odświeżana, a formularz ukrywany. Wyświetlany jest też komunikat o sukcesie.

Wygląd formularza edycji zadania.

Zdjęcie nr.8 Interfejs użytkownika

Funkcja usuwania zadania.

```
public void DeleteTask_Click(Object sender, RoutedEventArgs e)
{
    if(selectedTask != null)
    {
        Lista.Items.Remove(selectedTask);
        selectedTask = null;
        Refresh_Properties();
    }
    else
    {
        MessageBox.Show("Nie wybrano zadania do usunięcia");
    }
}
```

Zdjęcie nr.9 Fragment kodu

Metoda usuwa wybrane zadanie z listy. Jeśli żadne zadanie nie jest zaznaczone, wyświetlany jest komunikat o błędzie. Po usunięciu lista i szczegóły zadania są odświeżane.

Funkcja eksportu listy zadań.

```
//Eksport zadań
public void Eksport_Click(object sender, RoutedEventArgs e)
{
    //Sprawdza czy lista nie jest pusta
    if (Lista.Items.Count < 1)
    {
        MessageBox.Show("Lista zadań jest pusta!");
        return;
    }

    //Nowe okno dialogowe
    var dialog = new Microsoft.Win32.SaveFileDialog();
    dialog.FileName = "file"; // Default file name
    dialog.DefaultExt = ".txt"; // Default file extension
    dialog.Filter = "Text documents (*.txt)|*.txt"; // Filter files by extension

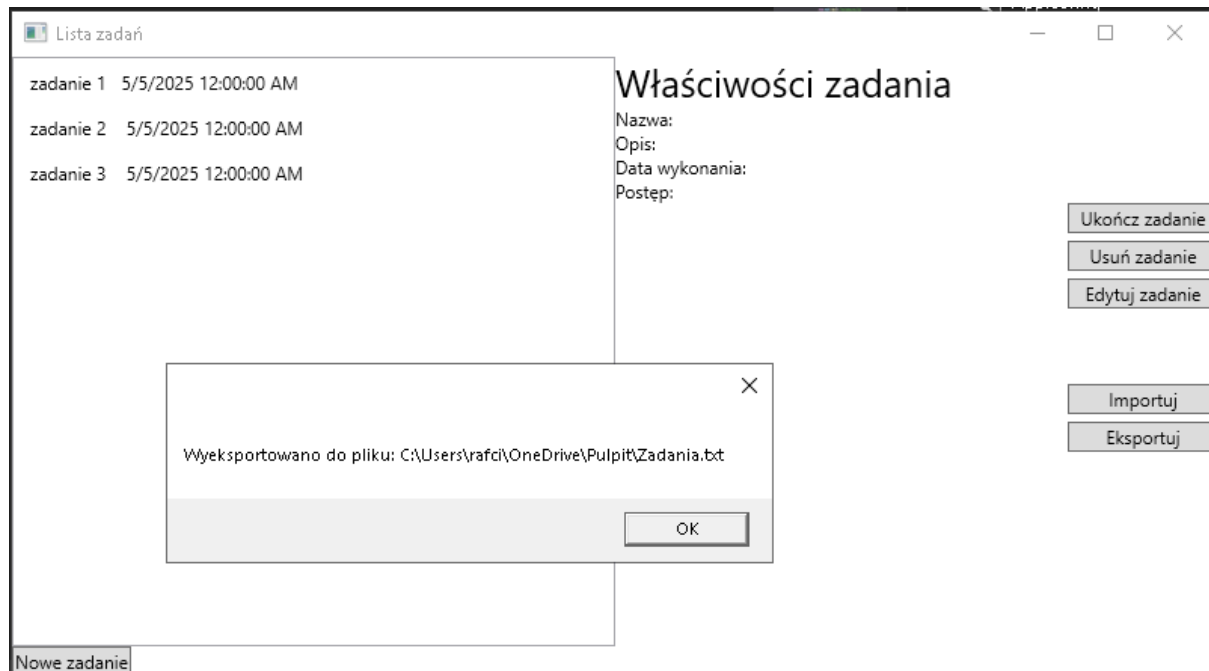
    // Show save file dialog box
    bool? result = dialog.ShowDialog();

    //Zapisz danych do pliku
    if (result == true)
    {
        string text = "";
        foreach (Task item in Lista.Items)
        {
            text += $"{item.Name};{item.Description};{item.DueDate};{item.IsDone}\n";
        }
        File.WriteAllText(dialog.FileName, text);
        MessageBox.Show($"Wyeksportowano do pliku: {dialog.FileName}");
    }
}
```

Zdjęcie nr.10 Fragment kodu

Eksportuje wszystkie zadania z listy do pliku .txt. Po wybraniu ścieżki zapisu, każde zadanie zostaje zapisane w formacie: Nazwa;Opis;Termin;Status. Jeśli lista jest pusta, wyświetlany jest komunikat o błędzie.

Komunikat o udanym eksporcie.



Zdjęcie nr.11 Interfejs użytkownika

Funkcja importu.

```
public void Import_Click(object sender, RoutedEventArgs e)
{
    var dialog = new Microsoft.Win32.OpenFileDialog();
    dialog.FileName = "file";
    dialog.DefaultExt = ".txt";
    dialog.Filter = "Text documents (.txt)|*.txt";

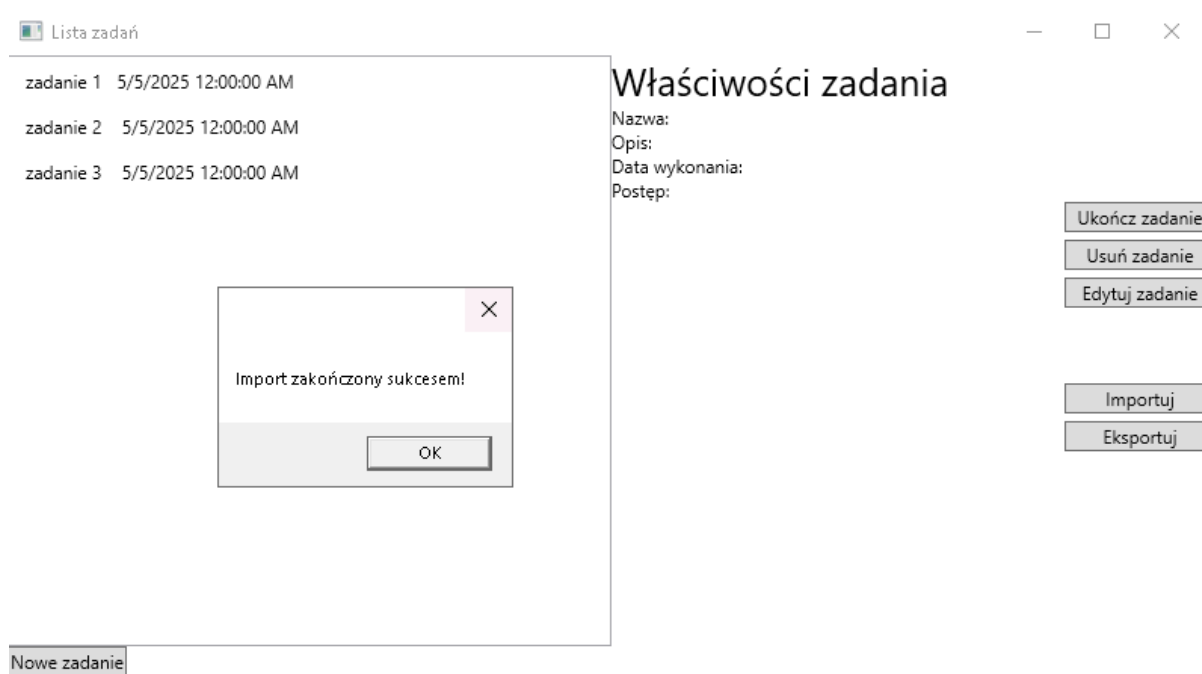
    // Show open file dialog box
    bool? result = dialog.ShowDialog();

    // Process open file dialog box results
    if (result == true)
    {
        string[] lines = File.ReadAllLines(dialog.FileName);
        if (lines.Length < 1)
        {
            MessageBox.Show("Plik jest pusty!");
            return;
        }
        try
        {
            foreach (string line in lines)
            {
                string[] itemdata = line.Split(';');
                Lista.Items.Add(new Task(itemdata[0], itemdata[1], Convert.ToDateTime(itemdata[2]), Convert.ToBoolean(itemdata[3])));
            }
            MessageBox.Show("Import zakończony sukcesem!");
        }
        catch
        {
            MessageBox.Show("Nastąpił nieoczekiwany błąd!");
            return;
        }
    }
}
```

Zdjęcie nr.12 Fragment kodu

Metoda Import_Click umożliwia wczytanie zadań z pliku .txt. Otwiera okno wyboru pliku, odczytuje dane linia po linii i dodaje zadania do listy. Jeśli plik jest pusty lub zawiera błędy, wyświetlany jest odpowiedni komunikat.

Komunikat o udanym imporcie.



Zdjęcie nr.13

Testy aplikacji:

Testy Wartości Bezwzględnych.

```
[TestClass]
public class Testy_Wartosci_Bezwglednych
{
    //Test funkcji Return_Bezwgledna
    [TestMethod]
    public void Test_Wartosci_Ujemnej()
    {
        int wynik = MainWindow.Return_Bezwgledna(-5);
        Assert.AreEqual(5, wynik); // spodziewana wartość to 5
    }

    [TestMethod]
    public void Test_Wartosci_Dodatniej()
    {
        int wynik = MainWindow.Return_Bezwgledna(5);
        Assert.AreEqual(5, wynik); // spodziewana wartość to 5
    }

    [TestMethod]
    public void Test_Duzej_Wartosci_Dodatniej()
    {
        int wynik = MainWindow.Return_Bezwgledna(5000);
        Assert.AreEqual(5000, wynik); // spodziewana wartość to 5000
    }

    [TestMethod]
    public void Test_Duzej_Wartosci_Ujemnej()
    {
        int wynik = MainWindow.Return_Bezwgledna(-5000);
        Assert.AreEqual(5000, wynik); // spodziewana wartość to 5000
    }

    [TestMethod]
    public void Test_Wartosci_Zero()
    {
        int wynik = MainWindow.Return_Bezwgledna(0);
        Assert.AreEqual(0, wynik); // spodziewana wartość to 0
    }
}
```

Zdjęcie nr.14 Fragment testów

Kod przedstawia klasę testową `Testy_Wartosci_Bezwglednych`, która zawiera zestaw testów jednostkowych dla funkcji `Return_Bezwgledna` z klasy `MainWindow`. Celem testów jest sprawdzenie, czy metoda poprawnie zwraca wartość bezwzględną liczby całkowitej. Przetestowane zostały przypadki dla liczb dodatnich, ujemnych, dużych wartości oraz zera. Do testowania wykorzystano framework `MSTest`.

Testy dat.

```
[TestClass]
public class Testy_Dat
{
    //Testy funkcji Glue_Time
    [TestMethod]
    public void Test_Daty()
    {
        DateTime wynik = MainWindow.Glue_Time(Convert.ToDateTime("22/04/2025 00:00:00"), "7", "00");
        Assert.AreEqual(Convert.ToDateTime("22/04/2025 07:00:00"), wynik); // spodziewana wartość to 22/04/2025 07:00:00
    }
    [TestMethod]
    public void Test_Przyszlej_Daty_2()
    {
        DateTime wynik = MainWindow.Glue_Time(Convert.ToDateTime("22/04/2077 00:00:00"), "9", "45");
        Assert.AreEqual(Convert.ToDateTime("22/04/2077 09:45:00"), wynik); // spodziewana wartość to 22/04/2077 09:45:00
    }
    [TestMethod]
    public void Test_Daty_3()
    {
        DateTime wynik = MainWindow.Glue_Time(Convert.ToDateTime("22/05/2025 00:00:00"), "19", "45");
        Assert.AreEqual(Convert.ToDateTime("22/05/2025 19:45:00"), wynik); // spodziewana wartość to 22/05/2025 19:45:00
    }
    [TestMethod]
    public void Test_Bardzo_Starej_Daty()
    {
        DateTime wynik = MainWindow.Glue_Time(Convert.ToDateTime("22/05/1700 00:00:00"), "23", "59");
        Assert.AreEqual(Convert.ToDateTime("22/05/1700 23:59:00"), wynik); // spodziewana wartość to 22/05/1700 23:59:00
    }
    [TestMethod]
    public void Test_Dawnej_Daty()
    {
        DateTime wynik = MainWindow.Glue_Time(Convert.ToDateTime("2/04/2005 00:00:00"), "21", "37");
        Assert.AreEqual(Convert.ToDateTime("2/04/2005 21:37:00"), wynik); // spodziewana wartość to 2/04/2005 21:37:00
    }
}
[TestClass]
```

Zdjęcie nr.15 Fragment testów

Testy sprawdzają poprawność działania funkcji dla różnych przypadków, w tym dat współczesnych, przyszłych oraz bardzo dawnych.

Testy tekstu.

```
public class Testy_Tekstu
{
    [TestMethod]
    public void Test_Tekstu()
    {
        string wynik = MainWindow.MakeLines("Lena poszła po zakupy", 6);
        Assert.AreEqual("Lena\nposzła\npo\nzakupy", wynik);
    }

    [TestMethod]
    public void Test_Tekstu_2()
    {
        string wynik = MainWindow.MakeLines("Lena poszła po zakupy", 9);
        Assert.AreEqual("Lena\nposzła po\nzakupy", wynik);
    }

    [TestMethod]
    public void Test_Tekstu_3()
    {
        string wynik = MainWindow.MakeLines("Lena poszła po zakupy", 11);
        Assert.AreEqual("Lena poszła\npo zakupy", wynik);
    }

    [TestMethod]
    public void Test_Tekstu_4()
    {
        string wynik = MainWindow.MakeLines("Renata poszła po zakupy", 11);
        Assert.AreEqual("Renata\nposzła po\nzakupy", wynik);
    }

    [TestMethod]
    public void Test_Tekstu_5()
    {
        string wynik = MainWindow.MakeLines("Renata jednak nie poszła po zakupy", 14);
        Assert.AreEqual("Renata jednak\nnie poszła po\nzakupy", wynik);
    }
}
```

Zdjęcie nr.16 Fragment testów

Testy sprawdzają poprawność działania algorytmu dla różnych długości maksymalnych linii oraz zróżnicowanych tekstów.

Testy opisu.

```
[TestClass]
public class Testy_Opisu
{
    [TestMethod]
    public void Test_Opisu()
    {
        Task task = new Task("Podlewanie", "Podlanie roślin m.in. truskawek, pomidorów, ziemniaków i marchewek", Convert.ToDateTime("9/05/2025 18:00:00"));
        string wynik = MainWindow.MakeLines(task);
        Assert.AreEqual("Podlanie roślin m.in. truskawek, pomidorów, ziemniaków i marchewek", wynik);
    }

    [TestMethod]
    public void Test_Opisu_2()
    {
        Task task = new Task("Podlewanie", "Podlanie roślin m.in. truskawek, pomidorów, ziemniaków i marchewek", Convert.ToDateTime("9/05/2025 18:00:00"));
        string wynik = MainWindow.MakeLines(task, 30);
        Assert.AreEqual("Podlanie roślin m.in.\ntruskawek, pomidorów,\nziemniaków i marchewek", wynik);
    }

    [TestMethod]
    public void Test_Opisu_3()
    {
        Task task = new Task("Podlewanie", "Podlanie roślin m.in. truskawek, pomidorów, ziemniaków i marchewek", Convert.ToDateTime("9/05/2025 18:00:00"));
        string wynik = MainWindow.MakeLines(task, 11);
        Assert.AreEqual("Podlanie\nroślin\nm.in.\ntruskawek,\npomidorów,\nziemniaków\ni marchewek", wynik);
    }

    [TestMethod]
    public void Test_Opisu_4()
    {
        Task task = new Task("Zakupy", "Lista zakupów: -chleb -masło -mąka -jabłko", Convert.ToDateTime("9/05/2025 18:00:00"));
        string wynik = MainWindow.MakeLines(task);
        Assert.AreEqual("Lista zakupów:\n- chleb\n- masło\n- mąka\n- jabłko", wynik);
    }

    [TestMethod]
    public void Test_Opisu_5()
    {
        Task task = new Task("Zakupy", "Lista zakupów: -chleb - masło - mąka -klej do płytek -beton ", Convert.ToDateTime("9/05/2025 18:00:00"));
        string wynik = MainWindow.MakeLines(task);
        Assert.AreEqual("Lista zakupów:\n- chleb\n- masło\n- mąka\n- klej do płytek\n- beton", wynik);
    }
}
```

Zdjęcie nr.17 Fragment testów

Testy mają na celu weryfikację, czy metoda MakeLines prawidłowo dzieli długi tekst na linie, uwzględniając różne długości tekstów oraz wymagania formatowania, takie jak lista punktowana.

Podział obowiązków

Diagramy

- Czynności: Rafał Złotkowski
- Klas: Konrad Wierzbicki
- Przypadków użycia: Konrad Wierzbicki
- Sekwencji :Michał Truszkowski

Aplikacja + testy: Michał Truszkowski

Sprawozdanie: Rafał Złotkowski