MAUI i MVVM — lista zakupów

Czas: 2 × 45 min | praca indywidualna | Wykład 2.25

Cel: Zbudujesz listę zakupów w .NET MAUI we wzorcu MVVM z CommunityToolkit.Mvvm: ViewModel wstrzykiwany przez kontener, kompilowane wiązania x:DataType, usuwanie przesunięciem.

Po zajęciach potrafisz:

- stosuje MVVM w MAUI z BindingContext ustawionym przez DI
- używa kompilowanych wiązań x:DataType na stronie i w szablonie
- wywołuje komendę ViewModelu strony z szablonu elementu
- przenosi ViewModel między platformami bez zmian

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
- **Obciążenie VS „Programowanie aplikacji wieloplatformowych w .NET” (.NET MAUI) z emulatorem Androida** — emulator wymaga wirtualizacji sprzętowej; awaryjnie uruchamiaj jako Windows Machine learn.microsoft.com/dotnet/maui/get-started/installation
- **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.

Przebieg zadania

1 ViewModel

Projekt MAUI `Zakupy` z pakietem CommunityToolkit.Mvvm.

1. `Produkt` (ObservableObject): Nazwa, Kupiony.
2. `ZakupyViewModel`: kolekcja, `Nowy` z NotifyCanExecuteChangedFor, komendy Dodaj (bez duplikatów), Usun(Produkt), UsunKupione.
3. `DoKupienia` odświeżane po zmianie kolekcji i po zaznaczeniu produktu.

2 Kontener

MauiProgram i strona.

```
public MainPage(ZakupyViewModel vm)
{
    InitializeComponent();
    BindingContext = vm;
}
```

1. `builder.Services.AddSingleton<ZakupyViewModel>()` i `AddSingleton<MainPage>()`.
2. Konstruktor strony przyjmuje ViewModel i ustawia `BindingContext`.
3. W AppShell strona główna przez `ContentTemplate` — kontener sam dostarczy ViewModel.

3 Strona

XAML z kompilowanymi wiązaniami.

1. `x:DataType="vm:ZakupyViewModel"` na stronie i `x:DataType="m:Produkt"` w DataTemplate.
2. Entry z `ReturnCommand` (Enter dodaje), przycisk Dodaj, CollectionView z EmptyView, CheckBox i nazwą.
3. SwipeView z akcją Usuń: komenda przez `RelativeSource AncestorType`.
4. Wprowadź literówkę w nazwie właściwości i zapisz komunikat kompilatora; popraw.

4 Przenośność

Ten sam ViewModel w WPF.

1. Skopiuj `Produkt` i `ZakupyViewModel` do nowego projektu WPF i podłącz do prostego okna (ListBox z CheckBox).
2. Zapisz, czy trzeba było zmienić choć jedną linię ViewModelu.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod ViewModelu
- ☐ XAML strony
- ☐ komunikat błędu kompilowanego wiązania
- ☐ zrzuty z MAUI i WPF z tym samym ViewModelem

Pytania kontrolne

1. Co daje x:DataType?
2. Dlaczego ViewModel jest Singletonem?
3. Jak z szablonu elementu wywołać komendę strony?
4. Dlaczego ViewModel zadziałał w WPF bez zmian?

Kryteria oceny

Kryterium	Punkty
ViewModel z komendami i licznikiem	6
DI i BindingContext	3
Strona z x:DataType i SwipeView	6
Przeniesienie do WPF i pytania	5
Razem	**20**

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