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).