Wiązanie danych — przelicznik jednostek

Czas: 2 × 45 min | praca indywidualna | Wykład 2.01

Cel: Zbudujesz przelicznik odległości i temperatury, w którym kontrolki nie mają ani jednej metody obsługi zdarzeń — wszystko łączy wiązanie danych z modelem implementującym INotifyPropertyChanged.

Po zajęciach potrafisz:

- implementuje INotifyPropertyChanged w klasie bazowej z metodą SetProperty
- wiąże TextBox, Slider i TextBlock z modelem przez DataContext
- dobiera tryb wiązania i UpdateSourceTrigger
- wyszukuje błędy wiązań w oknie XAML Binding Failures

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

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 Model

Utwórz projekt WPF `Przelicznik` i dwie klasy bez WPF.

```
public double Kilometry
{
    get => _kilometry;
    set
    {
        if (value < 0) value = 0;
        if (SetProperty(ref _kilometry, value))
        {
            OnPropertyChanged(nameof(Mile));
            OnPropertyChanged(nameof(Metry));
        }
    }
}
```

1. `ObservableBase` z metodami `OnPropertyChanged` i `SetProperty` (lekcja 2.01).
2. `PrzelicznikModel`: `Kilometry` (wartości ujemne zamień na 0), `Mile` z getterem i setterem (1 mila = 1,609344 km), `Metry` tylko do odczytu.
3. `Celsjusz`, `Fahrenheit` (do odczytu i zapisu) i `Kelwin`.
4. Przy zmianie właściwości bazowej zgłaszaj zmiany właściwości zależnych (`OnPropertyChanged(nameof(Mile))`).

2 Okno

Zbuduj okno bez metod obsługi zdarzeń.

1. W konstruktorze okna: `DataContext = new PrzelicznikModel();`.
2. Slider 0–100 i TextBox kilometrów związane z `Kilometry`; TextBox mil związany z `Mile`.
3. TextBlock z metrami: `StringFormat=To {0:N0} metrów`.
4. Część temperatury: dwa pola tekstowe i TextBlock z kelwinami.
5. ProgressBar związany z suwakiem przez `ElementName`.

3 Tryby i błędy

Eksperymenty z wiązaniem.

1. Porównaj `UpdateSourceTrigger=PropertyChanged` i domyślny `LostFocus` w polu mil — zapisz różnicę.
2. Ustaw `Mode=OneWay` w polu kilometrów i sprawdź, co się dzieje przy pisaniu.
3. Wpisz tekst `abc` w pole kilometrów — co pokazuje okno i co trafia do modelu?
4. Wprowadź literówkę w ścieżce (`Kilometr`) i znajdź komunikat w oknie **XAML Binding Failures** (Debugowanie → Okna). Popraw.
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod modelu
- ☐ kod XAML okna (bez zdarzeń)
- ☐ tabela wyników eksperymentów
- ☐ zrzut komunikatu XAML Binding Failures

Pytania kontrolne

1. Dlaczego właściwość Mile wymaga zgłoszenia zmiany przy zmianie Kilometry?

2. Czym różni się UpdateSourceTrigger=LostFocus od PropertyChanged?
3. Co się stanie z wiązaniem, gdy model nie implementuje INotifyPropertyChanged?
4. Kiedy użyjesz wiązania ElementName?

Kryteria oceny

Kryterium	Punkty
ObservableBase i model z zależnościami	6
Okno wyłącznie na wiązaniach	6
Eksperymenty z trybami i błędami	5
Pytania	3
Razem	**20**

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

Lista z ObservableCollection — ToDo

Czas: 2 × 45 min | praca indywidualna | Wykład 2.02

Cel: Zbudujesz listę zadań z dodawaniem, usuwaniem i oznaczaniem wykonanych, korzystając z ObservableCollection, szablonu elementu i licznika aktualizowanego automatycznie.

Po zajęciach potrafisz:

- wyświetla ObservableCollection w ListBox przez ItemsSource
- projektuje DataTemplate z wiązaniami do elementu
- reaguje na zmiany wewnątrz elementów kolekcji
- bezpiecznie usuwa wiele elementów z kolekcji

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

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 Model

Utwórz projekt WPF `ToDo`.

1. Klasa `Zadanie` (dziedziczy po ObservableBase): `Tytul`, `Termin`, `Wykonane` z powiadomieniem.
2. Klasa `ListaZadan`: `ObservableCollection<Zadanie> Zadania`, `NowyTytul`, `NowyTermin`, `Wybrane`.
3. Metody `Dodaj()` (odrzuca pusty tytuł, czyści pole), `UsunWybrane()`, `UsunWykonane()` (zwraca liczbę usuniętych).
4. Właściwość `DoZrobienia` — liczba niewykonanych.

2 Licznik na żywo

Licznik musi się zmieniać także po zaznaczeniu CheckBoxa w liście.

```
foreach (var z in Zadania.Where(z => z.Wykonane).ToList())
    Zadania.Remove(z);
```

1. Podłącz `Zadania.CollectionChanged` i w nim subskrybuj `PropertyChanged` nowych elementów (i odłączaj usuniętych).
2. Przy zmianie `Wykonane` zgłaszaj `OnPropertyChanged(nameof(DoZrobienia))`.
3. W `UsunWykonane` usuwaj z kopii listy (`.ToList()`) — sprawdź, co się dzieje bez kopii.

3 Okno

Interfejs.

1. Górny pasek: TextBox tytułu, DatePicker terminu, przycisk Dodaj (IsDefault).
2. ListBox z `ItemTemplate`: CheckBox, tytuł, termin w formacie dd.MM.yyyy w kolorze szarym.
3. Dolny pasek: licznik „Do zrobienia: N”, przyciski Usuń zaznaczone i Usuń wykonane.
4. Przyciski obsłuż w code-behind jedną linijką wywołującą metodę modelu (w zadaniu 6 zamienimy je na komendy).

4 Testy ręczne

Sprawdź działanie.

1. Dodaj 5 zadań, zaznacz 2 jako wykonane, usuń wykonane — licznik i lista muszą się zgadzać.
2. Spróbuj dodać pusty tytuł i tytuł z samych spacji.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klas Zadanie i ListaZadan
- ☐ XAML z DataTemplate
- ☐ opis błędu przy usuwaniu bez kopii
- ☐ zrzut okna z kilkoma zadaniami

Pytania kontrolne

1. Dlaczego List<T> nie wystarcza dla ListBox?
2. Co jest DataContextem wewnątrz DataTemplate?
3. Dlaczego licznik wymaga subskrypcji PropertyChanged elementów?
4. Jaki wyjątek zgłasza modyfikacja kolekcji w pętli foreach?

Kryteria oceny

Kryterium	Punkty
Klasy Zadanie i ListaZadan	5
Licznik reagujący na zmiany elementów	4
Okno z DataTemplate	5
Usuwanie (pojedyncze i wykonanych)	3
Pytania	3
Razem	**20**

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

Katalog w DataGrid

Czas: 2 × 45 min | praca indywidualna | Wykład 2.03

Cel: Zbudujesz katalog książek w DataGrid z polskimi nagłówkami, formatowaniem walut, edycją, filtrowaniem tekstem i podsumowaniem pod tabelą.

Po zajęciach potrafisz:

- definiuje kolumny DataGrid różnych typów
- formatuje waluty i daty po polsku w wiązaniach
- filtruje i sortuje dane przez ICollectionView
- aktualizuje podsumowania po zmianach danych

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

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 Dane

Utwórz projekt WPF `Katalog`.

1. Klasa `Ksiazka` z powiadomieniami: Tytuł, Autor, Rok, Cena (decimal), Dostępna (bool), Gatunek (enum).
2. Metoda statyczna `Przyklady()` z co najmniej 8 książkami.
3. W `App.xaml.cs` ustaw język formatowania pl-PL (lekcja 2.03) — bez tego cena pokaże się w dolarach.

2 DataGrid

Kolumny ręcznie (`AutoGenerateColumns="False"`).

1. Kolumny tekstowe Tytuł (szerokość *), Autor, Rok; kolumna Cena ze `StringFormat={}{0:C}`.
2. Kolumna `DataGridComboBoxColumn` dla gatunku i `DataGridCheckBoxColumn` dla dostępności.
3. Naprzemienne kolory wierszy, `CanUserAddRows="False"`, `SelectedItem` związany z modelem.
4. Sprawdź sortowanie kliknięciem nagłówka i edycję komórki dwuklikiem.

3 Filtrowanie

Model okna `KatalogModel`.

```
Widok = CollectionViewSource.GetDefaultView(Ksiazki);
Widok.Filter = o => o is Ksiazka k &&
    (string.IsNullOrWhiteSpace(Szukaj) ||
     k.Tytul.Contains(Szukaj, StringComparison.CurrentCultureIgnoreCase) ||
     k.Autor.Contains(Szukaj, StringComparison.CurrentCultureIgnoreCase));
```

1. `Widok = CollectionViewSource.GetDefaultView(Ksiazki)` i `Widok.Filter` po tytule i autorze (bez rozróżniania wielkości liter).
2. Właściwość `Szukaj` wywołuje `Widok.Refresh()`; DataGrid wiąże się z `Widok`.
3. Domyślne sortowanie po tytule: `SortDescriptions`.

4 Podsumowanie i usuwanie

Pod tabelą liczba książek i łączna wartość (widocznych po filtrze).

1. Zgłaszaj zmianę podsumowań po zmianie kolekcji, filtra i ceny w wierszu.
2. Przycisk Usuń zaznaczoną z pytaniem MessageBox.
3. Sprawdź: filtr „lem”, zmiana ceny w tabeli, usunięcie książki — podsumowanie zawsze aktualne.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klas i XAML
- ☐ zrzuty: cała lista, filtr „lem”, po edycji ceny
- ☐ wyjaśnienie poprawki języka pl-PL

Pytania kontrolne

1. Dlaczego bez poprawki języka cena wyświetla się w dolarach?
2. Czym różni się filtrowanie widoku od tworzenia nowej listy?
3. Jak dodać kolumnę z wyborem gatunku z listy?
4. Dlaczego zmiana ceny w wierszu nie zgłasza CollectionChanged?

Kryteria oceny

Kryterium	Punkty
Klasa danych i przykładowe dane	3
Kolumny DataGrid i formatowanie pl-PL	5
Filtr i sortowanie przez ICollectionView	5
Podsumowanie i usuwanie	4
Pytania	3
Razem	**20**

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

Konwertery i walidacja w wiązaniu

Czas: 2 × 45 min | praca indywidualna | Wykład 2.04

Cel: Zbudujesz formularz rejestracji, który waliduje dane w modelu przez INotifyDataErrorInfo, pokazuje komunikaty pod polami i używa konwerterów do koloru i widoczności.

Po zajęciach potrafisz:

- implementuje INotifyDataErrorInfo w modelu
- definiuje szablon błędu Validation.ErrorTemplate w stylu
- pisze własny IValueConverter i używa BooleanToVisibilityConverter
- blokuje zapis przy błędach danych

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

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 Model z walidacją

Utwórz projekt WPF `Rejestracja` i klasę `Formularz`.

1. Właściwości `Imie`, `Email`, `Wiek` (string — własny komunikat zamiast błędu konwersji).
2. Reguły jako metody statyczne zwracające tablicę błędów: imię min. 2 litery, e-mail wyrażeniem regularnym, wiek liczba 13–120.
3. Implementacja `HasErrors`, `GetErrors`, `ErrorsChanged` (lekcja 2.04).
4. `MoznaZapisac` — true, gdy pola wypełnione i brak błędów; `Niepelnoletni` — wiek < 18.

2 Szablon błędu

Komunikat pod polem zamiast samej czerwonej ramki.

```
<ControlTemplate x:Key="BladPodPolem">
  <StackPanel>
    <AdornedElementPlaceholder />
    <TextBlock Foreground="#B42318" FontSize="12"
               Text="{Binding [0].ErrorContent}" />
  </StackPanel>
</ControlTemplate>
```

1. W zasobach okna zdefiniuj `ControlTemplate` z `AdornedElementPlaceholder` i TextBlock `{Binding [0].ErrorContent}`.
2. Styl TextBox: dolny margines na komunikat i `Validation.ErrorTemplate`.
3. Sprawdź komunikaty dla: „O”, „ola@”, „dwanaście”, „121”.

3 Konwertery

Wygląd zależny od danych.

1. Konwerter `WiekNaKolorConverter`: wiek < 18 — pomarańczowy, ≥ 18 — zielony, brak — szary; użyj go dla koloru tekstu pola wieku.
2. Komunikat „potrzebna zgoda rodzica” widoczny tylko dla niepełnoletnich (BooleanToVisibilityConverter).
3. Przycisk Zapisz z `IsEnabled="{Binding MoznaZapisac}"`.

4 Sprawdzenie

Testy ręczne.

1. Wypełnij tabelę testów (min. 8 przypadków).
2. Zrób zrzuty: błędy pod polami, niepełnoletni, poprawny formularz.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy Formularz i konwertera
- ☐ XAML z szablonem błędu
- ☐ tabela testów
- ☐ zrzuty

Pytania kontrolne

1. Dlaczego wiek przechowujemy jako string?
2. Co zwraca GetErrors i kiedy WPF ją wywołuje?
3. Czym różni się konwerter od StringFormat?
4. Dlaczego BooleanToVisibilityConverter nie pozwala odwrócić warunku i jak to obejść?

Kryteria oceny

Kryterium	Punkty
INotifyDataErrorInfo z regułami	6
Szablon błędu w stylu	4
Konwerter koloru i widoczność	4
MoznaZapisac i testy	3
Pytania	3
Razem	**20**

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

MVVM od podstaw — kalkulator BMI

Czas: 2 × 45 min | praca indywidualna | Wykład 2.05

Cel: Zbudujesz kalkulator BMI we wzorcu MVVM z własną klasą RelayCommand: okno bez kodu w zdarzeniach, ViewModel bez typów WPF i testy jednostkowe ViewModelu.

Po zajęciach potrafisz:

- dzieli aplikację na View i ViewModel
- implementuje ICommand z CanExecute i CanExecuteChanged
- ustawia DataContext w XAML
- testuje ViewModel testami MSTest bez uruchamiania okna

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

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 Infrastruktura

Utwórz projekt WPF `Bmi` z folderami `Mvvm` i `ViewModels`.

1. `Mvvm/ObservableBase.cs` — z zadania 1.
2. `Mvvm/RelayCommand.cs` — `Execute`, `CanExecute`, `CanExecuteChanged`, `NotifyCanExecuteChanged()`.

2 ViewModel

`BmiViewModel` — wszystkie dane i operacje.

```
public string Masa
{
    get => _masa;
    set { if (SetProperty(ref _masa, value)) ObliczCommand.NotifyCanExecuteChanged(); }
}
```

1. Właściwości `Masa`, `Wzrost` (string), `Wynik`, `Kategoria` (prywatny setter).
2. Komenda `ObliczCommand` aktywna tylko dla masy 20–300 kg i wzrostu 100–250 cm; zmiana pól wywołuje `NotifyCanExecuteChanged`.
3. Metody statyczne `ObliczBmi` i `KategoriaBmi` (progi 18,5 / 25 / 30) oraz komenda `WyczyscCommand`.
4. Liczby parsuj z kulturą pl-PL (przecinek dziesiętny).

3 Widok

Okno bez obsługi zdarzeń.

1. `<Window.DataContext><vm:BmiViewModel /></Window.DataContext>`.
2. Pola z `UpdateSourceTrigger=PropertyChanged`, przyciski z `Command`, wynik i kategoria w TextBlock.
3. Sprawdź, że `MainWindow.xaml.cs` zawiera tylko konstruktor z `InitializeComponent()`.

4 Testy jednostkowe

Projekt MSTest z odwołaniem do projektu `Bmi`.

1. Test: bez danych komenda nieaktywna; dla 70 kg i 175 cm wynik „BMI 22,9” i „waga prawidłowa”.
2. Test parametryzowany CanExecute (min. 5 przypadków) i granice kategorii (18,4 / 18,5 / 25 / 30).
3. Test, że zmiana masy zgłasza CanExecuteChanged, i test Wyczyść.
4. Zrzut Eksploratora testów. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod RelayCommand i BmiViewModel
- ☐ XAML okna
- ☐ kod testów i zrzut Eksploratora testów
- ☐ uzasadnienie: dlaczego ViewModel da się testować bez okna

Pytania kontrolne

1. Za co odpowiada View, a za co ViewModel?
2. Skąd przycisk wie, że ma się wyszarzyć?
3. Dlaczego projekt testów może używać ViewModelu, a nie może używać okna?
4. Co trzeba zmienić, aby dodać przycisk „Pokaż normy”?

Kryteria oceny

Kryterium	Punkty
RelayCommand	3
BmiViewModel z CanExecute	6
Widok bez kodu zdarzeń	4
Testy (min. 10 przypadków)	5
Pytania	2
Razem	**20**

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

CommunityToolkit.Mvvm — ToDo

Czas: 2 × 45 min | praca indywidualna | Wykład 2.06

Cel: Przepiszesz listę zadań z zadania 2 na MVVM z pakietem CommunityToolkit.Mvvm: właściwości i komendy wygeneruje generator kodu, a okno nie będzie miało żadnej metody obsługi zdarzeń.

Po zajęciach potrafisz:

- instaluje i stosuje pakiet CommunityToolkit.Mvvm
- używa [ObservableProperty], [RelayCommand] i atrybutów powiadomień
- przekazuje element listy do komendy przez CommandParameter
- porównuje objętość kodu ręcznego i generowanego

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

Przygotowanie przed zajęciami

- Projekt ToDo z zadania 2 (kopia w nowym folderze lub nowa gałąź Git).

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 Pakiet i ViewModele

Zainstaluj CommunityToolkit.Mvvm.

```
[ObservableProperty]
[NotifyCanExecuteChangedFor(nameof(DodajCommand))]
private string _nowyTytul = "";

[RelayCommand(CanExecute = nameof(MoznaDodac))]
private void Dodaj()
{
    Zadania.Add(new ZadanieViewModel { Tytul = NowyTytul.Trim(), Termin = NowyTermin });
    NowyTytul = "";
}
```

1. `ZadanieViewModel` (partial, ObservableObject): pola `_tytul`, `_termin`, `_wykonane` z [ObservableProperty].
2. Właściwość `PoTerminie` (niewykonane z terminem w przeszłości) odświeżana w `OnWykonaneChanged` i `OnTerminChanged`.
3. `ListaViewModel`: kolekcja, `_nowyTytul` z `[NotifyCanExecuteChangedFor(nameof(DodajCommand))]`, `_nowyTermin`.
4. Komendy: `Dodaj` (CanExecute: niepusty tytuł), `Usun(ZadanieViewModel?)`, `UsunWykonane` (aktywna, gdy są wykonane).

2 Widok bez zdarzeń

Przerób okno.

1. DataContext w XAML; przyciski z `Command`; usuń metody z code-behind.
2. Przycisk ✕ w szablonie każdego zadania: komenda okna przez `RelativeSource AncestorType=ListBox` i `CommandParameter="{Binding}"`.
3. Klawisz Delete usuwa zaznaczone zadanie (`KeyBinding` z `CommandParameter` = zaznaczony element).
4. Termin po terminie na czerwono — `DataTrigger` na `PoTerminie` (kolor domyślny w stylu, nie na kontrolce).

3 Porównanie

Oceń zmianę.

1. Policz linie kodu C# w wersji z zadania 2 i w nowej wersji.
2. Podejrzyj wygenerowany kod: Eksplorator rozwiązań → Zależności → Analizatory → CommunityToolkit.Mvvm.SourceGenerators.
3. Zapisz, jaką właściwość i komendę wygenerowało pole `_nowyTytul` i metoda `UsunWykonane`.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod ViewModeli
- ☐ XAML okna
- ☐ porównanie liczby linii
- ☐ fragment wygenerowanego kodu z opisem

Pytania kontrolne

1. Dlaczego klasa z [ObservableProperty] musi być partial?
2. Jaką nazwę ma komenda wygenerowana z metody UsunWykonane?

3. Dlaczego ustawienie Foreground bezpośrednio na TextBlock blokuje DataTrigger ze stylu?
4. Jak przekazać do komendy element listy?

Kryteria oceny

Kryterium	Punkty
ViewModele z [ObservableProperty] i [RelayCommand]	7
CanExecute i powiadomienia	4
Widok bez zdarzeń, przycisk w szablonie, Delete	5
Porównanie i pytania	4
Razem	**20**

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

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).

Menu, karty i skróty — edytor notatek

Czas: 2 × 45 min | praca indywidualna | Wykład 2.08

Cel: Zbudujesz edytor notatek z menu, paskiem narzędzi, paskiem stanu i kartami TabControl, w którym menu, przyciski i skróty klawiszowe wywołują te same komendy.

Po zajęciach potrafisz:

- buduje Menu, ToolBar i StatusBar z wiązaniami do komend
- wyświetla kolekcję jako karty TabControl
- dodaje działające skróty klawiszowe KeyBinding
- wykorzystuje komendy wbudowane ApplicationCommands

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 ViewModele

Projekt WPF `Notatki` z CommunityToolkit.Mvvm.

1. Klasa statyczna `Tekst.LiczSlowa(string?)` (bez WPF).
2. `NotatkaViewModel`: Tytul, Tresc, Zmieniona; właściwości LiczbaZnakow, LiczbaSlow, Naglowek (tytuł z gwiazdką przy zmianach).
3. `EdytorViewModel`: kolekcja notatek, `Aktywna`, `PokazPasek`, `Komunikat`; komendy Nowa, Zamknij, Zapisz (aktywne, gdy jest karta).

2 Okno

Interfejs aplikacji biurowej.

```
<Window.InputBindings>
  <KeyBinding Key="N" Modifiers="Control" Command="{Binding NowaCommand}" />
  <KeyBinding Key="S" Modifiers="Control" Command="{Binding ZapiszCommand}" />
  <KeyBinding Key="W" Modifiers="Control" Command="{Binding ZamknijCommand}" />
</Window.InputBindings>
```

1. Menu Plik (Nowa, Zapisz, Zamknij kartę), Edycja (Wytnij, Kopiuj, Wklej — `ApplicationCommands`), Widok (Pasek stanu — IsCheckable).
2. ToolBar z ikonami z czcionki Segoe Fluent Icons (zapasowo Segoe MDL2 Assets), ToolTip i AutomationProperties.Name.
3. StatusBar: znaki, słowa, komunikat; ukrywany przez BooleanToVisibilityConverter.
4. TabControl: `ItemTemplate` (nagłówek) i `ContentTemplate` (TextBox treści).

3 Skróty i sprawdzenie

Spójność poleceń.

1. Dodaj KeyBinding dla Ctrl+N, Ctrl+S, Ctrl+W; w menu tylko `InputGestureText`.
2. Sprawdź każde polecenie trzema drogami (menu, pasek, klawiatura).
3. Zamknij wszystkie karty — przyciski Zapisz i Zamknij muszą się wyszarzyć.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod ViewModeli
- ☐ XAML okna
- ☐ tabela poleceń (menu / pasek / skrót)
- ☐ zrzut okna z kilkoma kartami

Pytania kontrolne

1. Dlaczego InputGestureText nie uruchamia skrótu?
2. Czym różni się ItemTemplate od ContentTemplate w TabControl?
3. Dlaczego Kopiuj i Wklej działają bez własnego kodu?
4. Jak sprawić, by pasek stanu dało się ukryć?

Kryteria oceny

Kryterium	Punkty
ViewModele i licznik słów	5
Menu, ToolBar, StatusBar	5
TabControl z kolekcji	4
Skróty i CanExecute	3
Pytania	3
Razem	**20**

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

Grafika i animacja — zegar analogowy

Czas: 2 × 45 min | praca indywidualna | Wykład 2.09

Cel: Narysujesz zegar analogowy na Canvas z gradientem i transformacjami obrotu, napędzany DispatcherTimer, oraz dodasz animację Storyboard.

Po zajęciach potrafisz:

- rysuje kształty i podziałkę na Canvas
- obraca elementy RotateTransform związanym z ViewModelem
- aktualizuje interfejs cyklicznie DispatcherTimer
- tworzy animację DoubleAnimation wyzwalaną zdarzeniem

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 Obliczenia

Projekt WPF `Zegar`.

1. Klasa statyczna `KatyZegara` z metodami `Sekundy`, `Minuty`, `Godziny` (DateTime → kąt w stopniach).
2. Sprawdź ręcznie i w krótkim teście: 12:30:00 → 15°, 180°, 0°; 21:45:15 → 292,5°, 271,5°, 90°.
3. `ZegarViewModel` z właściwościami kątów i czasu cyfrowego oraz metodą `Ustaw(DateTime)`.

2 Tarcza

Canvas 300 × 300.

```
<Line X1="150" Y1="150" X2="150" Y2="40" Stroke="#26243A" StrokeThickness="4">
  <Line.RenderTransform>
    <RotateTransform Angle="{Binding KatMinut}" CenterX="150" CenterY="150" />
  </Line.RenderTransform>
</Line>
```

1. Koło z `RadialGradientBrush` i obramowaniem.
2. 12 kresek podziałki (dłuższe co 3 godziny) — współrzędne oblicz: x = 150 + r·sin(α), y = 150 − r·cos(α).
3. Trzy wskazówki jako `Line` od środka w górę z `RotateTransform Angle="{Binding KatMinut}" CenterX="150" CenterY="150"`.
4. Środek zegara — małe koło.

3 Ruch i animacja

Zegar ożywa.

1. DispatcherTimer co 250 ms wywołuje `Ustaw(DateTime.Now)`.
2. Czas cyfrowy pod zegarem (czcionka Consolas).
3. Przycisk Start/stop z animacją powiększenia po najechaniu (EventTrigger MouseEnter, AutoReverse).
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ obliczenia kątów dla 3 godzin
- ☐ kod XAML tarczy
- ☐ zrzut zegara
- ☐ wyjaśnienie, dlaczego DispatcherTimer, a nie System.Timers.Timer

Pytania kontrolne

1. Jak obliczyć kąt wskazówki godzinowej?
2. Dlaczego obrót wymaga podania CenterX i CenterY?
3. Co się stanie, gdy zmienisz właściwość kontrolki w zdarzeniu timera z innego wątku?
4. Do czego służy AutoReverse?

Kryteria oceny

Kryterium	Punkty
Obliczenia kątów i ViewModel	5
Tarcza z gradientem i podziałką	5
Wskazówki z RotateTransform	4
Timer i animacja	4
Pytania	2
Razem	**20**

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

Własny wygląd kontrolek

Czas: 2 × 45 min | praca indywidualna | Wykład 2.10

Cel: Zaprojektujesz własne szablony przycisku i przełącznika ze wszystkimi stanami oraz pokolorujesz wiersze tabeli wyzwalaczami danych.

Po zajęciach potrafisz:

- tworzy ControlTemplate z TemplateBinding i ContentPresenter
- obsługuje stany IsMouseOver, IsPressed, IsKeyboardFocused, IsEnabled
- stosuje DataTrigger i MultiDataTrigger
- przechowuje style w słowniku zasobów aplikacji

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

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 Przycisk

Projekt WPF `Szablony`, plik `Styles.xaml` dołączony w App.xaml.

1. Styl `PrzyciskZaokraglony` z szablonem: Border z CornerRadius i ContentPresenter.
2. Wyzwalacze: najechanie (jaśniejszy), wciśnięcie (ciemniejszy), fokus klawiatury (obramowanie), nieaktywny (przezroczystość 0,4).
3. Sprawdź, że kolor tła da się zmienić na przycisku (`Background="#B42318"`) dzięki TemplateBinding.
4. Przejdź po przyciskach klawiszem Tab — fokus musi być widoczny.

2 Przełącznik

Styl `Przelacznik` dla CheckBox.

```
<Trigger Property="IsChecked" Value="True">
  <Setter TargetName="Tor" Property="Background" Value="#0E7490" />
  <Setter TargetName="Galka" Property="HorizontalAlignment" Value="Right" />
</Trigger>
```

1. Tor (Border 44 × 22, zaokrąglony) i gałka (Ellipse).
2. IsChecked=True: kolor toru i gałka po prawej.
3. Etykieta przez ContentPresenter; stan nieaktywny.

3 Wiersze danych

Rekord `Zadanie(Tytul, Status, Priorytet)` z enumami.

1. Styl `WierszZadania` dla DataGridRow: Opóźnione — czerwone tło, W trakcie — żółte, Zakończone — szary tekst.
2. MultiDataTrigger: opóźnione o wysokim priorytecie — pogrubienie.
3. Przełącznik „Tylko opóźnione” filtruje listę.
4. Wyeksportuj kopię domyślnego szablonu Button (projektant → Edytuj szablon → Edytuj kopię) i zapisz nazwy 3 elementów, które w nim znalazłeś.
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ Styles.xaml
- ☐ zrzuty przycisków w różnych stanach i przełącznika
- ☐ zrzut tabeli z kolorowaniem
- ☐ elementy znalezione w domyślnym szablonie

Pytania kontrolne

1. Czym różni się styl od szablonu?
2. Do czego służy TemplateBinding?
3. Dlaczego trzeba samodzielnie obsłużyć fokus klawiatury w szablonie?
4. Kiedy użyjesz MultiDataTrigger?

Kryteria oceny

Kryterium	Punkty
Szablon przycisku ze stanami	6
Przełącznik	4
DataTrigger i MultiDataTrigger	5
Analiza domyślnego szablonu i pytania	5
Razem	**20**

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

Operacje asynchroniczne

Czas: 2 × 45 min | praca indywidualna | Wykład 2.11

Cel: Zbudujesz aplikację liczącą liczby pierwsze w dużym zakresie bez zawieszania interfejsu: z paskiem postępu, anulowaniem i pomiarem czasu.

Po zajęciach potrafisz:

- przenosi obliczenia do wątku roboczego Task.Run i czeka na wynik await
- raportuje postęp IProgress<T>
- anuluje operację CancellationToken
- unika typowych błędów: .Result, async void, dostęp do kontrolek z innego wątku

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 Obliczenia

Projekt WPF `Pierwsze`; klasa statyczna `Obliczenia` bez WPF.

1. `CzyPierwsza(int n)` z dzieleniem tylko przez nieparzyste dzielniki do √n.
2. `LiczPierwsze(od, do, IProgress<int>?, CancellationToken)` — raportuje zmianę procentu, co 1024 liczby sprawdza token.
3. Sprawdź: do 100 → 25, do 1 000 000 → 78 498.

2 Wersja blokująca

Najpierw zobacz problem.

1. Napisz komendę synchroniczną liczącą do 20 000 000 bez Task.Run.
2. Uruchom i spróbuj przesunąć okno — zapisz, co się dzieje i ile trwa obliczenie.

3 Wersja asynchroniczna

`PierwszeViewModel` z CommunityToolkit.Mvvm.

```
var postep = new Progress<int>(p => Postep = p);
int od = Od, doL = Do;
int ile = await Task.Run(() =>
    Obliczenia.LiczPierwsze(od, doL, postep, _cts.Token));
```

1. Komenda `LiczAsync`: `await Task.Run(() => Obliczenia.LiczPierwsze(...))` — wartości Od i Do skopiuj do zmiennych lokalnych.
2. `new Progress<int>(p => Postep = p)` → ProgressBar.
3. Komenda `Anuluj` (aktywna tylko w trakcie) i obsługa `OperationCanceledException`.
4. Czas obliczeń ze `Stopwatch` w komunikacie.

4 Pułapki

Sprawdź celowo błędy (i cofnij je).

1. Zamień `await` na `.Result` — co się dzieje z oknem?
2. W Task.Run spróbuj ustawić właściwość ProgressBar bezpośrednio — zapisz wyjątek.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod Obliczenia i ViewModelu
- ☐ porównanie wersji blokującej i asynchronicznej (czas, zachowanie okna)
- ☐ opis skutków pułapek
- ☐ zrzut w trakcie liczenia

Pytania kontrolne

1. Dlaczego obliczenie w wątku interfejsu zawiesza okno?
2. Po co kopiować Od i Do do zmiennych lokalnych przed Task.Run?
3. W którym wątku wykonuje się lambda przekazana do Progress<int>?
4. Dlaczego .Result może zakleszczyć aplikację?

Kryteria oceny

Kryterium	Punkty
Klasa Obliczenia z postępem i anulowaniem	5
Wersja blokująca — obserwacje	2
ViewModel asynchroniczny	6
Pułapki	4
Pytania	3
Razem	**20**

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

SQLite i ADO.NET — książka adresowa

Czas: 2 × 45 min | praca indywidualna | Wykład 2.12

Cel: Zbudujesz książkę adresową z bazą SQLite: tabela tworzona z kodu, dodawanie, edycja, usuwanie i wyszukiwanie wyłącznie przez polecenia z parametrami.

Po zajęciach potrafisz:

- łączy się z bazą SQLite przez Microsoft.Data.Sqlite
- wykonuje polecenia CRUD z parametrami
- obsługuje wartości NULL i odczyt DataReader
- wyjaśnia i sprawdza ochronę przed wstrzykiwaniem SQL

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
- **Pakiet NuGet Microsoft.Data.Sqlite** nuget.org/packages/Microsoft.Data.Sqlite
- **DB Browser for SQLite** — podgląd pliku bazy; wersja przenośna nie wymaga instalacji sqlitebrowser.org/dl

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.
- W bazie testowej używaj fikcyjnych danych — nie wpisuj prawdziwych numerów telefonów kolegów.

Przebieg zadania

1 Repozytorium

Projekt WPF `KsiazkaAdresowa`, klasa `Kontakt` i `KontaktyRepo`.

```
cmd.CommandText = "DELETE FROM Kontakty WHERE Id = $id";
cmd.Parameters.AddWithValue("$id", id);
cmd.ExecuteNonQuery();
```

1. `UtworzTabele()` — `CREATE TABLE IF NOT EXISTS` (Id, Imie, Nazwisko NOT NULL, Telefon, Email).
2. `Szukaj(fraza)` — LIKE z parametrem `$f`; znaki % dodaj do wartości parametru.
3. `Dodaj` (z `last_insert_rowid()`), `Zmien`, `Usun(id)` — wspólna metoda dodająca parametry; puste pola jako `DBNull.Value`.

2 ViewModel i okno

Interfejs.

1. DataGrid kontaktów, pole wyszukiwania (wyniki na żywo), formularz po prawej.
2. Zaznaczenie wiersza wypełnia formularz; Nowy czyści; Zapisz dodaje lub zmienia (po Id); Usuń.
3. Plik bazy obok programu: `Path.Combine(AppContext.BaseDirectory, "kontakty.db")`.
4. Błędy bazy (`SqliteException`) pokazuj w komunikacie, nie wyjątkiem.

3 Bezpieczeństwo

Sprawdź ochronę.

1. Dodaj kontakt z nazwiskiem `O'Brien` i z imieniem `x' OR '1'='1` — oba zapisują się poprawnie.
2. Wyszukaj frazę `x' OR '1'='1` — wynik tylko jeden kontakt.
3. Dla porównania napisz (w komentarzu, NIE uruchamiaj na danych) wersję ze sklejaniem tekstu i opisz, co by się stało.
4. Otwórz plik bazy w DB Browser for SQLite i sprawdź zawartość tabeli.
5. Zatwierdź pracę w Git (plik .db dodaj do .gitignore).

W sprawozdaniu umieść

- ☐ kod KontaktyRepo
- ☐ zrzut aplikacji i bazy w DB Browser
- ☐ wyniki testów bezpieczeństwa
- ☐ wyjaśnienie wstrzykiwania SQL

Pytania kontrolne

1. Kiedy używasz ExecuteNonQuery, ExecuteScalar i ExecuteReader?
2. Dlaczego parametry chronią przed wstrzykiwaniem SQL?
3. Jak zapisać do bazy brak wartości?
4. Dlaczego pliku bazy nie dodajemy do repozytorium?

Kryteria oceny

Kryterium	Punkty
Repozytorium z parametrami	7
ViewModel i okno CRUD	6
Testy bezpieczeństwa	4
Pytania	3
Razem	**20**

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

Entity Framework Core — CRUD

Czas: 2 × 45 min | praca indywidualna | Wykład 2.13

Cel: Zbudujesz bibliotekę książek opartą na EF Core z SQLite: model, kontekst, migracje i pełne operacje CRUD z interfejsu.

Po zajęciach potrafisz:

- konfiguruje EF Core z dostawcą SQLite
- tworzy i stosuje migracje schematu bazy
- wykonuje operacje Add, Find, Where, Update, Remove i SaveChanges
- stosuje krótko żyjący kontekst w operacjach ViewModelu

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
- **Pakiety Microsoft.EntityFrameworkCore.Sqlite i .Tools 10.x** — wersja zgodna z .NET 10 nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite
- **DB Browser for SQLite** — podgląd pliku bazy; wersja przenośna nie wymaga instalacji sqlitebrowser.org/dl

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 Model i kontekst

Projekt WPF `Biblioteka`; zainstaluj pakiety EF Core 10 dla SQLite i Tools.

1. Klasa `Ksiazka`: Id, Tytul, Autor, Rok, Dostepna (na razie bez ceny).
2. `BibliotekaContext` z `DbSet<Ksiazka>` i `UseSqlite` (plik obok programu).
3. W `OnModelCreating`: maksymalne długości i wymagalność tekstów, 3 książki startowe `HasData`.

2 Migracje

Schemat bazy z kodu.

```
protected override void OnStartup(StartupEventArgs e)
{
    using (var db = new BibliotekaContext())
        db.Database.Migrate();
    base.OnStartup(e);
}
```

1. Konsola menedżera pakietów: `Add-Migration Poczatkowa`, obejrzyj wygenerowany plik w folderze Migrations.
2. W `App.OnStartup`: `db.Database.Migrate()` — baza tworzy się przy pierwszym uruchomieniu.
3. Dodaj do klasy właściwość `Cena`, potem `Add-Migration DodanaCena` i uruchom program — kolumna dochodzi bez utraty danych.
4. Sprawdź tabelę `__EFMigrationsHistory` w DB Browser for SQLite.

3 CRUD w ViewModelu

`KsiazkiViewModel` z CommunityToolkit.Mvvm.

1. `Wczytaj` — lista posortowana po tytule; `Dodaj` z formularza (po SaveChanges obiekt ma Id z bazy).
2. Edycja w DataGrid i komenda `ZapiszZmiany` (`UpdateRange` + `SaveChanges`, komunikat z liczbą zmian).
3. `Usun` zaznaczonej książki.
4. Każda operacja: `using var db = new BibliotekaContext();`.

4 Sprawdzenie

Dane trwałe.

1. Dodaj, zmień i usuń książki; zamknij i uruchom program ponownie — dane zostają.
2. Porównaj zawartość pliku bazy w DB Browser z widokiem w aplikacji.
3. Zatwierdź pracę w Git: folder Migrations TAK, plik .db NIE.

W sprawozdaniu umieść

☐ kod modelu i kontekstu

- ☐ fragment migracji DodanaCena z opisem
- ☐ kod ViewModelu
- ☐ zrzut aplikacji i tabeli w DB Browser

Pytania kontrolne

1. Co to jest DbContext i DbSet?
2. Po co są migracje i dlaczego trafiają do repozytorium?
3. Co robi SaveChanges?
4. Dlaczego kontekst tworzymy na czas jednej operacji?

Kryteria oceny

Kryterium	Punkty
Model, kontekst, HasData	4
Migracje (dwie) i Migrate przy starcie	5
CRUD w ViewModelu	7
Pytania	4
Razem	**20**

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

EF Core — relacje: dziennik ocen

Czas: 2 × 45 min | praca indywidualna | Wykład 2.14

Cel: Zbudujesz dziennik ocen z trzema powiązanymi encjami: listą uczniów ze średnimi, szczegółami ocen wybranego ucznia i średnimi przedmiotów liczonymi w bazie.

Po zajęciach potrafisz:

- modeluje relacje 1:N z kluczami obcymi i właściwościami nawigacyjnymi
- wczytuje dane powiązane Include/ThenInclude
- pisze zapytania LINQ z grupowaniem tłumaczone na SQL
- nakłada ograniczenia w Fluent API

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
- **Pakiety Microsoft.EntityFrameworkCore.Sqlite i .Tools 10.x** — wersja zgodna z .NET 10 nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite
- **DB Browser for SQLite** — podgląd pliku bazy; wersja przenośna nie wymaga instalacji sqlitebrowser.org/dl

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 Model

Projekt WPF `Dziennik`.

1. Encje `Uczen` (Imie, Nazwisko, Klasa, Oceny), `Przedmiot` (Nazwa, Oceny), `Ocena` (Wartosc, Data, Opis, UczenId, Uczen, PrzedmiotId, Przedmiot).
2. W Uczen właściwości pomocnicze `PelneImie` i `Srednia` (null, gdy brak ocen).
3. `DziennikContext`: ograniczenie CHECK `Wartosc BETWEEN 1 AND 6`, unikalna nazwa przedmiotu, dane startowe (3 uczniów, 3 przedmioty, 7 ocen).
4. Migracja i `Migrate()` przy starcie.

2 Zapytania

`DziennikViewModel`.

```
foreach (var s in db.Oceny
    .GroupBy(o => o.Przedmiot.Nazwa)
    .Select(g => new SredniaPrzedmiotu(g.Key, g.Average(o => o.Wartosc), g.Count()))
    .OrderBy(s => s.Przedmiot))
    SredniePrzedmiotow.Add(s);
```

1. Uczniowie z ocenami i przedmiotami: `Include(u => u.Oceny).ThenInclude(o => o.Przedmiot).AsNoTracking()`.
2. Średnie przedmiotów liczone w bazie: `GroupBy(o => o.Przedmiot.Nazwa).Select(g => new SredniaPrzedmiotu(...))`.
3. Podejrzyj SQL: `Debug.WriteLine(zapytanie.ToQueryString())` i zapisz, czy jest w nim GROUP BY.
4. Sprawdź, co pokazuje lista ocen, gdy usuniesz Include.

3 Interfejs

Okno z listą i szczegółami.

1. DataGrid uczniów: uczeń, klasa, liczba ocen, średnia (dla braku ocen „—”: `TargetNullValue`).
2. Panel szczegółów: oceny wybranego ucznia (`Run` z przedmiotem, wartością i datą).
3. Dodawanie oceny: przedmiot (ComboBox z DisplayMemberPath), wartość 1–6, przycisk aktywny po wyborze ucznia i przedmiotu.
4. Lista średnich przedmiotów odświeżana po dodaniu oceny.

4 Sprawdzenie

Integralność danych.

1. Spróbuj w DB Browser wpisać ręcznie ocenę 7 — co się dzieje?
2. Porównaj średnie z aplikacji z obliczeniem ręcznym.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod modelu i kontekstu
- ☐ wygenerowany SQL zapytania średnich
- ☐ zrzuty okna
- ☐ wynik próby wpisania oceny 7

Pytania kontrolne

1. Jak EF rozpoznaje klucz obcy UczenId?
2. Dlaczego bez Include lista ocen jest pusta?
3. Gdzie liczona jest średnia przedmiotu — w bazie czy w pamięci? Skąd wiesz?
4. Do czego służy ograniczenie CHECK?

Kryteria oceny

Kryterium	Punkty
Model i relacje 1:N	5
Fluent API i dane startowe	3
Zapytania Include i GroupBy	5
Interfejs z dodawaniem ocen	4
Pytania	3
Razem	**20**

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

Wstrzykiwanie zależności i repozytorium

Czas: 2 × 45 min | praca indywidualna | Wykład 2.15

Cel: Zrefaktoryzujesz dziennik z zadania 14: dostęp do danych za interfejsem repozytorium, okna dialogowe za interfejsem usługi, obiekty tworzone przez kontener DI.

Po zajęciach potrafisz:

- wydziela interfejsy usług i ich implementacje
- konfiguruje kontener Microsoft.Extensions.DependencyInjection w aplikacji WPF
- dobiera czas życia usług i używa fabryki kontekstu EF
- zastępuje implementację atrapą bez zmian w ViewModelu

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
- **Pakiety Microsoft.EntityFrameworkCore.Sqlite i .Tools 10.x** — wersja zgodna z .NET 10 nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite
- **Pakiet Microsoft.Extensions.DependencyInjection 10.x** nuget.org/packages/Microsoft.Extensions.DependencyInjection

Przygotowanie przed zajęciami

- Projekt dziennika z zadania 14 (nowa gałąź Git `funkcja/di`).

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 Interfejsy

Folder `Services`.

1. `IOcenyRepo`: PobierzUczniow, PobierzPrzedmioty, DodajOcene.
2. `IDialogService`: Info, Potwierdz.
3. `EfOcenyRepo` — przyjmuje `IDbContextFactory<DziennikContext>` i tworzy kontekst na operację.
4. `DialogService` (MessageBox) i `PamiecOcenyRepo` (dane w listach).

2 ViewModel bez new

`OcenyViewModel(IOcenyRepo repo, IDialogService dialog)`.

1. Przenieś logikę z DziennikViewModel; usuń wszystkie `new DziennikContext()` i `MessageBox` z ViewModelu.
2. Przy ocenie 1 pytaj o potwierdzenie przez `dialog.Potwierdz`.
3. W kontekście dodaj konstruktor z `DbContextOptions` i warunek `if (!o.IsConfigured)` w OnConfiguring.

3 Kontener

App.xaml.cs.

```
var sc = new ServiceCollection();
sc.AddDbContextFactory<DziennikContext>(o => o.UseSqlite(baza));
sc.AddSingleton<IOcenyRepo, EfOcenyRepo>();
sc.AddSingleton<IDialogService, DialogService>();
sc.AddTransient<OcenyViewModel>();
sc.AddTransient<MainWindow>();
Uslugi = sc.BuildServiceProvider();
Uslugi.GetRequiredService<MainWindow>().Show();
```

1. Usuń `StartupUri` z App.xaml.
2. Zarejestruj: `AddDbContextFactory`, repozytorium i dialog jako Singleton, ViewModel i okno jako Transient.
3. Okno przyjmuje ViewModel w konstruktorze i ustawia DataContext.
4. Zamień jedną linią repozytorium na `PamiecOcenyRepo` i uruchom aplikację bez bazy. Wróć do EF.

4 Podsumowanie

Oceń refaktoryzację.

1. Narysuj schemat zależności (kto zna kogo) przed i po zmianach.
2. Sprawdź wyszukiwaniem (Ctrl+Shift+F), że w ViewModelach nie ma `new DziennikContext` ani `MessageBox`.
3. Pull request z gałęzi do main z opisem zmian.

W sprawozdaniu umieść

- ☐ kod interfejsów i App.xaml.cs
- ☐ schematy zależności przed i po
- ☐ zrzut aplikacji z PamiecOcenyRepo
- ☐ opis pull requesta

Pytania kontrolne

1. Co zyskujemy, gdy ViewModel zależy od interfejsu, a nie od klasy EF?
2. Dlaczego DbContext nie powinien być Singletonem?
3. Czym różni się AddDbContext od AddDbContextFactory?
4. Co to jest composition root?

Kryteria oceny

Kryterium	Punkty
Interfejsy i implementacje	5
ViewModel bez new i MessageBox	5
Konfiguracja kontenera	5
Zamiana na atrapę i schematy	3
Pytania	2
Razem	**20**

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

Klient REST API — kursy walut NBP

Czas: 2 × 45 min | praca indywidualna | Wykład 2.16

Cel: Zbudujesz aplikację pobierającą aktualną tabelę kursów walut z API Narodowego Banku Polskiego, z przelicznikiem, historią notowań i odporną na brak sieci.

Po zajęciach potrafisz:

- pobiera dane JSON asynchronicznie przez HttpClient
- mapuje odpowiedź na rekordy DTO
- obsługuje błędy sieci, limitu czasu i formatu
- oddziela logikę przeliczeń od komunikacji sieciowej

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
- **API NBP (bez rejestracji)** — kursy średnie: tabela A api.nbp.pl

Przygotowanie przed zajęciami

- Przykładowe odpowiedzi API (`pliki/tabela_A_przyklad.json`, `pliki/EUR_last5_przyklad.json`) — do pracy bez sieci i testów.

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 Rozpoznanie API

Otwórz w przeglądarce adres tabeli A.

1. `https://api.nbp.pl/api/exchangerates/tables/A/?format=json` — zapisz nazwy pól i ich typy.
2. `https://api.nbp.pl/api/exchangerates/rates/A/EUR/last/10/?format=json` — historia jednej waluty.
3. Rekordy DTO: `Kurs`, `TabelaKursow`, `KursDnia`, `KursyWaluty` (data jako DateOnly, kurs jako decimal).

2 Usługa i logika

Projekt WPF `Kursy`.

```
public async Task<TabelaKursow> PobierzTabeleAAsync(CancellationToken token = default)
{
    var tabele = await Http.GetFromJsonAsync<List<TabelaKursow>>(
        "exchangerates/tables/A/?format=json", token);
    return tabele is { Count: > 0 } ? tabele[0]
        : throw new InvalidOperationException("Pusta odpowiedź serwera.");
}
```

1. Interfejs `INbpService` i `NbpService` z jednym statycznym HttpClient (BaseAddress, Timeout 10 s).
2. Klasa statyczna `Przelicznik`: NaZlote, ZeZlotych, ZmianaProcent (zaokrąglenie do 2 miejsc).
3. Sprawdź przelicznik na danych z pliku przykładowego (100 EUR po kursie 4,4032 = 440,32 zł).

3 ViewModel i okno

Interfejs.

1. Komenda `WczytajAsync` z obsługą `HttpRequestException`, `TaskCanceledException`, `JsonException` — zrozumiałe komunikaty.
2. DataGrid kursów; przy wyborze waluty pobierz 10 ostatnich notowań i pokaż zmianę procentową.
3. Przelicznik: kwota → wynik w złotych i w walucie (właściwości odświeżane NotifyPropertyChangedFor).
4. Wskaźnik ładowania: ProgressBar widoczny podczas `WczytajCommand.IsRunning`.

4 Odporność

Sprawdź złe warunki.

1. Wyłącz sieć (tryb samolotowy lub odłączony kabel) — komunikat, brak awarii.
2. Zapytaj o nieistniejący kod waluty (np. XYZ) — co zwraca serwer (kod HTTP 404) i jak reaguje aplikacja?
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ opis struktury JSON
- ☐ kod DTO, usługi i przelicznika
- ☐ zrzut aplikacji z kursami i historią
- ☐ wyniki testów odporności

Pytania kontrolne

1. Co to jest DTO?
2. Dlaczego HttpClient tworzymy raz?
3. Jakie wyjątki może zgłosić GetFromJsonAsync i co oznaczają?

4. Dlaczego przelicznik da się przetestować bez internetu?

Kryteria oceny

Kryterium	Punkty
DTO i usługa z HttpClient	5
Przelicznik	3
ViewModel i okno z historią	6
Obsługa błędów i testy odporności	4
Pytania	2
Razem	**20**

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

Ustawienia i dziennik zdarzeń

Czas: 2 × 45 min | praca indywidualna | Wykład 2.17

Cel: Dodasz do swojej aplikacji zapamiętywanie ustawień użytkownika w pliku JSON w AppData, listę ostatnich plików, dziennik zdarzeń i przechwytywanie nieobsłużonych wyjątków.

Po zajęciach potrafisz:

- zapisuje i odczytuje ustawienia w folderze danych aplikacji
- odpornie obsługuje brak i uszkodzenie pliku ustawień
- prowadzi dziennik zdarzeń z poziomami
- przechwytuje nieobsłużone wyjątki aplikacji WPF

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

Przygotowanie przed zajęciami

- Notatnik z roku 1 (zadanie 30) lub edytor notatek z zadania 8.

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.
- Nie zapisuj w dzienniku haseł, numerów PESEL ani treści notatek użytkownika.

Przebieg zadania

1 Ścieżki i ustawienia

Klasy bez WPF.

1. `Sciezki`: folder w `ApplicationData` (ustawienia) i w `LocalApplicationData` (dzienniki), tworzone przy pierwszym użyciu.
2. `Ustawienia`: szerokość, wysokość, zmaksymalizowane, ciemny motyw, lista ostatnich plików; metoda `DodajOstatni` (bez duplikatów, max 5).
3. `UstawieniaService`: Wczytaj (brak pliku / uszkodzony JSON → domyślne), Zapisz.

2 Dziennik

Klasa statyczna `Dziennik`.

```
public static void Blad(string t, Exception ex) =>
    Zapisz("BŁĄD", $"{t}: {ex.GetType().Name}: {ex.Message}{Environment.NewLine}
{ex.StackTrace}");
```

1. Metody Info, Ostrzezenie, Blad (z typem wyjątku i stosem wywołań).
2. Linia: data z milisekundami, poziom, tekst; zapis do pliku z `lock` i do okna Output.
3. Błąd zapisu dziennika nie może zatrzymać programu.

3 Połączenie z aplikacją

App.xaml.cs (instrukcja w pliku INSTRUKCJA.txt rozwiązania).

1. Usuń StartupUri; w OnStartup wczytaj ustawienia i utwórz okno z zapamiętanym rozmiarem i stanem.
2. Przy zamykaniu zapisz rozmiar (`RestoreBounds` dla zmaksymalizowanego okna) i stan.
3. Menu Plik → Ostatnie z listą plików; brakujący plik — komunikat i usunięcie z listy.
4. `DispatcherUnhandledException`, `TaskScheduler.UnobservedTaskException`, `AppDomain.UnhandledException` — wpis do dziennika i komunikat.

4 Sprawdzenie

Scenariusze.

1. Zmień rozmiar okna, zamknij, uruchom — rozmiar zapamiętany.
2. Zepsuj ręcznie plik ustawień — program startuje z domyślnymi i zapisuje ostrzeżenie.
3. Wywołaj wyjątek przyciskiem testowym (tylko Debug) — wpis w dzienniku, program działa dalej.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klas Sciezki, Ustawienia, Dziennik
- ☐ zawartość pliku ustawień
- ☐ fragment dziennika
- ☐ wyniki scenariuszy

Pytania kontrolne

1. Dlaczego ustawień nie zapisujemy w folderze programu?
2. Czym różnią się foldery ApplicationData i LocalApplicationData?
3. Po co lock w metodzie zapisu dziennika?
4. Czy DispatcherUnhandledException zastępuje try/catch? Uzasadnij.

Kryteria oceny

Kryterium	Punkty
Ustawienia z odpornym odczytem	5
Lista ostatnich plików	3
Dziennik	4
Połączenie z aplikacją i wyjątki	5
Pytania	3
Razem	**20**

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

Testy ViewModeli

Czas: 2 × 45 min | praca indywidualna | Wykład 2.18

Cel: Napiszesz testy jednostkowe ViewModeli z zadań 1, 5 i 15 — z powiadomieniami, komendami i atrapami usług — i ocenisz pokrycie kodu.

Po zajęciach potrafisz:

- testuje właściwości, CanExecute i Execute ViewModelu
- sprawdza zgłaszanie PropertyChanged
- zastępuje usługi atrapami przez interfejsy
- analizuje pokrycie kodu i uzupełnia brakujące testy

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

Przygotowanie przed zajęciami

- Projekty z zadań 1, 5 i 15.

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 Projekt testów

Rozwiązanie z projektami aplikacji.

1. Dodaj projekt MSTest (TargetFramework `net10.0-windows`, jeśli odwołuje się do projektu WPF).
2. Odwołania do projektów z zadań 1, 5 i 15 (lub kopie klas ViewModeli do biblioteki klas).

2 Model i BMI

Testy klas bez usług.

```
var zmienione = new List<string?>();
m.PropertyChanged += (s, e) => zmienione.Add(e.PropertyName);
m.Kilometry = 42.195;
CollectionAssert.Contains(zmienione, nameof(PrzelicznikModel.Mile));
```

1. PrzelicznikModel: zmiana km zgłasza Mile i Metry; ta sama wartość — brak powiadomienia; ujemne → 0; °C → °F (DataRow).
2. BmiViewModel: CanExecute dla min. 5 przypadków, wynik dla 70 kg i 175 cm, granice kategorii, Wyczyść.

3 Atrapy

OcenyViewModel z zadania 15.

1. Klasa `AtrapaDialogu(bool odpowiedz)` zapamiętuje komunikaty i pytania.
2. Repozytorium: `PamiecOcenyRepo` z zadania 15.
3. Testy: wczytanie danych w konstruktorze, komenda nieaktywna bez wyboru, zapis oceny i komunikat, ocena 1 z potwierdzeniem i bez (DataRow).

4 Pokrycie

Test → Analizuj pokrycie kodu dla wszystkich testów.

1. Zapisz procent pokrycia klas ViewModeli.
2. Znajdź niepokrytą gałąź i dopisz test.
3. Zrzut wyników. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klas testowych i atrap
- ☐ zrzut Eksploratora testów
- ☐ zrzut pokrycia przed i po dopisaniu testu
- ☐ wniosek: czego testy nie sprawdzają

Pytania kontrolne

1. Dlaczego atrapa jest lepsza od prawdziwej bazy w teście jednostkowym?
2. Jak sprawdzić, że właściwość zgłosiła PropertyChanged?
3. Co oznacza 100% pokrycia i czego nie gwarantuje?
4. Jak nazywasz testy i dlaczego?

Kryteria oceny

Kryterium	Punkty
Testy modelu przelicznika	4
Testy BmiViewModel	4
Atrapy i testy OcenyViewModel	7
Pokrycie i dopisany test	3
Pytania	2
Razem	**20**

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

Lokalizacja i dostępność

Czas: 2 × 45 min | praca indywidualna | Wykład 2.19

Cel: Przygotujesz polską i angielską wersję aplikacji ToDo z zadania 6 w plikach zasobów oraz poprawisz jej dostępność: nazwy dla czytnika ekranu, kolejność klawisza Tab, kontrast i skalowanie.

Po zajęciach potrafisz:

- przenosi teksty interfejsu do plików .resx i tworzy wersję językową
- wybiera język i format przez CultureInfo
- opisuje kontrolki właściwościami AutomationProperties
- sprawdza aplikację Narratorem, klawiaturą i przy powiększeniu tekstu

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
- **Accessibility Insights for Windows** — bezpłatne narzędzie Microsoft accessibilityinsights.io/downloads

Przygotowanie przed zajęciami

- Aplikacja ToDo z zadania 6.

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 Zasoby tekstowe

Folder `Resources`.

1. `Teksty.resx` (Modyfikator dostępu: Public) — wszystkie napisy okna: tytuł, przyciski, licznik „Do zrobienia: {0}”, podpowiedzi.
2. `Teksty.en.resx` — te same nazwy, teksty angielskie.
3. W XAML: `xmlns:r="clr-namespace:ToDo.Resources"` i `{x:Static r:Teksty.Dodaj}`; licznik przez `StringFormat={x:Static r:Teksty.Licznik}`.
4. Sprawdź, że w XAML i ViewModelu nie został żaden napis po polsku.

2 Wybór języka

App.xaml.cs.

```
var kultura = new CultureInfo(jezyk);            // "pl-PL" lub "en-US"
CultureInfo.DefaultThreadCurrentUICulture = kultura;   // wybór pliku .resx
CultureInfo.DefaultThreadCurrentCulture = kultura;     // liczby i daty
```

1. Ustaw `DefaultThreadCurrentUICulture` i `DefaultThreadCurrentCulture` przed utworzeniem okna.
2. Ustaw język formatowania wiązań (`FrameworkElement.LanguageProperty`) — daty w liście w formacie wybranego języka.
3. Uruchom w obu językach i porównaj format daty.

3 Dostępność

Opisz i uporządkuj kontrolki.

1. Przycisk ✕ w szablonie: `AutomationProperties.Name` = „Usuń zadanie”; CheckBox: nazwa = tytuł zadania.
2. Pole nowego zadania: `AutomationProperties.HelpText`; licznik: `AutomationProperties.LiveSetting="Polite"`.
3. Logiczna kolejność `TabIndex`; skróty Alt+litera w etykietach.
4. Zamień szary #808080 na ciemniejszy (kontrast ≥ 4,5:1 na białym tle) — policz kontrast w narzędziu lub kalkulatorze WCAG.

4 Sprawdzenie

Test bez myszy.

1. Odłącz mysz: dodaj, oznacz i usuń zadanie samą klawiaturą.
2. Narrator (Win+Ctrl+Enter): co czyta przy przyciskach ✕?
3. Accessibility Insights → FastPass: zapisz liczbę problemów przed i po poprawkach.
4. Rozmiar tekstu Windows 150% — czy nic się nie ucina?
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ tabela tekstów PL/EN
- ☐ zrzuty okna w obu językach
- ☐ wyniki FastPass przed i po
- ☐ opis testu klawiaturą i Narratorem

Pytania kontrolne

1. Jak .NET wybiera plik .resx?
2. Czym różni się CurrentCulture od CurrentUICulture?

3. Do czego służy AutomationProperties.Name, skoro przycisk ma ToolTip?
4. Dlaczego informacji nie przekazujemy wyłącznie kolorem?

Kryteria oceny

Kryterium	Punkty
Pliki .resx i XAML bez napisów na sztywno	6
Wybór języka i formatów	3
AutomationProperties, TabIndex, kontrast	5
Testy dostępności	4
Pytania	2
Razem	**20**

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

Publikowanie aplikacji

Czas: 2 × 45 min | praca indywidualna | Wykład 2.20

Cel: Przygotujesz wydanie aplikacji z zadania 14/15: metadane, ikonę, dwie wersje publikacji (zależną od .NET i samodzielną), instrukcję dla użytkownika — i sprawdzisz je na innym komputerze.

Po zajęciach potrafisz:

- ustawia metadane wydania i ikonę w pliku projektu
- publikuje aplikację w konfiguracji Release w trybie framework-dependent i self-contained
- przygotowuje paczkę i instrukcję dla użytkownika
- testuje wydanie poza środowiskiem programisty

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu
- drugi komputer (np. sąsiednie stanowisko lub laptop) do testu

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

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 Metadane i ikona

Plik projektu (prawy przycisk → Edytuj plik projektu).

1. Version 1.0.0, Product, Company, Copyright, Description, AssemblyName.
2. Ikona `Assets\dziennik.ico` (kilka rozmiarów, np. wygenerowana z PNG w darmowym narzędziu) — `ApplicationIcon` i ikona okna.
3. Okno „O programie": produkt, wersja i środowisko odczytane z metadanych (klasa `InformacjeOProgramie`).

2 Publikacja

Dwa warianty.

```
dotnet publish -c Release -r win-x64 --self-contained true `
    -p:PublishSingleFile=true -p:IncludeNativeLibrariesForSelfExtract=true `
    -o publish\samodzielna
```

1. Zależna od .NET, jeden plik: `dotnet publish -c Release -r win-x64 --self-contained false -p:PublishSingleFile=true`.
2. Samodzielna, jeden plik: `--self-contained true` i `-p:IncludeNativeLibrariesForSelfExtract=true` (SQLite ma bibliotekę natywną).
3. Zapisz rozmiary obu wersji. Ten sam proces z interfejsu: prawy przycisk → Publikuj → Folder.
4. Skrypt `publikuj.ps1` z rozwiązania tworzy obie wersje i archiwa ZIP.

3 Paczka

Dla użytkownika.

1. README.md: wymagania, uruchomienie, ostrzeżenie SmartScreen, autor (szablon w rozwiązaniu).
2. Archiwa ZIP z wersją i rodzajem w nazwie.
3. Sprawdź właściwości pliku .exe w Eksploratorze (Szczegóły) — czy widać wersję i produkt?

4 Test na innym komputerze

Poza Visual Studio.

1. Uruchom obie wersje na innym komputerze (wersja zależna — bez zainstalowanego .NET 10 Desktop Runtime pojawi się komunikat; zapisz go).
2. Sprawdź: utworzenie bazy, dodanie oceny, zapis ustawień.
3. Zatwierdź pracę w Git (folder publish w .gitignore).

W sprawozdaniu umieść

- ☐ fragment pliku projektu z metadanymi
- ☐ tabela rozmiarów i wyników testu
- ☐ README
- ☐ zrzut właściwości pliku .exe i okna O programie

Pytania kontrolne

1. Kiedy wybierzesz publikację self-contained?
2. Dlaczego potrzebny jest parametr IncludeNativeLibrariesForSelfExtract?
3. Skąd okno „O programie” zna numer wersji?
4. Co oznacza ostrzeżenie SmartScreen i jak je usunąć w profesjonalnym oprogramowaniu?

Kryteria oceny

Kryterium	Punkty
Metadane, ikona, O programie	4
Dwie publikacje	5
README i paczki	3
Test na innym komputerze	5
Pytania	3
Razem	**20**

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

Praca zespołowa z Git

Czas: 2 × 45 min | praca w parach (osoba A i osoba B — role w scenariuszu) | Wykład 2.21

Cel: W parach poprowadzicie wspólne repozytorium na GitHub: gałęzie funkcji, pull requesty z przeglądem, kontrolowany konflikt scalania i ochronę gałęzi main.

Po zajęciach potrafisz:

- tworzy gałęzie funkcji i wypycha je do zdalnego repozytorium
- otwiera pull request, recenzuje i scala zmiany
- rozwiązuje konflikt scalania w Visual Studio
- konfiguruje ochronę gałęzi i szablon pull requesta

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
- **Git for Windows** — lub Git wbudowany w Visual Studio git-scm.com/downloads/win
- **Konto GitHub (bezpłatne)** — lub szkolne konto GitHub Education github.com/signup

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.
- Do logowania na GitHub używaj uwierzytelniania dwuskładnikowego; nie udostępniaj tokenów i haseł koledze.

Przebieg zadania

1 Repozytorium

Osoba A.

1. Wypycha aplikację ToDo (zadanie 6) na GitHub jako repozytorium prywatne; dodaje osobę B jako współpracownika.
2. Dodaje plik `.github/pull_request_template.md` (szablon w rozwiązaniu) i 2 zgłoszenia (Issues) z zadaniami.
3. Osoba B klonuje repozytorium w Visual Studio (Klonuj repozytorium).

2 Gałęzie i pull requesty

Praca równoległa (szczegóły: SCENARIUSZ_KONFLIKTU.md).

```
git switch -c funkcja/sortowanie
git add .
git commit -m "Sortowanie zadań po terminie (#2)"
git push -u origin funkcja/sortowanie
```

1. Każda osoba: gałąź `funkcja/…`, zmiana w innym miejscu kodu, commit z dobrym komunikatem, push.
2. Pull request z opisem i `Closes #numer`; druga osoba robi przegląd z co najmniej jednym komentarzem do linii kodu.
3. Po poprawce — merge i usunięcie gałęzi.

3 Konflikt

Celowo zmieniacie ten sam wiersz XAML.

1. Obie osoby zmieniają wiersz przycisku Dodaj na różne sposoby, każda na swojej gałęzi.
2. Osoba A scala pierwsza; osoba B scala main do swojej gałęzi i dostaje konflikt.
3. Rozwiązanie w Edytorze scalania VS, build, testy, commit, push, pull request.
4. Zrzut `git log --oneline --graph --all`.

4 Ochrona main

Ustawienia repozytorium.

1. Reguła: wymagany pull request i 1 akceptacja przed scaleniem do main.
2. Sprawdźcie, że bezpośredni push na main zostaje odrzucony — zapiszcie komunikat.

W sprawozdaniu umieść

- ☐ adres repozytorium i pull requestów
- ☐ zrzut komentarza z przeglądu
- ☐ zrzut edytora scalania i opis rozwiązania konfliktu
- ☐ graf historii
- ☐ komunikat odrzuconego pusha

Pytania kontrolne

1. Po co pracować na gałęziach?
2. Co sprawdzasz podczas przeglądu pull requesta?
3. Dlaczego konflikt powstał w kroku 3, a nie w kroku 2?
4. Co daje ochrona gałęzi main?

Kryteria oceny

Kryterium	Punkty
Repozytorium, szablon PR, zgłoszenia	3
Gałęzie i pull requesty z przeglądem	6
Rozwiązany konflikt	5
Ochrona main	3
Pytania	3
Razem	**20**

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

Pierwsza aplikacja .NET MAUI

Czas: 2 × 45 min | praca indywidualna | Wykład 2.22

Cel: Utworzysz pierwszą aplikację .NET MAUI, uruchomisz ją na Windows i w emulatorze Androida oraz zmienisz stronę główną na licznik kliknięć.

Po zajęciach potrafisz:

- przygotowuje środowisko MAUI i emulator Androida
- rozpoznaje strukturę projektu MAUI
- buduje stronę z układem, etykietą i przyciskami
- obsługuje zdarzenie Clicked i właściwości wyglądu

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu
- komputer z włączoną wirtualizacją sprzętową (dla emulatora) lub telefon z Androidem i kablem USB (opcjonalnie)

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

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.
- Telefon podłączaj tylko za zgodą nauczyciela; w trybie programisty nie zmieniaj innych ustawień telefonu.

Przebieg zadania

1 Środowisko

Przygotowanie (część może być już zrobiona w pracowni).

1. Instalator VS: obciążenie .NET MAUI — sprawdź, czy jest zainstalowane.
2. Narzędzia → Android → Menedżer urządzeń: utwórz emulator (np. Pixel z Androidem 15 lub 16, obraz x86_64).
3. Zapisz wersję Androida emulatora i API.

2 Projekt

Szablon **Aplikacja .NET MAUI**, nazwa `Licznik`, .NET 10.

1. Uruchom jako **Windows Machine**, potem w emulatorze — zmierz czas pierwszego i drugiego uruchomienia.
2. Opisz w sprawozdaniu foldery Platforms, Resources (AppIcon, Images, Fonts, Styles) i pliki MauiProgram, App, AppShell, MainPage.
3. Znajdź w App.xaml.cs metodę `CreateWindow` — co tworzy?

3 Licznik

Zmień MainPage.

```
private void OnKlik(object? sender, EventArgs e)
{
    _licznik++;
    LblLicznik.Text = _licznik == 1 ? "Kliknięto 1 raz" : $"Kliknięto {_licznik} razy";
    LblLicznik.TextColor = _licznik >= 10 ? Color.FromArgb("#0E7490") : Colors.Black;
}
```

1. Etykieta „Kliknięto N razy” (poprawna odmiana dla 1), przyciski Kliknij i Zeruj (czerwony).
2. Po 10 kliknięciach kolor tekstu zmienia się na #0E7490.
3. Etykieta z platformą i modelem urządzenia (`DeviceInfo.Platform`, `DeviceInfo.Model`).
4. Zmień tekst w XAML podczas działania aplikacji — sprawdź Hot Reload.

4 Ikona

Personalizacja.

1. Podmień obraz w Resources/AppIcon (SVG lub PNG) i kolor tła ikony w pliku projektu (`MauiIcon`, `Color`).
2. Sprawdź ikonę w emulatorze na liście aplikacji.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ wersja emulatora i czasy uruchomienia
- ☐ opis struktury projektu
- ☐ kod strony
- ☐ zrzuty: Windows i emulator, ikona aplikacji

Pytania kontrolne

1. Czym MAUI różni się od WPF?
2. Dlaczego pierwsze uruchomienie w emulatorze trwa długo?
3. Do czego służy folder Platforms?
4. Co zwraca DeviceInfo.Platform na Windows, a co w emulatorze?

Kryteria oceny

Kryterium	Punkty
Środowisko i emulator	4
Uruchomienie na dwóch platformach	4
Strona licznika	6
Ikona i opis struktury	4
Pytania	2
Razem	**20**

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

Układy i kontrolki MAUI — rezerwacja

Czas: 2 × 45 min | praca indywidualna | Wykład 2.23

Cel: Zbudujesz mobilny formularz rezerwacji sali gier z kontrolkami MAUI, walidacją w osobnej klasie i podsumowaniem z ceną.

Po zajęciach potrafisz:

- buduje formularz na Grid z RowDefinitions i ColumnDefinitions
- dobiera kontrolki i klawiatury do danych
- ogranicza zakres dat w DatePicker
- waliduje dane w klasie bez MAUI i pokazuje komunikat DisplayAlertAsync

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

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 Logika

Projekt MAUI `Rezerwacje`; rekord `Rezerwacja` bez typów MAUI.

1. Pola: Imie, Telefon, Sala, Data, Osoby, Przekaski; lista sal i cennik (VR 45 zł, konsola 30 zł, karaoke 25 zł za osobę; przekąski 12 zł za osobę).
2. Metoda `Sprawdz(...)` zwraca listę błędów: imię min. 2 litery, telefon 9 cyfr (spacje dozwolone), sala z listy, data od dziś do 30 dni, 1–10 osób.
3. Sprawdź ręcznie: konsola, 4 osoby, z przekąskami = 168 zł.

2 Formularz

MainPage.

```
<Grid Padding="20" RowSpacing="8" ColumnSpacing="12"
      RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto"
      ColumnDefinitions="*,Auto">
```

1. Grid z odstępami: Entry imienia (ReturnType Next), Entry telefonu (`Keyboard="Telephone"`), Picker sal, DatePicker (`Format="dd.MM.yyyy"`).
2. Slider 1–10 z przeskokiem do liczb całkowitych i etykietą liczby osób; Switch przekąsek.
3. Przycisk Zarezerwuj (wysokość ≥ 48) i Border z podsumowaniem.
4. W konstruktorze: lista sal, MinimumDate = dziś, MaximumDate = dziś + 30 dni.

3 Obsługa

Zdarzenia.

1. `OnZarezerwuj` (async): walidacja; błędy w jednym `DisplayAlertAsync`; podsumowanie z ceną w złotych.
2. Odczyt daty odporny na pustą wartość: `DtpData.Date is DateTime d ? d : DateTime.Today`.
3. Przetestuj na Windows i w emulatorze: klawiatury, wybór daty spoza zakresu, wszystkie błędy naraz.

4 Porównanie

Wygląd na platformach.

1. Zrzuty formularza na Windows i Androidzie; zapisz 3 różnice wyglądu kontrolek.
2. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy Rezerwacja
- ☐ XAML formularza
- ☐ tabela testów walidacji
- ☐ zrzuty z dwóch platform z opisem różnic

Pytania kontrolne

1. Do czego służy właściwość Keyboard?
2. Dlaczego walidację umieszczamy w osobnej klasie?
3. Jak ograniczyć daty w DatePicker?
4. Dlaczego przycisk w aplikacji mobilnej powinien mieć wysokość co najmniej 44–48?

Kryteria oceny

Kryterium	Punkty
Klasa Rezerwacja z walidacją i ceną	6
Formularz z kontrolkami	6
Obsługa zdarzeń i komunikaty	4
Porównanie platform i pytania	4
Razem	**20**

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

Gra w kości w MAUI (styl INF.04)

Czas: 2 × 45 min | praca indywidualna | Wykład 2.24

Cel: Zbudujesz mobilną grę w kości według specyfikacji w stylu arkusza INF.04: obrazy kości, rzut, blokowanie dotknięciem z przezroczystością, wynik losowania i gry — z klasą Kosc z roku 1.

Po zajęciach potrafisz:

- wyświetla i zmienia obrazy z Resources/Images
- obsługuje gest dotknięcia z parametrem
- realizuje wygląd zgodnie ze specyfikacją (kolory, marginesy, czcionki)
- wykorzystuje istniejącą klasę logiki w nowej platformie

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

Przygotowanie przed zajęciami

- Klasa `Kosc` z roku 1 (zadanie 15) i obrazy `kosc0.png`–`kosc6.png` z folderu `pliki` przy tym zadaniu.

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 Specyfikacja

Wymagania wyglądu (traktuj jak arkusz egzaminacyjny).

1. Tło strony #F0F9FF, marginesy wewnętrzne strony 20, odstęp między elementami 16.
2. Nagłówek „Gra w kości”: 28, pogrubiony, kolor #0E7490, wyśrodkowany.
3. Pięć obrazów kości 64 × 64 z marginesem 4, w FlexLayout zawijanym i wyśrodkowanym.
4. Przycisk „RZUĆ KOŚĆMI”: tło #0E7490, biały pogrubiony tekst, wysokość 50, zaokrąglenie 10.
5. Etykiety: „Rzut N z 3”, „Wynik tego losowania: X” (20), „Wynik gry: Y” (24, pogrubiony).

2 Zasoby i układ

Projekt MAUI `GraWKosci`.

1. Obrazy do Resources/Images (Akcja kompilacji MauiImage); klasa Kosc do projektu bez zmian.
2. Każdy obraz z `TapGestureRecognizer Tapped="OnKoscTapped" CommandParameter="0"` (..."4").
3. Zbuduj stronę zgodnie ze specyfikacją.

3 Logika

Code-behind strony.

```
private void OnKoscTapped(object? sender, TappedEventArgs e)
{
    if (_rzut == 0 || e.Parameter is not string tekst) return;
    int nr = int.Parse(tekst);
    _kosci[nr].Zablokowana = !_kosci[nr].Zablokowana;
    Odswiez();
}
```

1. Rzut: wszystkie niezablokowane kości losują; licznik rzutów; po 3 rzutach przycisk nieaktywny.
2. Dotknięcie kości (po pierwszym rzucie) blokuje/odblokowuje; zablokowana ma `Opacity = 0.4`.
3. Wynik losowania: `Kosc.Punkty(...)`; Zapisz turę dodaje go do wyniku gry i odblokowuje kości.
4. Nowa gra z potwierdzeniem `DisplayAlertAsync` (gdy wynik > 0).

4 Dokumentacja

Jak na egzaminie.

1. Komentarze w kodzie strony (nazwa, opis, parametry).
2. Zrzuty z emulatora: start, po rzucie z 2 zablokowanymi kośćmi, po 3 turach.
3. Lista kontrolna zgodności ze specyfikacją (każdy punkt: tak/nie).
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod XAML i C# strony
- ☐ lista kontrolna zgodności ze specyfikacją
- ☐ trzy zrzuty z emulatora
- ☐ opis wykorzystania klasy Kosc

Pytania kontrolne

1. Jak przekazać numer kości do metody obsługi dotknięcia?
2. Dlaczego w XAML piszemy kosc1.png?
3. Co zmieniłbyś, aby gra miała 6 kości?
4. Które elementy specyfikacji najłatwiej przeoczyć na egzaminie?

Kryteria oceny

Kryterium	Punkty
Zgodność ze specyfikacją wyglądu	6
Rzut i obrazy	4
Blokowanie z przezroczystością	4
Tury i wynik gry	3
Dokumentacja i zrzuty	3
Razem	**20**

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

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).

Nawigacja Shell — sklep

Czas: 2 × 45 min | praca indywidualna | Wykład 2.26

Cel: Zbudujesz aplikację sklepu z TabBar (Produkty, Koszyk, O sklepie), stroną szczegółów otwieraną trasą z parametrem i koszykiem współdzielonym przez usługę.

Po zajęciach potrafisz:

- buduje strukturę aplikacji w AppShell z TabBar
- rejestruje trasy i nawiguje GoToAsync z parametrami
- odbiera parametry QueryProperty w ViewModelu
- współdzieli stan między stronami przez usługę Singleton

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 Usługa

Projekt MAUI `Sklep`; klasa `SklepService` bez MAUI.

1. Rekordy `Produkt(Id, Nazwa, Kategoria, Cena, Opis)` i `PozycjaKoszyka(Produkt, Ilosc)` z właściwością Wartosc.
2. Lista 5 produktów; metody Znajdz(id), Dodaj(produkt, ilosc) — scala ilości tego samego produktu, Usun(id); zdarzenie KoszykZmieniony.
3. Sprawdź: 3 × słuchawki 149 zł + 2 × pendrive 34,50 zł = 516 zł.

2 Shell

AppShell.

1. TabBar z trzema `ShellContent` (Route: produkty, koszyk, info; ContentTemplate).
2. `Routing.RegisterRoute("szczegoly", typeof(SzczegolyPage))` w konstruktorze AppShell.
3. Rejestracja usług, ViewModeli i stron w MauiProgram (usługa i koszyk — Singleton).

3 Strony

Produkty, szczegóły, koszyk.

```
[QueryProperty(nameof(Id), "id")]
public partial class SzczegolyViewModel(SklepService sklep) : ObservableObject
{
    [ObservableProperty] private int _id;
    [ObservableProperty] private Produkt? _produkt;
    partial void OnIdChanged(int value) => Produkt = sklep.Znajdz(value);
}
```

1. Produkty: CollectionView; wybór → `GoToAsync($"szczegoly?id={produkt.Id}")`; po powrocie odznacz wybór.
2. Szczegóły: `[QueryProperty(nameof(Id), "id")]`, wczytanie produktu w `OnIdChanged`, Stepper ilości, Dodaj do koszyka → komunikat i `GoToAsync("..")`.
3. Koszyk: pozycje z przyciskiem Usuń, podsumowanie, EmptyView z przyciskiem „Przejdź do produktów” (`//produkty`).

4 Sprawdzenie

Na Androidzie.

1. Systemowy przycisk Wstecz ze strony szczegółów wraca do listy.
2. Koszyk aktualizuje się po dodaniu produktu bez ręcznego odświeżania.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod SklepService i AppShell
- ☐ kod SzczegolyViewModel
- ☐ schemat nawigacji (strony i trasy)
- ☐ zrzuty trzech kart i szczegółów

Pytania kontrolne

1. Czym różni się ".." od "//produkty"?
2. Dlaczego strony szczegółów nie umieszczamy w TabBar?
3. Jak koszyk dowiaduje się o zmianach?
4. Dlaczego po powrocie trzeba odznaczyć element listy?

Kryteria oceny

Kryterium	Punkty
SklepService	4
AppShell i trasy	4
Szczegóły z parametrem	5
Koszyk z odświeżaniem	4
Pytania	3
Razem	**20**

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

CollectionView — katalog produktów

Czas: 2 × 45 min | praca indywidualna | Wykład 2.27

Cel: Zbudujesz katalog produktów z CollectionView: szablon z obrazem, wyszukiwanie, grupowanie po kategorii, odświeżanie gestem i usuwanie przesunięciem z potwierdzeniem.

Po zajęciach potrafisz:

- projektuje szablon elementu i nagłówek grupy CollectionView
- filtruje i grupuje dane w klasie logiki
- obsługuje RefreshView i EmptyView
- stosuje SwipeView i gest dotknięcia z komendą

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

Przygotowanie przed zajęciami

- Ikony kategorii `audio.png`, `akcesoria.png`, `pamieci.png` z folderu `pliki` przy tym zadaniu.

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 Logika

Projekt MAUI `Katalog`; klasy bez MAUI.

1. Rekord `Produkt(Id, Nazwa, Kategoria, Cena, Obraz)` i 7 produktów w 3 kategoriach.
2. `GrupaProduktow : List<Produkt>` z właściwościami Nazwa i Suma.
3. `KatalogLogika.Filtruj(produkty, fraza)` (nazwa lub kategoria, bez wielkości liter) i `Grupuj` (kategorie alfabetycznie).

2 ViewModel

`ProduktyViewModel`.

1. Kolekcja `Grupy`; `Fraza` → ponowne filtrowanie i grupowanie (OnFrazaChanged).
2. `OdswiezAsync`: symulacja pobrania (Task.Delay), przywrócenie danych, `Odswiezanie = false`.
3. `UsunAsync(Produkt)` z pytaniem; `PokazSzczegolyAsync` — komunikat z ceną.
4. Podsumowanie „Produktów: N".

3 Strona

XAML.

```
<CollectionView ItemsSource="{Binding Grupy}" IsGrouped="True">
  <CollectionView.GroupHeaderTemplate>
    <DataTemplate x:DataType="m:GrupaProduktow">
      <Label Text="{Binding Nazwa}" FontAttributes="Bold" />
    </DataTemplate>
  </CollectionView.GroupHeaderTemplate>
```

1. RefreshView z CollectionView `IsGrouped="True"`; SearchBar w nagłówku; EmptyView z podpowiedzią.
2. GroupHeaderTemplate: nazwa kategorii i liczba sztuk na kolorowym tle.
3. ItemTemplate: obraz 48 × 48, nazwa, kategoria, cena; SwipeView (Reveal) z akcją Usuń; TapGestureRecognizer → szczegóły.
4. Tło elementów z `AppThemeBinding` (jasny/ciemny).

4 Sprawdzenie

Scenariusze.

1. Fraza „audio", „mysz", „xyz" — liczba wyników i EmptyView.
2. Usuń produkt, pociągnij w dół — produkt wraca.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klas logiki i ViewModelu
- ☐ XAML strony
- ☐ tabela scenariuszy
- ☐ zrzuty: grupy, wyszukiwanie, pusta lista

Pytania kontrolne

1. Jakie wymagania ma kolekcja dla CollectionView z grupami?
2. Co się stanie, jeśli nie ustawisz Odswiezanie = false?
3. Czym różni się SwipeItems Mode="Reveal" od "Execute"?
4. Dlaczego filtrowanie umieszczamy w osobnej klasie?

Kryteria oceny

Kryterium	Punkty
Klasy logiki (filtr, grupy)	5
ViewModel	5
Strona z grupami, EmptyView, Swipe, Refresh	7
Pytania	3
Razem	**20**

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

Dane w aplikacji mobilnej — notatnik

Czas: 2 × 45 min | praca indywidualna | Wykład 2.28

Cel: Zbudujesz notatnik mobilny: notatki w bazie SQLite, ustawienia w Preferences, informacja o braku internetu i udostępnianie notatki.

Po zajęciach potrafisz:

- zapisuje dane w SQLite przez sqlite-net-pcl w folderze danych aplikacji
- przechowuje ustawienia w Preferences
- reaguje na zmiany łączności (Connectivity)
- korzysta z funkcji systemu: udostępnianie (Share)

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
- **Pakiety sqlite-net-pcl, SQLitePCLRaw.bundle_green, CommunityToolkit.Maui** learn.microsoft.com/dotnet/maui/data-cloud/database-sqlite

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.
- Nie zapisuj w notatkach testowych danych osobowych ani haseł.

Przebieg zadania

1 Model i repozytorium

Projekt MAUI `NotatnikMobilny`.

1. `Notatka`: [PrimaryKey, AutoIncrement] Id, [MaxLength(100), NotNull] Tytul, Tresc, Utworzono, Zmieniono.
2. `NotatkiRepo`: połączenie tworzone przy pierwszym użyciu w `FileSystem.AppDataDirectory`, CreateTableAsync.
3. Metody: WszystkieAsync(alfabetycznie), PobierzAsync(id), ZapiszAsync (Insert/Update, aktualizuje datę zmiany), UsunAsync.

2 Lista i edycja

Dwie strony z ViewModelami.

1. Lista: CollectionView, ToolbarItem „Nowa", dotknięcie otwiera edycję (`edycja?id=…`), przesunięcie w lewo — Udostępnij, w prawo — Usuń.
2. Edycja: `[QueryProperty]` id (0 = nowa), Entry tytułu, Editor treści, Zapisz (aktywny przy niepustym tytule) i powrót.
3. Odświeżanie listy w `OnAppearing` strony.

3 Ustawienia i łączność

Preferences i Connectivity.

```
[ObservableProperty]
private bool _alfabetycznie = Preferences.Default.Get("sortAlfabetycznie", false);

partial void OnAlfabetycznieChanged(bool value)
{
    Preferences.Default.Set("sortAlfabetycznie", value);
    _ = WczytajAsync();
}
```

1. Przełącznik sortowania A–Z i Stepper rozmiaru czcionki zapisywane w Preferences; po restarcie aplikacji odtworzone.
2. Pasek „Brak internetu" widoczny, gdy `Connectivity.Current.NetworkAccess != NetworkAccess.Internet`; aktualizowany zdarzeniem ConnectivityChanged.
3. Sprawdź w emulatorze: tryb samolotowy włączony i wyłączony.

4 Trwałość

Sprawdź, gdzie są dane.

1. Zamknij aplikację w emulatorze (z listy ostatnich), uruchom ponownie — notatki i ustawienia zostają.
2. Odinstaluj i zainstaluj ponownie — co się stało z danymi? Zapisz wniosek.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod modelu i repozytorium
- ☐ kod ViewModeli
- ☐ wyniki testów trwałości i łączności
- ☐ zrzuty listy, edycji i paska braku internetu

Pytania kontrolne

1. Kiedy użyjesz Preferences, a kiedy bazy?
2. Gdzie fizycznie znajduje się plik bazy aplikacji MAUI?
3. Dlaczego połączenie z bazą tworzymy leniwie (przy pierwszym użyciu)?
4. Co dzieje się z danymi po odinstalowaniu aplikacji?

Kryteria oceny

Kryterium	Punkty
Model i repozytorium SQLite	6
Lista i edycja z nawigacją	5
Preferences i Connectivity	4
Udostępnianie i testy trwałości	3
Pytania	2
Razem	**20**

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

Motywy i różne ekrany

Czas: 2 × 45 min | praca indywidualna | Wykład 2.29

Cel: Przygotujesz aplikację MAUI działającą w motywie jasnym i ciemnym (systemowym lub wybranym) oraz zmieniającą układ przy obrocie ekranu.

Po zajęciach potrafisz:

- definiuje kolory zależne od motywu AppThemeBinding
- przełącza motyw aplikacji UserAppTheme i zapamiętuje wybór
- zmienia układ zależnie od orientacji (VisualStateManager, OrientationStateTrigger)
- testuje wygląd na różnych platformach i ekranach

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 Android**a — emulator wymaga wirtualizacji sprzętowej; awaryjnie uruchamiaj jako Windows Machine learn.microsoft.com/dotnet/maui/get-started/installation

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 Kolory i style

Słownik `Resources/Styles/Motywy.xaml` dołączony w App.xaml.

```
<Setter Property="BackgroundColor"
        Value="{AppThemeBinding Light={StaticResource TloJasne},
                                Dark={StaticResource TloCiemne}}" />
```

1. Pary kolorów dla jasnego i ciemnego motywu: tło, karta, tekst, akcent.
2. Style z `AppThemeBinding`: ContentPage (tło), Label (tekst), Naglowek, Karta (Border), Button.
3. Sprawdź kontrast tekstu w obu motywach (min. 4,5:1).

2 Wybór motywu

`MotywService`.

1. Picker: Systemowy / Jasny / Ciemny; zapis w Preferences.
2. `Application.Current.UserAppTheme` = Unspecified / Light / Dark; przywrócenie w konstruktorze App.
3. Etykieta z motywem systemu (`PlatformAppTheme`) i aplikacji (`RequestedTheme`), odświeżana zdarzeniem RequestedThemeChanged.

3 Orientacja

Układ strony.

1. Grid z VisualStateManager: pionowo jedna kolumna, poziomo dwie (OrientationStateTrigger).
2. Druga karta przenosi się pod pierwszą lub obok niej; nagłówek w poziomie zajmuje 2 kolumny.
3. Informacja o ekranie: rozdzielczość, gęstość, orientacja (`DeviceDisplay.Current.MainDisplayInfo`).

4 Testy

Na dwóch platformach.

1. Windows: zmiana trybu aplikacji w ustawieniach systemu przy opcji „Systemowy”.
2. Emulator: ciemny motyw w szybkich ustawieniach, obrót ekranu.
3. Tabela 6 zrzutów (2 platformy × motywy/orientacje). Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ Motywy.xaml
- ☐ kod MotywService
- ☐ tabela zrzutów
- ☐ pomiary kontrastu

Pytania kontrolne

1. Czym różni się PlatformAppTheme od RequestedTheme?
2. Jak aplikacja zapamiętuje wybrany motyw?
3. Dlaczego jednostki MAUI nie są pikselami ekranu?
4. Co musi się zmienić w układzie przy orientacji poziomej i dlaczego?

Kryteria oceny

Kryterium	Punkty
Kolory i style z AppThemeBinding	5
Wybór i zapamiętanie motywu	5
Układ zależny od orientacji	5
Testy na platformach	3
Pytania	2
Razem	**20**

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

Próbny egzamin INF.04 — część 1: konsola i testy

Czas: 2 × 45 min | praca samodzielna w warunkach egzaminacyjnych (90 min) | Wykład 2.29

Cel: W 90 minut wykonasz część konsolową próbnego arkusza: klasę KodRabatowy według specyfikacji, program testujący, testy jednostkowe i dokumentację.

Po zajęciach potrafisz:

- realizuje specyfikację klasy dokładnie co do nazw i typów
- obsługuje błędne dane wyjątkami i komunikatami
- pisze testy jednostkowe wartości granicznych
- dokumentuje kod i wykonuje zrzuty w wymaganej formie

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

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.
- Pracujesz samodzielnie, bez internetu i asystentów AI; dozwolona jest dokumentacja wbudowana w Visual Studio (F1, IntelliSense).
- Rozwiązanie zapisz w folderze `Egzamin_<numer z dziennika>` z podfolderami podanymi w poleceniu.

Przebieg zadania

1 Klasa KodRabatowy

Folder `konsola`. Nazwy DOKŁADNIE jak poniżej.

1. Statyczna właściwość `int LiczbaKodow` (prywatny setter) — liczba poprawnie utworzonych kodów.
2. Właściwości tylko do odczytu: `string Kod`, `int Procent`, `DateOnly WaznyDo`.
3. Konstruktor `(string kod, int procent, DateOnly waznyDo)`: kod w formacie trzy wielkie litery A–Z, myślnik, cztery cyfry (np. ABC-1234) — inaczej `ArgumentException`; procent 5–50 — inaczej `ArgumentOutOfRangeException`.
4. `bool CzyWazny(DateOnly dzien)` — ważny do dnia WaznyDo włącznie.
5. `decimal CenaPoRabacie(decimal cena, DateOnly dzien)` — cena ujemna: `ArgumentOutOfRangeException`; kod nieważny: cena bez zmian; ważny: cena × (100 − Procent) / 100 zaokrąglona do 0,01 (połówki w górę).
6. Statyczna `string Generuj(Random los)` — losowy kod w poprawnym formacie.

2 Program

Program testujący.

```
ABC-1234 (-10%, ważny do 18.12.2026) – ważny, cena po rabacie: 134,99 zł
PAD-2026 (-25%, ważny do 18.11.2026) – ważny, cena po rabacie: 112,49 zł
OLD-0001 (-50%, ważny do 17.11.2026) – NIEWAŻNY, cena po rabacie: 149,99 zł
```

1. Utwórz trzy kody: ABC-1234 (10%, ważny 30 dni od dziś), PAD-2026 (25%, ważny tylko dziś), OLD-0001 (50%, wygasł wczoraj).
2. Pobierz cenę (powtarzaj pytanie przy błędnej lub ujemnej) i wypisz dla każdego kodu: kod, procent, datę, ważność i cenę po rabacie.
3. Poproś o własny kod (rabat 15%, ważny 7 dni); błędny format — komunikat bez zakończenia programu.
4. Wypisz dwa wylosowane kody i wartość LiczbaKodow.

3 Testy

Projekt MSTest w folderze `testy`.

1. Co najmniej 12 przypadków: poprawny konstruktor, 4 złe formaty, procent 4/51 oraz 5/50, ceny (w tym 99,99 zł przy 25% = 74,99 zł), kod przeterminowany, ujemna cena, licznik.
2. Zrzut Eksploratora testów: `testy.png`.

4 Dokumentacja

Wymagany format komentarza.

1. Nad klasą i każdą metodą: nazwa, opis, parametry, zwracany typ i opis, autor (numer z dziennika).
2. Zrzut działania programu dla ceny 149,99 i kodu PAD-0007: `konsola.png`.
3. Sprawdź nazwy plików i folderów, usuń foldery bin i obj z kopii do oddania.

W sprawozdaniu umieść

- ☐ folder rozwiązania (kod, testy, zrzuty)
- ☐ samoocena według kryteriów i czas pracy każdej części

Pytania kontrolne

1. Która część specyfikacji była najtrudniejsza do dokładnego odwzorowania?
2. Jakie wartości graniczne sprawdziłeś testami?
3. Ile czasu zostało na dokumentację?

Kryteria oceny

Kryterium	Punkty
Klasa zgodna ze specyfikacją (nazwy, typy, wyjątki)	7
Program testujący	4
Testy jednostkowe (min. 12)	5
Dokumentacja i zrzuty	4
Razem	**20**

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

Próbny egzamin INF.04 — część 2: aplikacja z interfejsem

Czas: 2 × 45 min | praca samodzielna w warunkach egzaminacyjnych (90 min) | Wykład 2.29

Cel: W 90 minut wykonasz część z interfejsem próbnego arkusza: aplikację kasy (WPF lub MAUI — do wyboru) według makiety i specyfikacji, korzystającą z klasy z części 1, z dokumentacją.

Po zajęciach potrafisz:

- buduje interfejs dokładnie według specyfikacji wyglądu
- łączy interfejs z logiką z części 1
- pokazuje komunikaty błędów zgodnie z wymaganiami
- dokumentuje aplikację i wykonuje wymagane zrzuty

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

Przygotowanie przed zajęciami

- Klasa KodRabatowy z części 1 (zadanie 30).

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.
- Pracujesz samodzielnie, bez internetu i asystentów AI; dozwolona jest dokumentacja wbudowana w Visual Studio (F1, IntelliSense).
- Rozwiązanie zapisz w folderze `Egzamin_<numer z dziennika>` z podfolderami podanymi w poleceniu.

Przebieg zadania

1 Logika kasy

Folder `aplikacja`. Klasa `Kasa` (bez interfejsu).

1. Konstruktor `Kasa(DateOnly dzisiaj)` tworzy kody ABC-1234, PAD-2026, OLD-0001 jak w części 1.
2. Metoda `Oblicz(string? cenaTekst, string? kodTekst, DateOnly dzien)` zwraca rekord `Wynik(bool Poprawny, string Komunikat, decimal? Cena)`.
3. Komunikaty: zła cena — „Podaj poprawną cenę, np. 149,99.”; pusty kod — „Bez kodu: X zł”; nieznany — „Nieznany kod rabatowy.”; wygasły — „Kod ... wygasł ...”; poprawny — „Do zapłaty: X zł (rabat N%)”.
4. Kod porównuj bez spacji na brzegach i bez względu na wielkość liter.

2 Interfejs — specyfikacja

WPF (okno) lub MAUI (strona).

```
var w = _kasa.Oblicz(TxtCena.Text, TxtKod.Text, DateOnly.FromDateTime(DateTime.Today));
TxtWynik.Text = w.Komunikat;
TxtWynik.Foreground = w.Poprawny ? new SolidColorBrush(Color.FromRgb(0x0E, 0x74, 0x90)) :
Brushes.Red;
```

1. Tytuł „Kasa — kody rabatowe”; WPF: okno 420 × 380, bez zmiany rozmiaru; tło #F0F9FF; marginesy 20.
2. Nagłówek „Kasa sklepu”: 22, pogrubiony, kolor #0E7490.
3. Etykiety i pola: „Cena [zł]:” i „Kod rabatowy (opcjonalnie):” (pole kodu: wielkie litery, max 8 znaków).
4. Przycisk „Oblicz”: tło #0E7490, biały pogrubiony tekst, wysokość 36 (MAUI: 48), domyślny (Enter).
5. Wynik: 20, pogrubiony; kolor #0E7490 dla poprawnego, czerwony dla błędu. WPF: lista historii obliczeń pod wynikiem.

3 Sprawdzenie

Dane testowe.

1. 149,99 + brak kodu; 149,99 + abc-1234; 149,99 + PAD-2026; 149,99 + OLD-0001; 149,99 + XYZ-0000; „abc” + ABC-1234.
2. Zrzuty: `aplikacja_poprawny.png` (PAD-2026) i `aplikacja_blad.png` (OLD-0001).

4 Dokumentacja

Jak w części 1.

1. Komentarze nad klasą Kasa, metodą Oblicz i metodą obsługi przycisku.
2. Kontrola nazw plików i folderów; usuń bin i obj z kopii.

W sprawozdaniu umieść

- ☐ folder rozwiązania (kod, zrzuty)
- ☐ lista kontrolna zgodności ze specyfikacją
- ☐ samoocena

Pytania kontrolne

1. Które elementy specyfikacji wyglądu łatwo przeoczyć?
2. Dlaczego logikę kasy umieszczasz w osobnej klasie?
3. Co zmieniłbyś, aby ta sama logika działała w drugiej technologii?

Kryteria oceny

Kryterium	Punkty
Klasa Kasa i komunikaty	6
Interfejs zgodny ze specyfikacją	7
Działanie dla danych testowych	3
Dokumentacja i zrzuty	4
Razem	**20**

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

Projekt końcowy — specyfikacja

Czas: 2 × 45 min | praca zespołowa (2–3 osoby) | Wykład 2.30

Cel: W zespole przygotujecie specyfikację aplikacji MVVM z bazą danych: wymagania, makiety, architekturę warstw, plan testów i podział pracy w Git.

Po zajęciach potrafisz:

- formułuje sprawdzalne wymagania zespołowego projektu
- projektuje architekturę: widoki, ViewModele, usługi, repozytorium, model danych
- planuje pracę zespołu w zgłoszeniach i gałęziach Git
- planuje testy jednostkowe i ręczne

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
- **Git for Windows** — lub Git wbudowany w Visual Studio
  git-scm.com/downloads/win
- **Konto GitHub (bezpłatne)** — lub szkolne konto GitHub Education
  github.com/signup
- **Narzędzie do makiet (Penpot, Figma lub papier)**
  penpot.app

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 Temat i wymagania

Zatwierdźcie temat z nauczycielem.

1. Wymagania obowiązkowe: WPF lub MAUI w MVVM (CommunityToolkit.Mvvm), baza SQLite (EF Core lub sqlite-net), min. 3 widoki/strony, walidacja, DI, testy ViewModeli, publikacja.
2. 10–15 ponumerowanych wymagań funkcjonalnych i 3 pozafunkcjonalne (wydajność, dostępność, motyw).
3. Przykładowe tematy: wypożyczalnia sprzętu szkolnego, dziennik treningów, rezerwacja sal, budżet domowy z kategoriami, quiz z pytaniami w bazie.

2 Architektura

Dokument w folderze `docs`.

1. Diagram encji bazy (encje, relacje 1:N/N:M, klucze).
2. Lista widoków i ViewModeli z komendami; interfejsy usług (repozytorium, dialogi, ustawienia).
3. Makiety każdego widoku z nazwami kontrolek i przejściami.

3 Organizacja

Repozytorium zespołu.

1. Repozytorium na GitHub z ochroną main, szablonem pull requesta i .gitignore.
2. Zgłoszenia (Issues) dla wymagań z przypisaniem osób; tablica projektu (Projects) z kolumnami Do zrobienia / W toku / Gotowe.
3. Harmonogram do zadań 33 i 34; plan co najmniej 10 testów jednostkowych.
4. Przedstawcie specyfikację nauczycielowi (5 minut) i wprowadźcie poprawki.

W sprawozdaniu umieść

- ☐ dokument specyfikacji (wymagania, diagram encji, architektura, makiety)
- ☐ adres repozytorium i zrzut tablicy projektu
- ☐ podział pracy

Pytania kontrolne

1. Które wymaganie jest największym ryzykiem i jak je ograniczycie?
2. Jak podzieliliście pracę, aby uniknąć konfliktów w Git?
3. Które klasy będą testowane jednostkowo?

Kryteria oceny

Kryterium	Punkty
Wymagania	5
Model danych i architektura	6
Makiety	4
Organizacja repozytorium i plan	5
Razem	**20**

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

Projekt końcowy — implementacja

Czas: 2 × 45 min | praca zespołowa (2–3 osoby), kilka jednostek zajęć | Wykład 2.30

Cel: Zrealizujecie projekt zgodnie ze specyfikacją: warstwa danych, ViewModele, widoki i testy — w małych krokach z pull requestami i przeglądem kodu.

Po zajęciach potrafisz:

- implementuje warstwę danych z migracjami i repozytorium
- implementuje ViewModele bez zależności od widoków
- pracuje w zespole przez gałęzie i pull requesty z przeglądem
- pisze testy ViewModeli z atrapami usług

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
- **Git for Windows** — lub Git wbudowany w Visual Studio git-scm.com/downloads/win
- **Konto GitHub (bezpłatne)** — lub szkolne konto GitHub Education github.com/signup
- **Pakiet NuGet CommunityToolkit.Mvvm 8.4** — instalacja: Zarządzaj pakietami NuGet nuget.org/packages/CommunityToolkit.Mvvm
- **Pakiety Microsoft.EntityFrameworkCore.Sqlite i .Tools 10.x** — wersja zgodna z .NET 10 nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite

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 Warstwa danych

Pierwszy etap.

1. Encje, kontekst, migracja początkowa i dane startowe; repozytorium za interfejsem.
2. Krótki test integracyjny lub program sprawdzający zapis i odczyt.
3. Plik bazy w .gitignore; migracje w repozytorium.

2 ViewModele i widoki

Według makiet.

1. Kontener DI w App; ViewModele z komendami i walidacją; dialogi przez usługę.
2. Widoki bez logiki w code-behind; style i motyw w słownikach zasobów.
3. Każde wymaganie = gałąź + pull request z numerem zgłoszenia i zrzutem.

3 Testy i przegląd

Jakość.

```
[TestMethod]
public void Zapisz_PoprawneDane_DodajeRekordIInformuje()
{
    var repo = new PamiecRepo(); var dialog = new AtrapaDialogu();
    var vm = new FormularzViewModel(repo, dialog) { Nazwa = "Projektor" };
    vm.ZapiszCommand.Execute(null);
    Assert.AreEqual(1, repo.Dodane.Count);
}
```

1. Min. 10 testów ViewModeli z atrapami repozytorium.
2. Każdy pull request z przeglądem drugiej osoby według listy kontrolnej z lekcji 2.30.
3. Tabela realizacji wymagań aktualizowana po każdym scaleniu.
4. Na koniec etapu: pokaz działającej wersji nauczycielowi i lista poprawek do zadania 34.

W sprawozdaniu umieść

- ☐ tabela realizacji wymagań
- ☐ lista pull requestów z autorami i recenzentami
- ☐ wynik testów i pokrycia
- ☐ zrzuty widoków

Pytania kontrolne

1. Który przegląd kodu wykrył istotny problem?
2. Jak rozwiązaliście konflikty scalania?
3. Które wymaganie zmieniliście i dlaczego?

Kryteria oceny

Kryterium	Punkty
Realizacja wymagań	8
Architektura MVVM i DI	4
Testy	4
Praca w Git (PR, przeglądy)	4
Razem	**20**

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

Projekt końcowy — publikacja i obrona

Czas: 2 × 45 min | praca zespołowa (2–3 osoby) | Wykład 2.30

Cel: Przygotujecie wydanie projektu (publikacja, dokumentacja, testy na innym komputerze) i obronicie rozwiązanie przed klasą i nauczycielem.

Po zajęciach potrafisz:

- publikuje aplikację w wersji gotowej do przekazania
- przygotowuje dokumentację użytkownika i techniczną
- prezentuje architekturę i działanie aplikacji
- odpowiada na pytania o własny kod

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu
- komputer bez środowiska programisty do testu wydania
- projektor do prezentacji

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
- **Git for Windows** — lub Git wbudowany w Visual Studio git-scm.com/downloads/win
- **Konto GitHub (bezpłatne)** — lub szkolne konto GitHub Education github.com/signup

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 Wydanie

Według lekcji 2.20.

1. Wersja 1.0.0, ikona, metadane; publikacja WPF (self-contained) lub MAUI (pakiet Windows / APK Androida — dla chętnych).
2. Test na innym komputerze: instalacja, pierwsze uruchomienie (utworzenie bazy), scenariusz główny.
3. Wydanie na GitHub (Releases) z archiwum i opisem zmian.

2 Dokumentacja

Folder `docs` i README.

1. README: opis, wymagania, uruchomienie, zrzuty, autorzy i podział pracy.
2. Dokumentacja techniczna: architektura, diagram encji, lista wymagań ze stanem realizacji.
3. Komentarze dokumentacyjne w klasach usług i ViewModeli.

3 Obrona

10 minut na zespół + pytania.

1. Prezentacja: problem, pokaz aplikacji, architektura (diagram), ciekawa decyzja techniczna, testy, co dalej.
2. Każda osoba omawia własną część i odpowiada na pytania o swój kod.
3. Samoocena zespołu według kryteriów projektu.
4. Po obronie: poprawki zgłoszone przez klasę jako nowe zgłoszenia (Issues) do ewentualnej wersji 1.1.

W sprawozdaniu umieść

- ☐ adres wydania (Release)
- ☐ README i dokumentacja techniczna
- ☐ materiały prezentacji
- ☐ samoocena i ocena wkładu osób

Pytania kontrolne

1. Co zrobilibyście inaczej, zaczynając projekt od nowa?
2. Która decyzja architektoniczna najbardziej się opłaciła?
3. Jak rozwinąć aplikację (np. synchronizacja z serwerem, wersja mobilna)?

Kryteria oceny

Kryterium	Punkty
Wydanie i test na innym komputerze	5
Dokumentacja	5
Prezentacja	4
Odpowiedzi na pytania o własny kod	6
Razem	**20**

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