Widoki i nawigacja w MVVM

Czas: 2 × 45 min | praca indywidualna | Wykład 2.07

Cel: Zbudujesz aplikację z menu bocznym i trzema widokami UserControl przełączanymi przez ViewModel, a widoki wymienią dane komunikatami Messenger.

Po zajęciach potrafisz:

- tworzy widoki jako UserControl z własnymi ViewModelami
- przełącza widoki przez ContentControl i DataTemplate z DataType
- przesyła komunikaty między ViewModelami WeakReferenceMessenger
- zachowuje stan widoków przy przełączaniu

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0
- **Pakiet NuGet CommunityToolkit.Mvvm 8.4** — instalacja: Zarządzaj pakietami NuGet nuget.org/packages/CommunityToolkit.Mvvm

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Struktura

Utwórz projekt WPF `Biblioteka` z folderami `Views` i `ViewModels`; zainstaluj CommunityToolkit.Mvvm.

1. Widoki (Dodaj → Kontrolka użytkownika (WPF)): `ListaView`, `DodajView`, `InfoView`.
2. ViewModele: `ListaViewModel` (kolekcja książek), `DodajViewModel` (tytuł, autor, komenda Dodaj), `InfoViewModel` (wersja).
3. Rekord komunikatu `KsiazkaDodana(string Tytul, string Autor)`.

2 MainViewModel

Przełączanie widoków.

```
[ObservableProperty]
[NotifyPropertyChangedFor(nameof(NazwaWidoku))]
private ObservableObject _aktualnyWidok;

[RelayCommand] private void PokazListe() => AktualnyWidok = _lista;
```

1. Prywatne pola z trzema ViewModelami tworzonymi raz.
2. Właściwość `AktualnyWidok` ([ObservableProperty], typ ObservableObject) i komendy PokazListe, PokazDodaj, PokazInfo.
3. Właściwość `NazwaWidoku` odświeżana atrybutem `[NotifyPropertyChangedFor]`.

3 Okno

Rama aplikacji.

1. W zasobach okna trzy `DataTemplate DataType="{x:Type vm:…}"` z odpowiednimi widokami.
2. DockPanel: menu po lewej (ciemne tło, przyciski ze stylem) i `ContentControl Content="{Binding AktualnyWidok}"`.
3. Sprawdź: wpisz tytuł w widoku Dodaj, przejdź do Listy i z powrotem — tekst nie znika (ten sam ViewModel).

4 Komunikaty

Dodawanie książki z innego widoku.

1. `DodajViewModel` po kliknięciu wysyła `WeakReferenceMessenger.Default.Send(new KsiazkaDodana(...))`.
2. `ListaViewModel` rejestruje się w konstruktorze i dopisuje książkę do kolekcji.
3. Komenda Dodaj aktywna tylko przy wypełnionym tytule i autorze; po dodaniu komunikat „Dodano …”.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ schemat: okno, widoki, ViewModele i komunikat
- ☐ kod MainViewModel i DataTemplate
- ☐ zrzuty trzech widoków
- ☐ opis działania komunikatu

Pytania kontrolne

1. Jak ContentControl wybiera widok dla ViewModelu?
2. Dlaczego ViewModele widoków tworzymy raz?
3. Co by się stało, gdyby ListaViewModel trzymał referencję do DodajViewModel i odwrotnie?
4. Dlaczego messenger jest „słaby” (Weak)?

Kryteria oceny

Kryterium	Punkty
Widoki i ViewModele	5
MainViewModel i przełączanie	5
DataTemplate z DataType	3
Komunikaty Messenger	4
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).