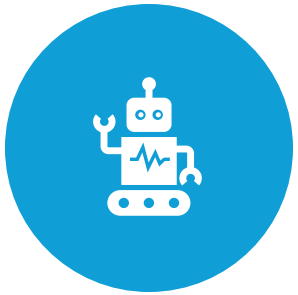
# .NET MAUI w aplikacjach Android

Zbigniew Kluczkowski

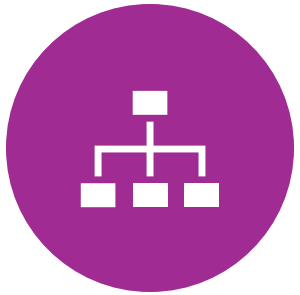
# Co to jest .NET MAUI?

Skrót od **Multi-platform App UI**

Framework firmy **Microsoft**



Pozwala tworzyć **aplikacje mobilne, desktopowe i webowe** z użyciem jednego kodu źródłowego



Następca **Xamarin.Forms**

# Dlaczego warto używać MAUI do aplikacji Android?

**Wieloplatformowość**: Jeden projekt = aplikacje na Android, iOS, Windows i macOS

**Kod współdzielony**: Logika aplikacji, modele danych, interfejsy użytkownika w jednym miejscu

**Wydajność**: Bezpośrednie mapowanie komponentów MAUI na natywne kontrolki Androida

**Wsparcie Microsoft**: Aktualizacje, integracja z Visual Studio

# Architektura aplikacji MAUI

Projekt zawiera katalogi:

- Platforms/Android – specyficzne ustawienia dla Androida
- Views, Models, ViewModels – zgodnie z wzorcem MVVM
- Plik MainPage.xaml – główny interfejs aplikacji
- App.xaml.cs – start aplikacji

# Wzorzec MVVM

MVVM (Model-View-ViewModel) ma za zadanie ułatwić tworzenie ekranów aplikacji poprzez zastosowanie podziału odpowiedzialności na trzy różne warstwy: widoku (View), widoku modelu (ViewModel) oraz modelu (Model).

Warstwa widoku odpowiedzialna jest za prezentacje danych, stanu systemu i bieżących operacji w interfejsie graficznym, a także za inicjalizację i wiązanie ViewModel z elementami widoku. Warstwa widoku modelu zajmuje się dostarczaniem danych modelu dla warstwy widoku oraz podejmowaniem akcji na rzecz wywołanego zdarzenia z widoku.

Natomiast warstwa modelu odpowiada za logikę biznesową, czyli przetwarzanie, przechowywanie, modyfikacje oraz dostarczanie oczekiwanych danych do widoku modelu.

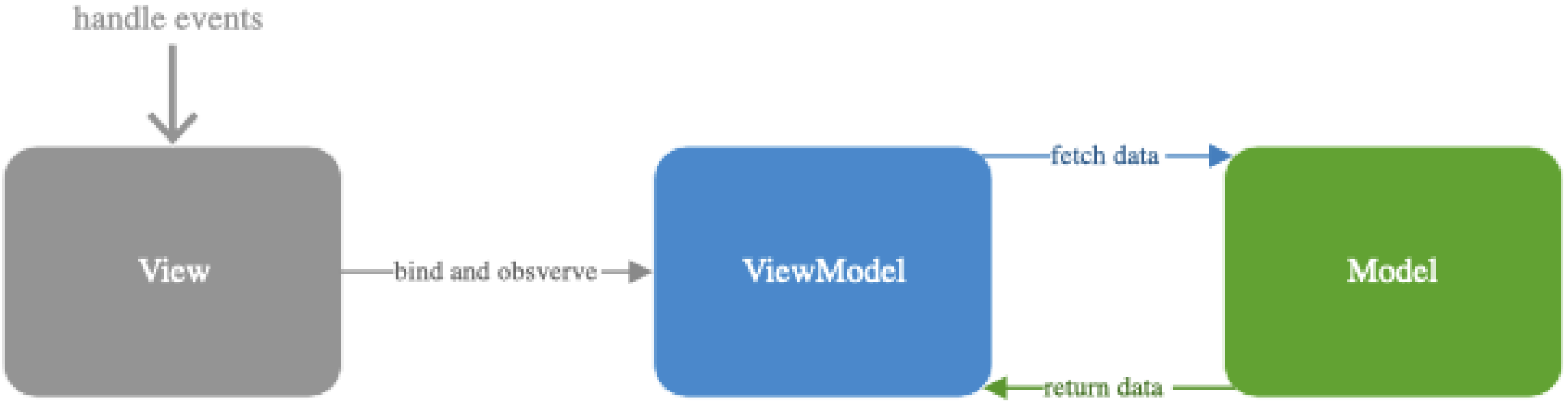
Dzięki zastosowaniu strategii wiązania danych (data binding) w warstwie widoku minimalizowana jest jego logika, kod staje się bardziej uporządkowany i otwarty na modyfikacje, a przeprowadzenie testów łatwiejsze.

# Implementacja MVVM

Warstwa widoku View tworzy instancje widoku modelu ViewModel oraz wiąże go z widokiem za pomocą mechanizmu data binding.

Klasa ViewModel przechowuje wszystkie niezbędne dane dla warstwy widoku, które otrzymuje z modelu.

Dzięki zastosowaniu wzorca Obserwator na polach widoku modelu każda zmiana stanu jest odnotowana przez widok. Warstwa modelu Model dostarcza implementacje pobierania, modyfikacji i przetwarzania danych z repozytoriów.



# Wzorzec „Obserwator”

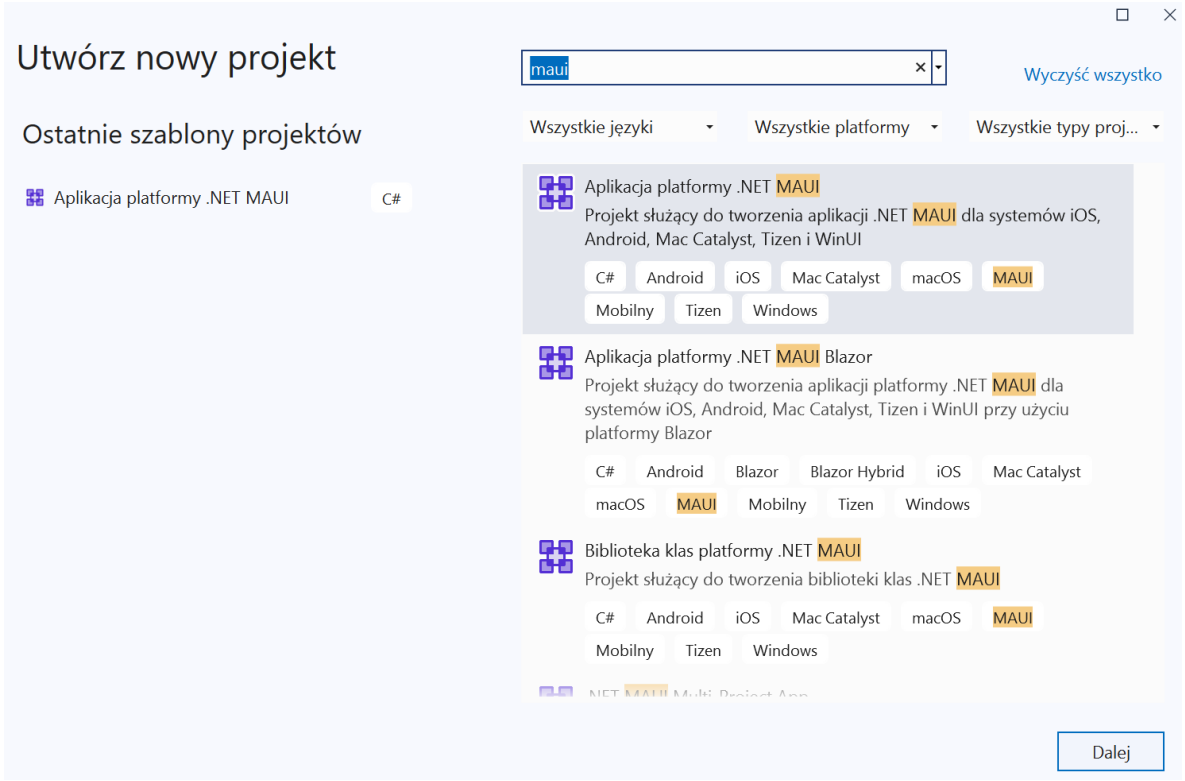
Idea wzorca MVVM opiera się przede wszystkim na obserwowaniu przez warstwę widoku (wzorzec Obserwator) zmieniających się danych w warstwie widoku modelu i reagowanie na zmiany poprzez mechanizm wiązania danych. Ze względu na różnorodność środowisk i technologii realizacja wzorca MVVM może zostać uzyskana na wiele sposobów.

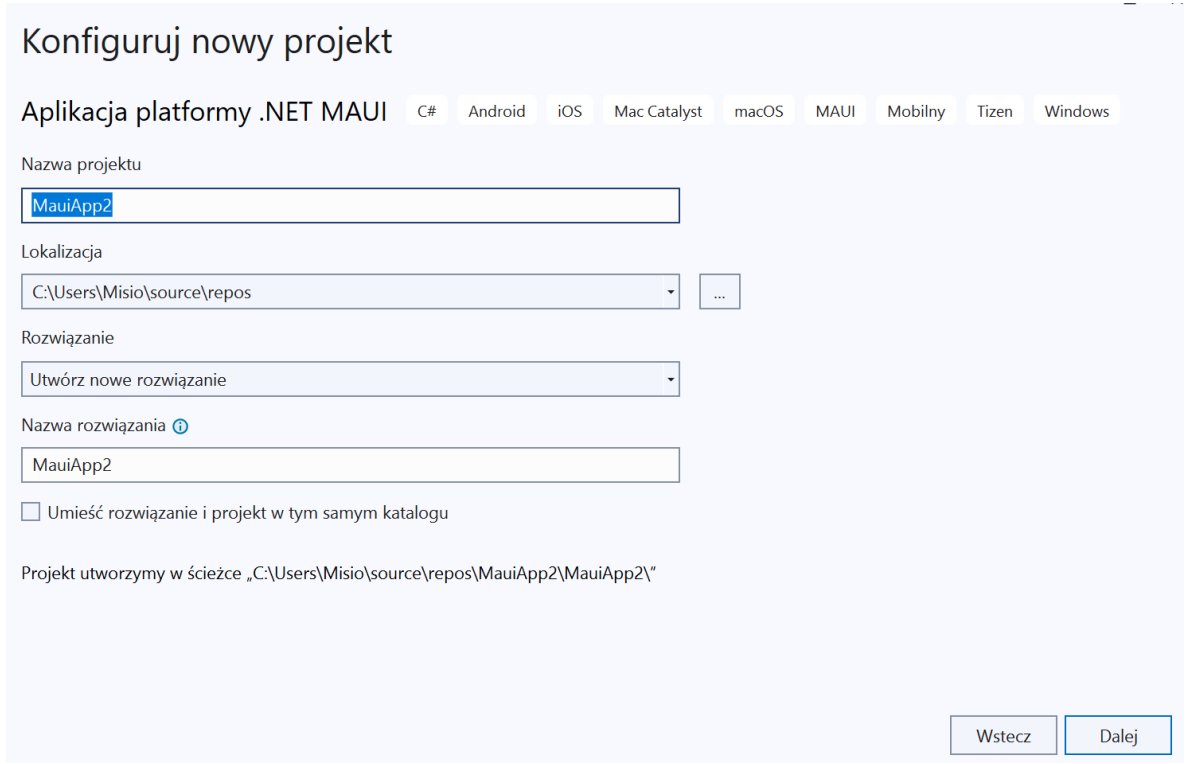
# Interfejs użytkownika (XAML)

```
<VerticalStackLayout>
   <Label Text="Witaj w aplikacji MAUI!"
          FontSize="24"
          HorizontalOptions="Center" />
   <Button Text="Kliknij mnie"
           Clicked="OnButtonClicked" />
</VerticalStackLayout>
```

# Obsługa zdarzeń i logiki (C#)

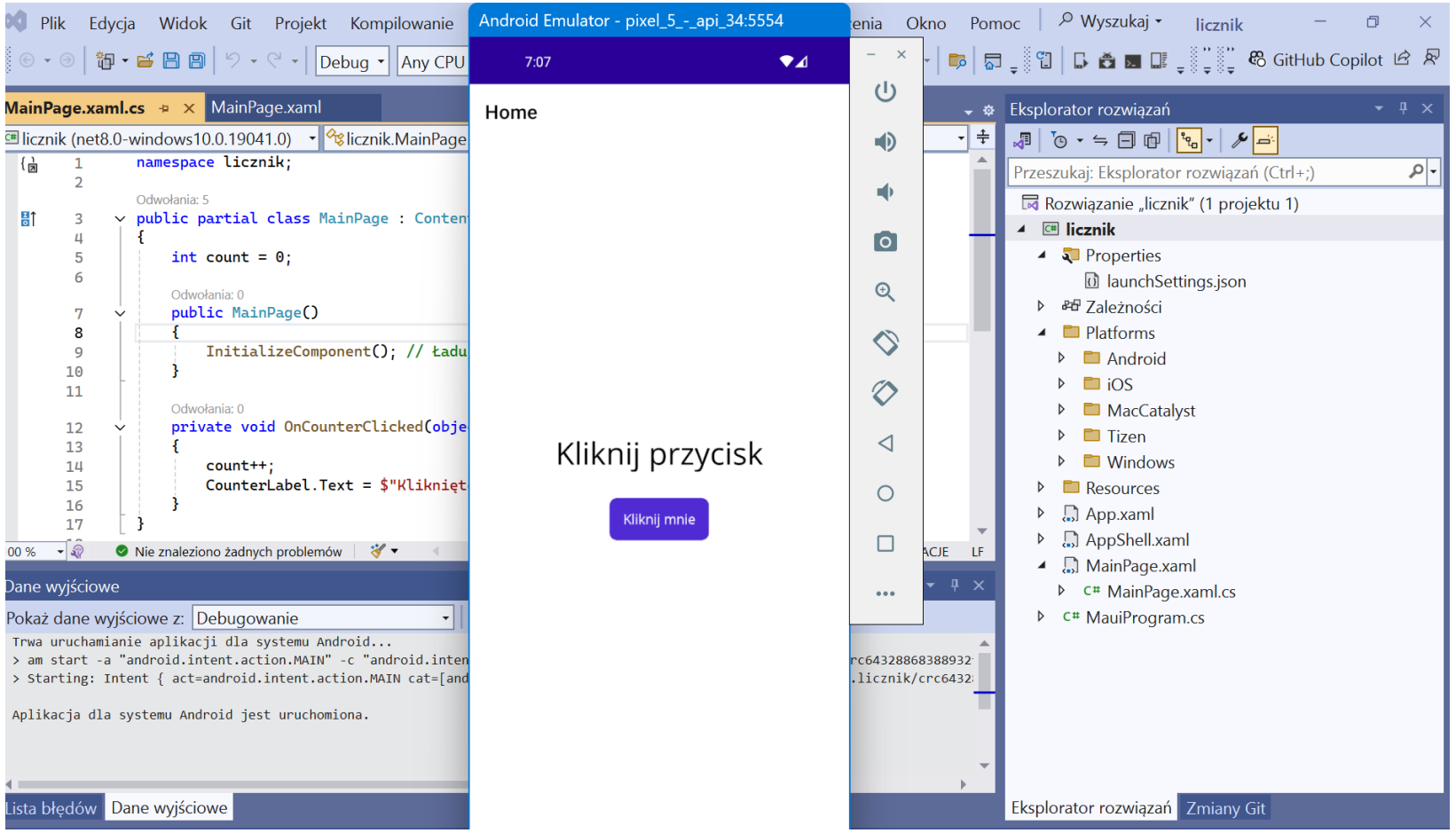
```
private void OnButtonClicked(object sender, EventArgs e)
{
    counter++;
    CounterLabel.Text = $"Kliknięto {counter} razy!";
}
```

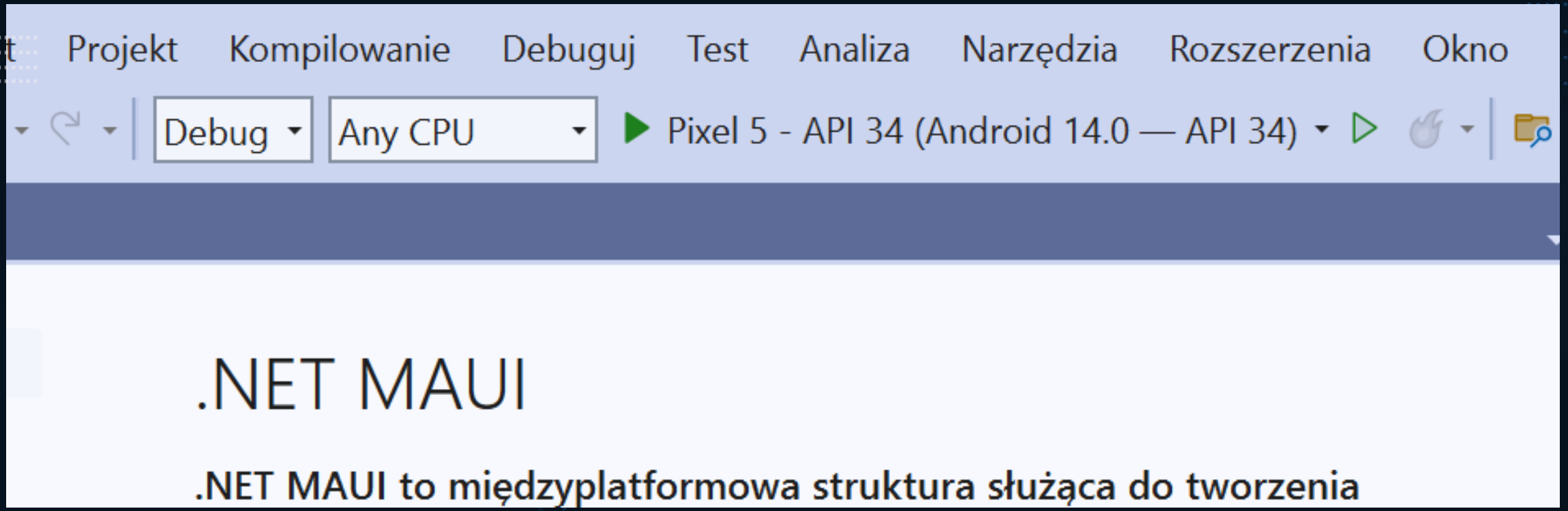



# Instalacja wewnątrz Visual Studio

# Struktura projektu






# Konfiguracja emulatora Android

Jeżeli nie ma na dysku komputera Android Studio to należy pobrać telefon i API.

Jeżeli jest już skonfigurowany telefon dla Android Studio to MAUI znajdzie konfigurację i ją uruchomi w środowisku Visual Studio.

# Przykład aplikacji Android w MAUI

Aplikacja "Licznik" – prosta aplikacja, która zlicza kliknięcia

UI i logika współdzielona

Działa natywnie na Androidzie – instalacja jak typowa aplikacja .apk

MainPage.xaml.cs | MainPage.xaml

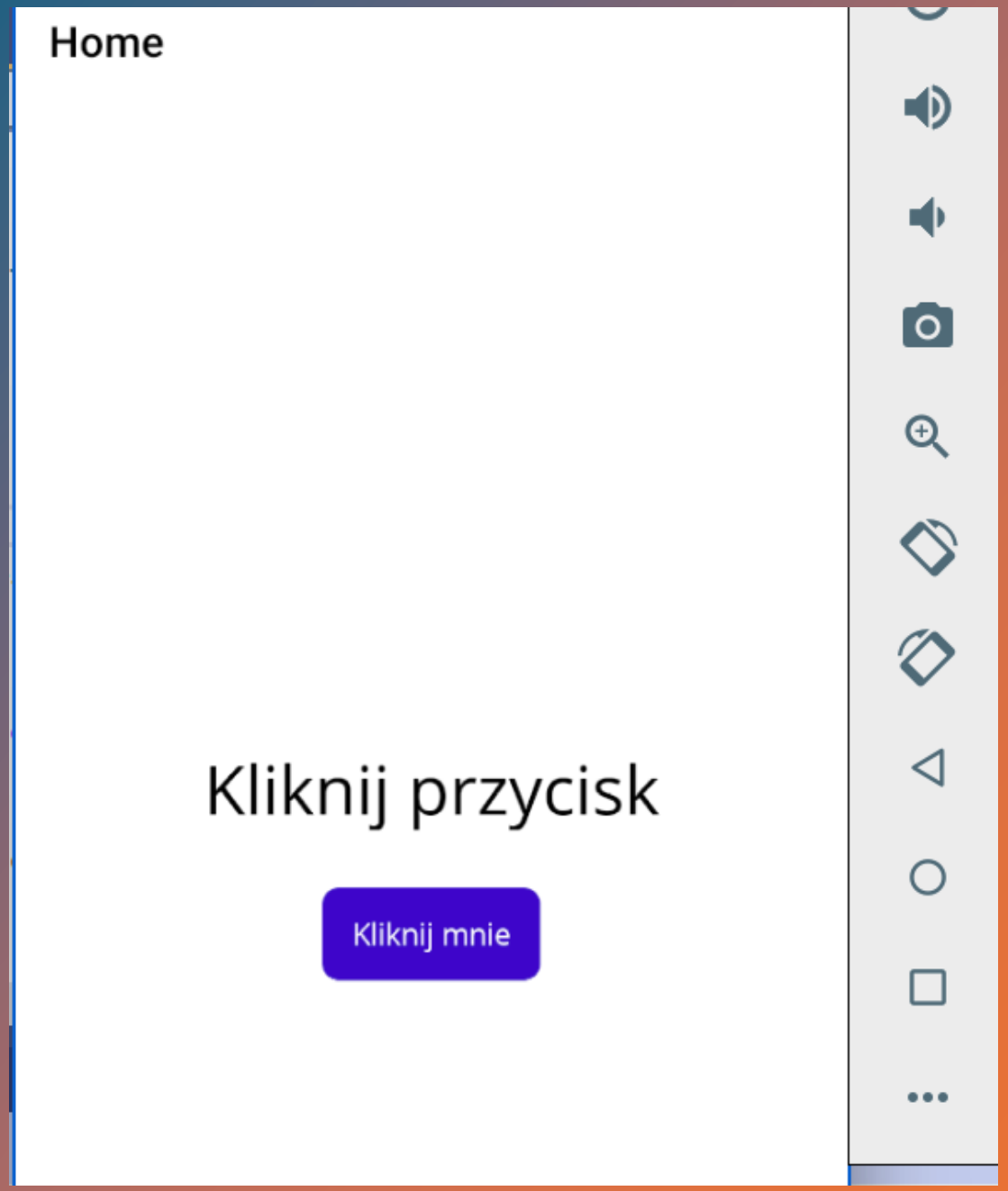
licznik (net8.0-windows10.0.19041.0) | ContentPage | x:Class

```
1   <?xml version="1.0" encoding="utf-8" ?>
2   <ContentPage
3       xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
4       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
5       x:Class="licznik.MainPage">
6
7       <VerticalStackLayout
8           Spacing="25"
9           Padding="30"
10          VerticalOptions="Center">
11
12          <Label
13              x:Name="CounterLabel"
14              Text="Kliknij przycisk"
15              FontSize="32"
16              HorizontalOptions="Center" />
17
18          <Button
19              Text="Kliknij mnie"
20              Clicked="OnCounterClicked"
21              HorizontalOptions="Center" />
22
```

MainPage.xaml.cs | MainPage.xaml

licznik (net8.0-windows10.0.19041.0) | licznik.MainPage | count

```
1   namespace licznik;
2
    Odwołania: 5
3   public partial class MainPage : ContentPage
4   {
5       int count = 0;
6
        Odwołania: 0
7       public MainPage()
8       {
9           InitializeComponent(); // Ładuje interfejs XAML
10      }
11
        Odwołania: 0
12      private void OnCounterClicked(object sender, EventArgs e)
13      {
14          count++;
15          CounterLabel.Text = $"Kliknięto: {count} razy";
16      }
17  }
18
```




# Efekt działania programu

Przed Wami prosty program. Użytkownik naciska przycisk i po każdym naciśnięciu zwiększa się licznik kliknięć, którego wartość jest wyświetlona na etykiecie.

Co ciekawe, wbudowany emulator działa szybciej i sprawniej niż ten z Android Studio.

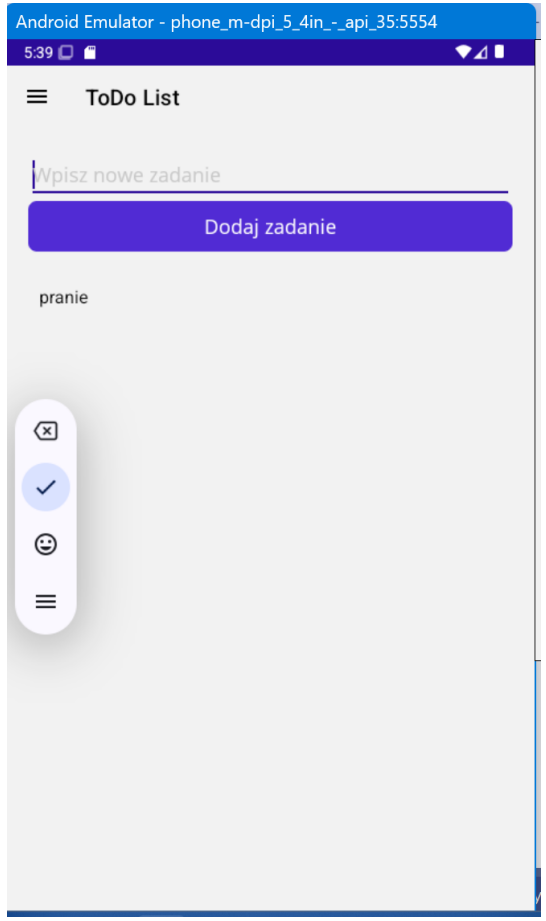
# Rozwój aplikacji

**Możesz dodać pliki do struktury i dopisać do nich akcję. Na przykład zdjęcie w Resources a następnie właściwość BuildAction.**

Można dodać do XML:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        x:Class="TwojaApp.MainPage"
        Title="Formularz ze zdjęciem w tle">

  <Grid>
    <!-- Tło -->
    <Image Source="tlo.jpg"
        Aspect="AspectFill"
        Opacity="0.3"
        IsOpaque="True"
        HorizontalOptions="Fill"
        VerticalOptions="Fill" />

    <!-- Treść strony -->
    <ScrollView>
      <VerticalStackLayout Padding="20" Spacing="20">
        <Label Text="Przykład formularza" FontSize="24" TextColor="Black"/>
        <Entry Placeholder="Wprowadź dane" />
        <Button Text="Wyślij" />
      </VerticalStackLayout>
    </ScrollView>
  </Grid>
</ContentPage>
```

# ToDo lista – prawdziwy „egzaminacyjny hicior”




MainPage.xaml

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="listazadan.Pages.MainPage"
             Title="ToDo List">
  <VerticalStackLayout Padding="20">
    <Entry x:Name="taskEntry" Placeholder="Wpisz nowe zadanie" />
    <Button Text="Dodaj zadanie" Clicked="OnAddTaskClicked" />

    <CollectionView x:Name="taskListView" Margin="0,20,0,0">
      <CollectionView.ItemTemplate>
        <DataTemplate>
          <SwipeView>
            <SwipeView.RightItems>
              <SwipeItems>
                <SwipeItem Text="Usuń" BackgroundColor="Red" Invoked="OnDeleteTask"/>
              </SwipeItems>
            </SwipeView.RightItems>
            <Grid Padding="10">
              <Label Text="{Binding}" FontSize="16" />
            </Grid>
          </SwipeView>
        </DataTemplate>
      </CollectionView.ItemTemplate>
    </CollectionView>
  </VerticalStackLayout>
</ContentPage>
```

# ToDo lista – prawdziwy „egzaminacyjny hicior”

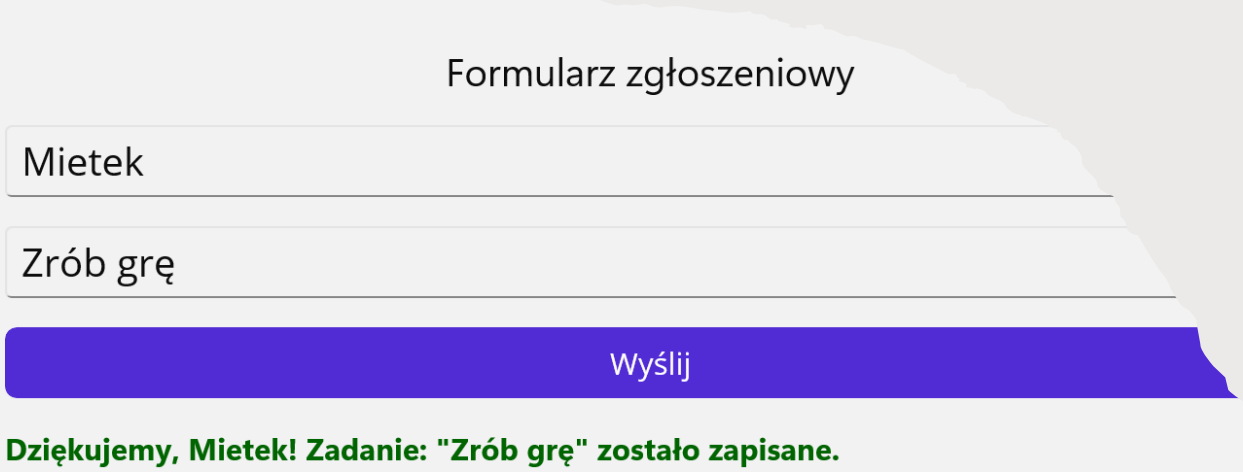
```
using System.Collections.ObjectModel;

namespace listazadan.Pages;

public partial class MainPage : ContentPage
{
    ObservableCollection<string> tasks = new ObservableCollection<string>();

    public MainPage()
    {
        InitializeComponent();
        taskListView.ItemsSource = tasks;
    }

    private void OnAddTaskClicked(object sender, EventArgs e)
    {
        if (!string.IsNullOrWhiteSpace(taskEntry.Text))
        {
            tasks.Add(taskEntry.Text.Trim());
            taskEntry.Text = string.Empty;
        }
    }

    private void OnDeleteTask(object sender, EventArgs e)
    {
        if (sender is SwipeItem swipeItem && swipeItem.BindingContext is string task)
        {
            tasks.Remove(task);
        }
    }
}
```

Częsty błąd!

Uważaj na namespace i umieszczenie najważniejszych plików projektu. W większości przypadków będą zlokalizowane w folderze Pages i przy deklaracji tzw. „przestrzeni” należy dokładnie określić miejsce.

Jeżeli tego nie zrobisz to będziesz świadkiem „festiwalu błędów” i poszukiwania plików z C#, w których znajduje się dana „przestrzeń”.

# Formularz – „pewniak egzaminacyjny”




```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       x:Class="prostyform.Pages.MainPage"
       Title="Formularz">
  <VerticalStackLayout Padding="30" Spacing="20">
    <Label Text="Formularz zgłoszeniowy"
         FontSize="24"
         HorizontalOptions="Center" />
    <Entry x:Name="imieEntry"
         Placeholder="Wpisz swoje imię" />
    <Entry x:Name="zadanieEntry"
         Placeholder="Wpisz treść zadania" />
    <Button Text="Wyślij"
          Clicked="OnSendClicked" />
    <Label x:Name="wynikLabel"
         Text=""
         TextColor="DarkGreen"
         FontSize="18"
         FontAttributes="Bold" />
  </VerticalStackLayout>
</ContentPage>
```

# Formularz – „pewniak egzaminacyjny”


```
namespace prostyform.Pages;

public partial class MainPage : ContentPage

{

  public MainPage()

  {

    InitializeComponent();

  }

  private void OnSendClicked(object sender, EventArgs e)

  {

    string imie = imieEntry.Text;

    string zadanie = zadanieEntry.Text;

    if (!string.IsNullOrWhiteSpace(imie) && !string.IsNullOrWhiteSpace(zadanie))

    {

      wynikLabel.Text = $"Dziękujemy, {imie}! Zadanie: \"{zadanie}\" zostało zapisane.";

    }

    else

    {

      wynikLabel.Text = "Uzupełnij wszystkie pola!";

    }

  }

}
```

# Zalety i wady MAUI

| Zalety | Wady |
| --- | --- |
| **Szybkie prototypowanie** | **Większy rozmiar aplikacji (na starcie)** |
| **Wspólny interfejs użytkownika** | **Czasami konieczność pisania kodu natywnego dla specyficznych funkcji Androida** |
| **Wbudowane wsparcie dla funkcji Androida (kamera, czujniki, itd.)** | **MAUI nadal się rozwija – mogą wystąpić błędy lub braki w dokumentacji** |
| **Dostęp do natywnych API przez DependencyService lub Platform API** | |

# Problemy z MAUI?

Nie jest to doskonałe narzędzie. Ma problemy z emulatorem oraz z pamięcią, przez co często kilkukrotnie trzeba czyścić projekt.

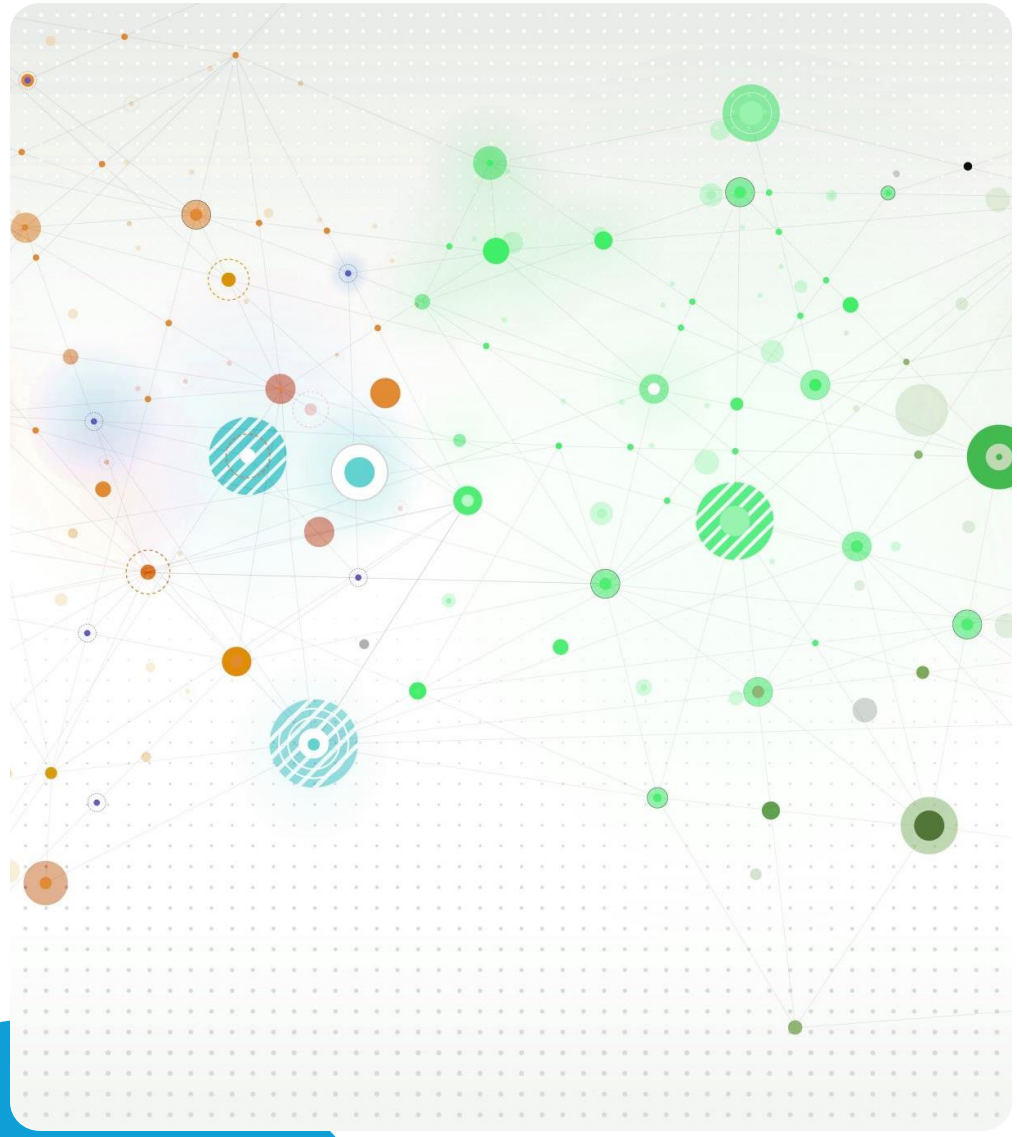
**Gdzie szukać przyczyn?**

Pliki z ustawieniami, zależnościami oraz App.xaml. W prostych projektach problemem jest Shell (narzędzie, które podobno ułatwia nawigację po projekcie). Wystarczy usunąć jego pozostałości w kodzie.

# Czy warto?

Według autorów tego „wynalazku” są to typowe „choroby wieku dziecięcego”, które mają zostać naprawione.

To rozwiązanie dla tych, dla których Java jest zbyt trudna lub nie mogą odnaleźć się w strukturze plików projektu Android Studio.



# Pytania:

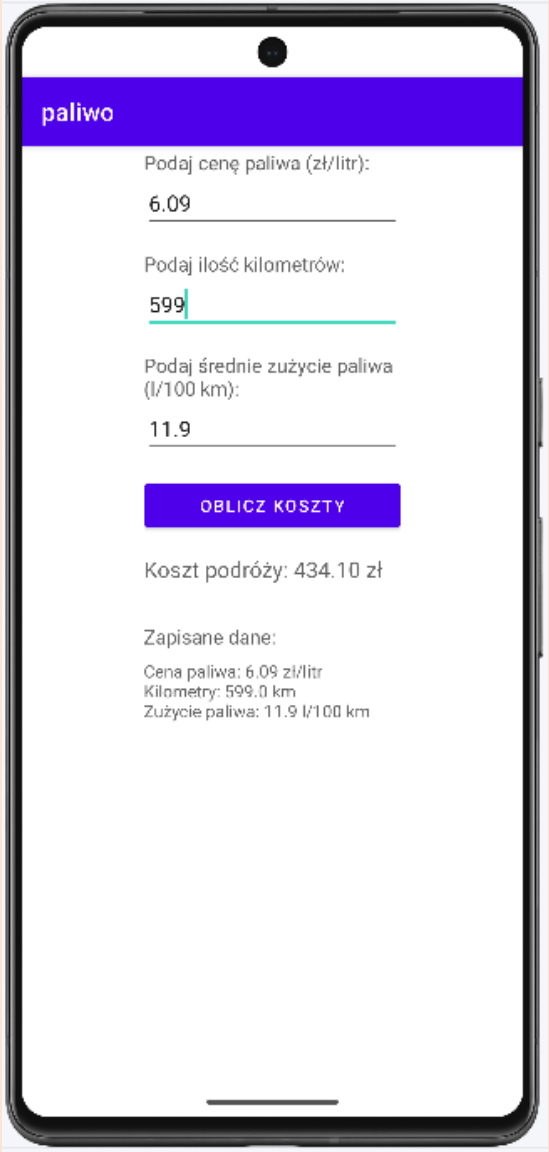
1. Na jakim modelu działa aplikacja .NET MAUI?
2. W jakim języku należy napisać pliki opisowe interfejsu a w jakim wykonywalne?
3. Jaka jest minimalna ilość plików, które pozwalają wygenerować projekt?
4. Czy można projekt z .NET MAUI przenieść do Android Studio?
5. Czy w .NET MAUI możemy skorzystać z urządzenia zewnętrznego czy tylko z emulatora?

# Zadania do samodzielnego wykonania

## Zadanie 1

**Napisz aplikację, która po wpisaniu danych przedstawionych na zdjęciu wyliczy koszt podróży.**

**Wszystkie dane pochodzą od użytkownika. Aplikacja wykorzystuje SharedPreferencies do zapisu danych. Dane są nadpisywane.**

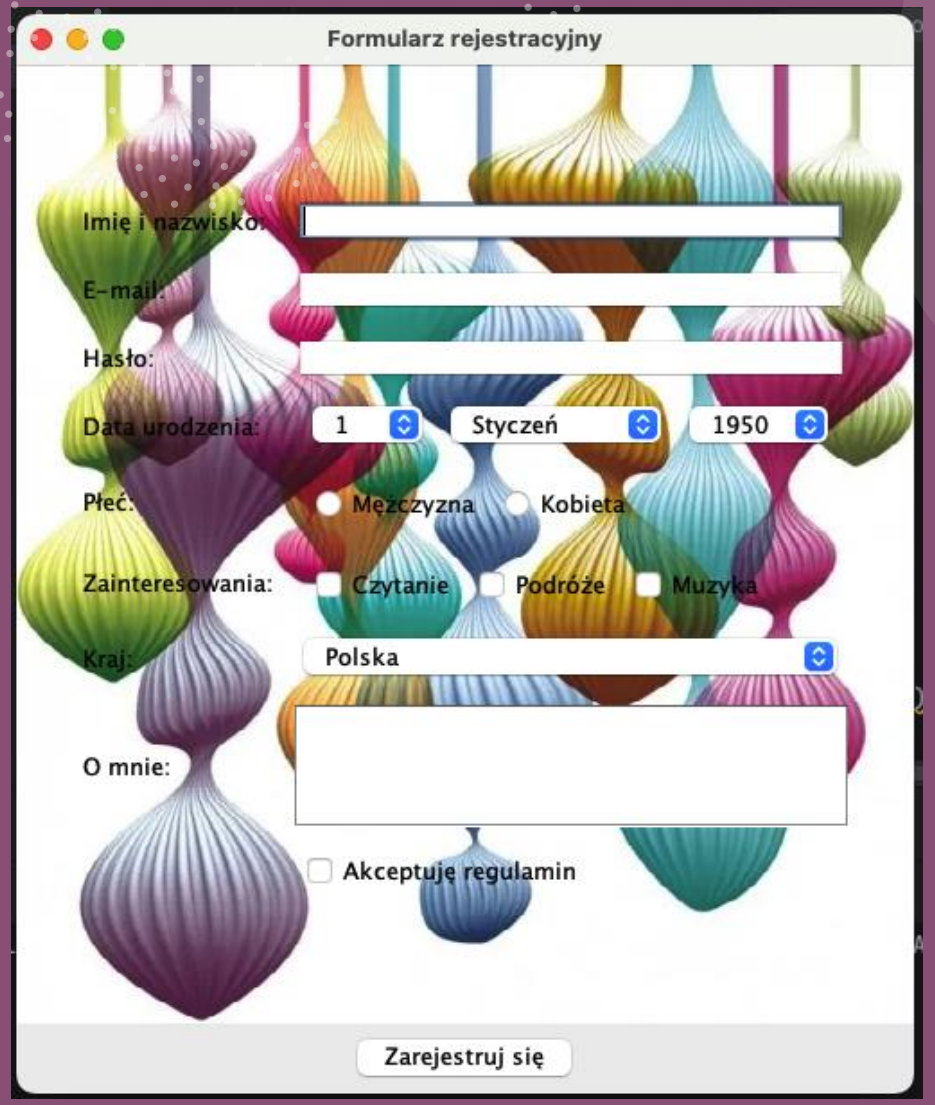


# Zadania do samodzielnego wykonania

Zadanie 2

Utwórz formularz według wzoru zamieszczonego obok. Posiada:

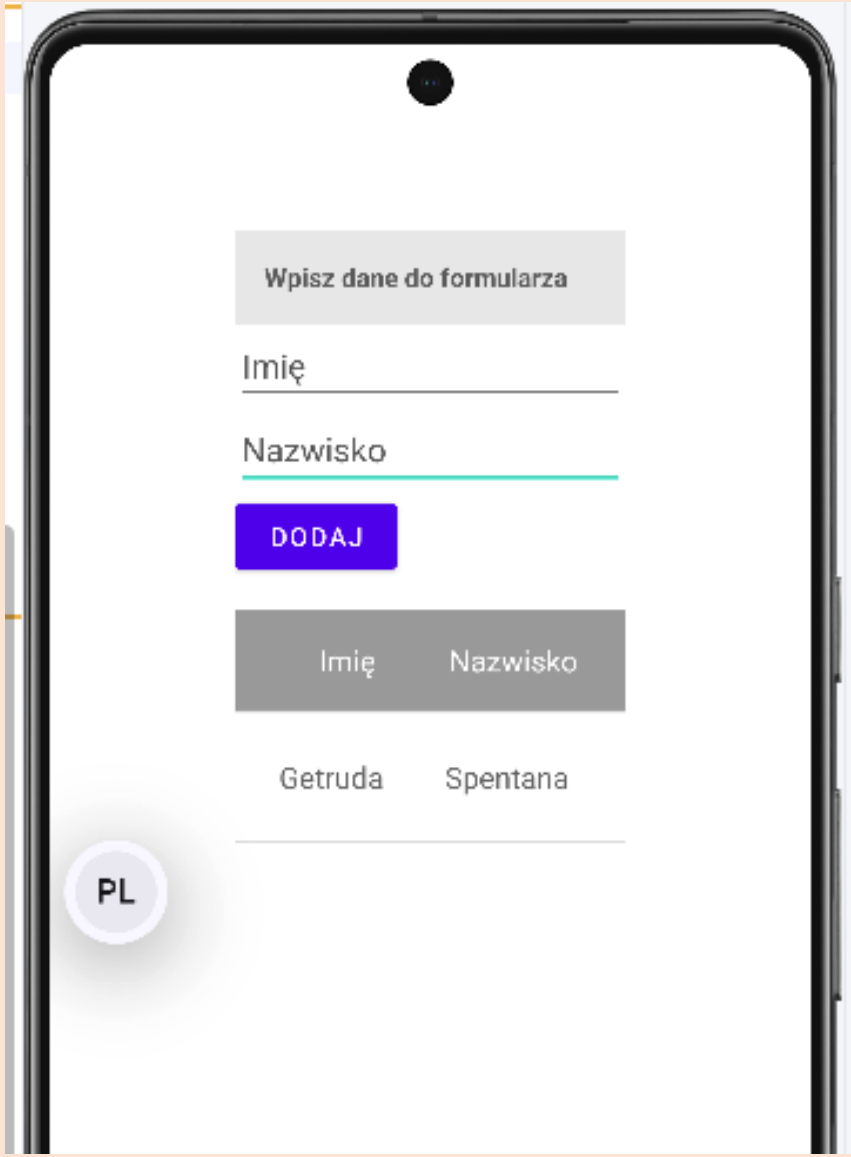
- Walidację pola „E-mail” (sprawdza, czy jest znak „@”)
- Walidację hasła (czy ma minimum 8 znaków)
- Elementy ułożone za pomocą GRID
- Estetyczne tło w postaci obrazka
- Zapis danych w pliku tekstowym



# Zadania do samodzielnego wykonania

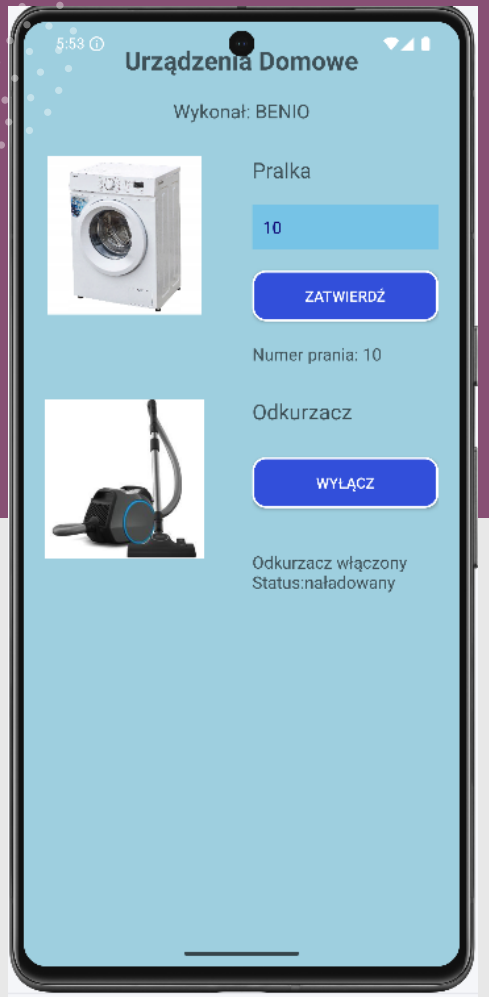
## Zadanie 3

Napisz aplikację według wzoru. Projekt zawiera formularz, etykiety oraz tabelę z nagłówkiem, w której zapisywane są dane.



# Zadania do samodzielnego wykonania

## Zadanie 4




Na podstawie rysunku utwórz aplikację, której zadaniem jest kontrola ilości prań oraz monitorowanie stanu odkurzacza. Użytkownik wpisuje wartość w polu tekstowym a następnie po naciśnięciu przycisku numer prania wyświetla się na etykiecie.

W przypadku odkurzacza po naciśnięciu przycisku „włącz” zamienia się na „wyłącz”. Powoduje to zmianę napisu na etykiecie – odkurzacz po włączeniu będzie naładowany i włączony.

# Podsumowanie

.NET MAUI to nowoczesne i efektywne narzędzie do tworzenia aplikacji Android

Umożliwia szybki rozwój dzięki współdzielonemu kodowi

Idealne rozwiązanie dla programistów .NET i C#