Kompletny Materiał do Nauki

Programowanie . NET

Przygotowanie do egzaminu

29 listopada 2025

Spis treści

Rozdział 1

Programowanie AI - Copilot, VS Code

1.1 Wstęp

GitHub Copilot to asystent AI w Visual Studio Code, który wspomaga pracę programisty. Zajęcia skupiają się na zaawansowanych funkcjach, od Agent Mode po personalizację poprzez instrukcje.

1.2 Agent Mode i jego zastosowania

1.2.1 Definicja

Agent Mode to tryb, w którym Copilot działa automatycznie bez konieczności potwierdzania każdego kroku. Różni się od klasycznego trybu interaktywnego.

1.2.2 Praktyczne zastosowania

- **Build i test**: automatyczne budowanie i testowanie projektów
- **Długotrwałe zadania**: delegowanie złożonych operacji
- **Code review**: analiza kodu przez agenta
- **Generowanie endpointów API**: szybkie tworzenie usług

1.2.3 Tryby pracy

Interaktywny Copilot prosi o potwierdzenie każdego kroku

W pełni automatyczny Agent wykonuje całe zadanie bez przerw

1.3 Allow / Deny List – bezpieczeństwo

1.3.1 Rola listy dozwolonych komend

Allow List zapewnia bezpieczeństwo przez limitowanie komend, które agent może wykonać automatycznie.

1.3.2 Praktyczne przykłady

```
// Dozwolone komendy
- dotnet build
- dotnet test
- dotnet run

// Zakazane komendy
- rm -rf .
- del *.*
```

1.3.3 Konfiguracja

1. Otwórz Settings → Copilot → Experimental
2. Dodaj komendy do Allow List:
   - `dotnet build`
   - `dotnet test`
3. Po skonfigurowaniu projekt buduje się automatycznie bez pytania

1.4 Zarządzanie limitami - Max Requests

1.4.1 Wpływ na wydajność

Domyślnie Max Requests wynosi 20. Limit wpływa na:

- Ilość automatycznych operacji
- Płynność pracy z agentem
- Czas wykonywania długotrwałych zadań

1.4.2 Praktyczne ustawienia

Projekt	**Max Requests**	**Uzasadnienie**
Mały projekt	20	Domyślnie wystarczy
Średni projekt	50-100	Lepszy flow pracy
Duży projekt	100+	Złożone zadania

1.5 Copilot Instructions - personalizacja

1.5.1 Automatyczne generowanie

1. Otwórz chat Copilota
2. Kliknij "Customize chat"→ "Generate Instructions"
3. Copilot skanuje projekt i tworzy `copilot-instructions.md`

1.5.2 Zawartość wygenerowanych instrukcji

- Struktura projektu
- Komendy budowania i uruchamiania
- Konwencje kodu (PascalCase, camelCase)
- DTO z atrybutami JSON
- Reguły CSS

1.5.3 Przykład własnych reguł

```
# Copilot Instructions

## Konwencje kodu
- Klasy i metody: PascalCase
- Zmienne: camelCase
- Pola private: _camelCase

## DTO
Wszystkie DTO maj  atrybuty JSON:
[JsonPropertyName("fieldName")]

## C# Best Practices
- U ywaj using statements
- Dokumentuj publiczne metody XML comments
```

1.5.4 Rozdzielanie zasad dla backendu i frontendu

Utwórz oddzielne pliki:

- `. github/csharp-guidelines.md`
- `.github/blazor-guidelines.md`

1.6 Snooze Completions - balans AI/samodzielne kodowanie

1.6.1 Kiedy wyciszyć Copilota

- Nauka nowych konceptów
- Ćwiczenie świadomego programowania
- Unikanie uzależnienia od AI

1.6.2 Jak korzystać

1. Kliknij ikonę Copilota w pasku statusu
2. Wybierz "Snooze"(np. 5 minut)
3. Pracuj samodzielnie
4. Po upłynięciu czasu podpowiedzi wracają

1.7 Custom Chat Modes

1.7.1 Beast Mode

Specjalny tryb, w którym AI rozbija zadanie na kroki i realizuje je sekwencyjnie.

1.7.2 Konfiguracja

1. Utwórz `. github/chat-modes/beast.json`
2. Uruchom Agent Mode
3. Wybierz Beast Mode z listy trybów
4. Poproś o dodanie endpointu API

1.8 Coding Agents - wirtualny współprogramista

1.8.1 Przypisywanie zadań

- Utwórz issue na GitHub
- Przypisz je do @copilot
- Agent automatycznie tworzy branch i PR

1.8.2 Delegowanie pracy

1. W VS Code otwórz panel agenta
2. Kliknij +, dodaj nowe zadanie
3. Agent tworzy nowy branch i sesję
4. Pracuje w tle, podczas gdy ty kodując lokalnie

1.8.3 Code review od agenta

- Dodaj Copilota jako reviewera PR
- Agent sugeruje ulepszenia
- Może proponować użycie ILogger zamiast Console. WriteLine

1.9 Integracja z GitHub Actions

1.9.1 Automatyzacja setupu środowiska

Plik `.github/workflows/copilot-setup-steps.yml`:

```
    name: Setup Steps
    on: [push]
    jobs:
    setup:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Setup .NET 9
    uses: actions/setup-dotnet@v3
    with:
    dotnet-version: '9.0'
    - name: Restore packages
    run: dotnet restore
```

1.10 Zarządzanie modelami AI

1.10.1 Dostępne modele

- **GPT-5**: Najnowszy, najlepsze rezultaty
- **GPT-5 mini**: Szybszy, mniej zasobów
- **Starsze modele**: Można usunąć

1.10.2 Zmiana modelu

1. VS Code 1.103+
2. Otwórz model selector
3. Wybierz preferowany model
4. Usuń starsze modele z listy

1.11 MCP Auto Start i narzędzia developerskie

1.11.1 Rola MCP

MCP (Model Context Protocol) udostępnia dodatkowe narzędzia dla Copilota.

1.11.2 Konfiguracja

1. Settings → MCP Auto Start → ustaw "always"
2. Configure Tools → przejrzyj dostępne narzędzia
3. Wyłącz nieużywane, aby zmniejszyć szum

1.12 Statystyki użycia AI

1.12.1 Analiza wydajności

- AI vs typing - co naprawdę daje Copilot?
- Eksperymenty porównawcze
- Eksport danych do analizy

1.12.2 Interpretacja danych

1. Włącz AI Stats
2. Sprawdź wykres w dolnym rogu (24h)
3. Porównaj produktywność z/bez AI

1.13 Pytania kontrolne

1. Jaka jest różnica między trybem interaktywnym a w pełni automatycznym?
2. Dlaczego Allow List jest ważny dla bezpieczeństwa?
3. Jak wygenerować automatyczne instrukcje dla projektu?
4. Kiedy warto używać Snooze Completions?
5. Jak przypisać zadanie do Coding Agent?
6. Jakie są dostępne modele AI w Copilot?

Rozdział 2

Tworzenie API REST

2.1 Wstęp

REST (Representational State Transfer) to architektura do tworzenia usług sieciowych. API REST używa standardowych czasowników HTTP i statusów do komunikacji między klientem a serwerem.

2.2 Stworzenie nowego Web API w Visual Studio

2.2.1 Inicjalizacja projektu

```
    dotnet new webapi -n ShopAPI. Api
    cd ShopAPI.Api
    code .
    dotnet run
```

2.2.2 Gdzie uruchomić się aplikacja

`https://localhost:7294/swagger/index.html`

2.3 HTTPS i certyfikat SSL

2.3.1 Weryfikacja certyfikatu

```
    dotnet dev-certs https --check
    dotnet dev-certs https --trust
```

2.3.2 Omijanie zabezpieczenia SSL w Chrome

1. Wpisz: `chrome://flags/#allow-insecure-localhost`
2. Ustaw na "Enabled"
3. Zrestartuj Chrome

2.4 Testowanie API

2.4.1 Narzędzia testowania

Swagger Interfejs webowy, intuicyjny, bezpośrednio w aplikacji

curl Minimalistyczne, działające z konsoli, idealne dla skryptów

Postman Rozbudowane, możliwość zapisywania kolekcji

Pliki . http Native w VS Code, idealnie dla . NET developerów

Przeglądarka Tylko dla GET, szybka weryfikacja

2.4.2 Przykład z Swagger

API automatycznie generuje dokumentację na `/swagger`

2.4.3 Przykład z curl

```
# GET
curl https://localhost:7294/api/cities

# POST
curl -X POST https://localhost:7294/api/cities \
-H "Content-Type: application/json" \
-d '{"name":"Warsaw","country":"Poland"}'
```

2.5 Czasowniki HTTP i dobre praktyki REST

2.5.1 Podstawowe metody

Metoda	Opis	Przykład
GET	Pobieranie danych	GET /api/cities
POST	Dodawanie nowych danych	POST /api/cities
PUT	Aktualizacja całego zasobu	PUT /api/cities/1
PATCH	Częściowa aktualizacja	PATCH /api/cities/1
DELETE	Usunięcie zasobu	DELETE /api/cities/1

2.5.2 Zaawansowane metody

HEAD Pobiera tylko nagłówki (sprawdzenie, czy zasób istnieje)

OPTIONS Zwraca dostępne metody dla endpointu

2.5.3 Idempotencja

- **Idempotentne**: GET, PUT, DELETE (wielokrotne wywołanie = ten sam efekt)
- **Nie idempotentne**: POST (za każdym razem tworzy nowy zasób)

2.6 Przykład CitiesController

2.6.1 Implementacja endpointów

```
[ApiController]
[Route("api/[controller]")]
public class CitiesController : ControllerBase
{
    private static List<CityDto> cities = new();

    // GET /api/cities
    [HttpGet]
    public ActionResult<List<CityDto>> GetAll()
    => Ok(cities);

    // GET /api/cities/{id}
    [HttpGet("{id}")]
    public ActionResult<CityDto> GetById(int id)
    {
        var city = cities.FirstOrDefault(c => c.Id == id);
        if (city == null) return NotFound();
        return Ok(city);
    }

    // POST /api/cities
    [HttpPost]
    public ActionResult<CityDto> Create([FromBody] CityDto dto
       )
    {
        var newCity = new CityDto
        {
            Id = cities.Count + 1,
            Name = dto.Name,
            Country = dto. Country
        };
        cities.Add(newCity);
        return CreatedAtAction(nameof(GetById),
        new { id = newCity. Id }, newCity);
    }

    // PUT /api/cities/{id}
    [HttpPut("{id}")]
    public IActionResult Update(int id,
    [FromBody] CityDto dto)
    {
```

```
            var city = cities.FirstOrDefault(c => c.Id == id);
            if (city == null) return NotFound();

            city.Name = dto.Name;
            city.Country = dto.Country;
            return NoContent();
        }

        // PATCH /api/cities/{id}
        [HttpPatch("{id}")]
        public IActionResult PartialUpdate(int id,
        [FromBody] JsonPatchDocument<CityDto> patch)
        {
            var city = cities.FirstOrDefault(c => c.Id == id);
            if (city == null) return NotFound();

            var cityDto = new CityDto
            {
                Name = city.Name,
                Country = city.Country
            };
            patch.ApplyTo(cityDto);

            city.Name = cityDto.Name;
            city.Country = cityDto.Country;
            return NoContent();
        }

        // DELETE /api/cities/{id}
        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            var city = cities.FirstOrDefault(c => c.Id == id);
            if (city == null) return NotFound();

            cities.Remove(city);
            return NoContent();
        }
    }
```

2.7 Statusy HTTP

2.7.1 Statusy sukcesu

Kod	Znaczenie	Przykład
200	OK	Poprawne pobranie danych
201	Created	Nowy zasób utworzony
204	No Content	Operacja udana, brak danych w odpowiedzi

2.7.2 Statusy błędów

Kod	Znaczenie	Przykład
400	Bad Request	Zła składnia lub brakujące pole
401	Unauthorized	Brak autoryzacji (brak JWT tokena)
403	Forbidden	Brak uprawnień (user nie jest adminem)
404	Not Found	Zasób nie istnieje
409	Conflict	Konflikt danych
500	Server Error	Wyjątek w kodzie

2.8 DTO i modele

2.8.1 Definicja DTO

DTO (Data Transfer Object) to obiekt wymiany danych między API a klientem.

2.8.2 Przykład

```
public class CityDto
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Country { get; set; }
}
```

2.8.3 Dlaczego DTO?

- Oddzielenie API od logiki biznesowej
- Bezpieczeństwo (nie wysyłamy całej encji z bazą)
- Elastyczność (łatwa zmiana struktury)
- Walidacja danych

2.9 Schematy i Swagger

2.9.1 Generowanie dokumentacji

Swagger automatycznie generuje `swagger.json` na `/swagger/v1/swagger.json`

2.9.2 Wykorzystanie schematu

- Dokumentacja API dla frontend developerów
- Generowanie klienta automatycznie (NSwag, OpenAPI Generator)
- Walidacja zapytań

2.10 Wysyłanie zapytań z aplikacji klienckiej

2.10.1 Konsola - przykład

```
dotnet new console -n ShopAPI. Client
cd ShopAPI.Client
dotnet add package System.Net.Http.Json

// Program.cs
using System. Net.Http. Json;

var client = new HttpClient();
client.BaseAddress = new Uri("https://localhost:7294");

// GET
var cities = await client.GetFromJsonAsync<List<CityDto>>(
"/api/cities");
Console.WriteLine($"Miasta: {string.Join(", ",
    cities.Select(c => c.Name))}");

// POST
var newCity = new CityDto
{
    Name = "Warsaw",
    Country = "Poland"
};
var response = await client.PostAsJsonAsync(
"/api/cities", newCity);
Console.WriteLine($"Dodano miasto, status: {response.
   StatusCode}");
```

2.11 Autentykacja i autoryzacja

2.11.1 Flow OAuth 2.0 + PKCE

1. Użytkownik kliknie "Zaloguj się"
2. Aplikacja generuje `code_challenge` i `state`
3. Redirect do providera (Google / Microsoft Entra ID)
4. Provider zwraca `code`
5. Aplikacja wymienia `code` na `access_token`
6. Aplikacja wysyła `access_token` w nagłówku `Authorization: Bearer`
7. API weryfikuje token i zwraca dane

2.11.2 Google Cloud Console

1. Utwórz OAuth consent screen
2. Dodaj zakresy: `openid, email, profile`
3. Utwórz OAuth Client ID
4. Zanotuj: Client ID, Client Secret, Redirect URI

2.11.3 Microsoft Entra ID

1. Azure Portal → App registrations
2. Zanotuj: Application ID, Tenant ID
3. Dodaj Redirect URIs
4. Włącz "ID tokens"

2.11.4 Integracja w . NET API

```
builder
.AddAuthentication(JwtBearer)
.AddJwtBearer(options =>
{
    options.Authority =
    "https://login.microsoftonline.com/{tenant}/v2.0";
    options. Audience = "{client-id}";
});

[Authorize]
[HttpGet]
public ActionResult<List<CityDto>> GetAll()
=> Ok(cities);
```

2.12 Paginacja i filtrowanie

2.12.1 Przykład

```
// GET /api/cities? page=1&pageSize=10&country=Poland
[HttpGet]
public ActionResult<PagedResult<CityDto>> GetAll(
int page = 1,
int pageSize = 10,
string?  country = null)
{
    var query = cities.AsQueryable();

    if (!string.IsNullOrEmpty(country))
```

```
        query = query.Where(c => c.Country == country);

        var total = query.Count();
        var items = query
        .Skip((page - 1) * pageSize)
        . Take(pageSize)
        . ToList();

        return Ok(new PagedResult<CityDto>
        {
            Items = items,
            Total = total,
            Page = page,
            PageSize = pageSize
        });
    }
```

2.13 Walidacja danych

2.13.1 Data Annotations

```
    public class CityDto
    {
        [Required(ErrorMessage = "Imi  jest wymagane")]
        [MinLength(2)]
        public string Name { get; set; }

        [Required]
        public string Country { get; set; }
    }
```

2.13.2 Walidacja automatyczna

ASP.NET automatycznie waliduje na podstawie atrybutów i zwraca 400 Bad Request z szczegółami błędów.

2.14 Pytania kontrolne

1. Jakie są podstawowe metody HTTP i ich zastosowania?
2. Kiedy używać PUT, a kiedy PATCH?
3. Jaka jest różnica między 200 a 201 a 204?
4. Co to jest DTO i dlaczego go używamy?
5. Jak obsługiwać błędy w API?
6. Jakie są kroki OAuth 2.0 + PKCE flow?

7. Jak zaimplementować autoryzację w API?
8. Jak dodać paginację do endpointu?

Rozdział 3

Tworzenie aplikacji MVC i MVVM

3.1 Wstęp

MVC (Model-View-Controller) i MVVM (Model-View-ViewModel) to wzorce architektoniczne zapewniające separację warstw i ułatwiające testowanie oraz utrzymanie kodu.

3.2 Dlaczego architektura w . NET MAUI?

3.2.1 Problemy bez architektury

- Logika biznesowa mieszana z UI (code-behind)
- Brak możliwości testów jednostkowych
- Trudna rozbudowa i utrzymanie
- Duplikacja kodu

3.2.2 Korzyści

1. Oddzielenie warstw (UI, logika prezentacji, logika biznesowa)
2. ViewModel bez zależności od UI - łatwe testowanie
3. Czystszy kod, mniejsza duplikacja
4. Łatwiejsze utrzymanie i rozwój

3.3 MVC vs MVVM

3.3.1 MVC (ASP.NET Core)

Model Dane aplikacji, baza danych

View HTML/Razor, interfejs użytkownika

Controller Pośrednik, obsługuje żądania HTTP

Flow MVC

1. Użytkownik wysyła żądanie HTTP
2. Controller odbiera żądanie
3. Controller pobrania dane z Model
4. Controller zwraca View (HTML)

3.3.2 MVVM (WPF / MAUI)

Model Dane aplikacji

View XAML, interfejs użytkownika

ViewModel Stan + logika prezentacji + INotifyPropertyChanged

Flow MVVM

1. Użytkownik klika przycisk w View
2. Command w ViewModel obsługuje zdarzenie
3. ViewModel zmienia stan (właściwości)
4. Binding automatycznie aktualizuje View

3.4 Data Binding

3.4.1 Tryby bindingu

Tryb	Kierunek	Opis
OneWay	View ← ViewModel	Tylko zmiana danych w VM
TwoWay	View ViewModel	Bidirektjonalny przepływ
OneTime	View ← ViewModel	Binding tylko raz przy załadowaniu

3.4.2 Compiled Bindings

```
<!-- XAML -->
<ContentPage
xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
x:DataType="local:MainViewModel">

<Entry Text="{Binding FirstName, Mode=TwoWay}" />
<Label Text="{Binding FullName}" />
<Button Command="{Binding SaveCommand}" Text="Zapisz" />
</ContentPage>
```

3.4.3 Benefit x:DataType

- Lepsze IntelliSense
- Lepsza wydajność (compiled bindings)
- Walidacja w compile time

3.5 CommunityToolkit.Mvvm

3.5.1 Instalacja

```
dotnet add package CommunityToolkit.Mvvm
```

3.5.2 ObservableObject

Klasa bazowa dla ViewModel z wbudowanym INotifyPropertyChanged.

3.5.3 ObservableProperty

```
public partial class MainViewModel : ObservableObject
{
    [ObservableProperty]
    private string firstName;

    [ObservableProperty]
    private string lastName;

    public string FullName => $"{FirstName} {LastName}";
}
```

3.5.4 Source Generators

Toolkit automatycznie generuje:

- PropertyChanged event
- Pełne implementacje właściwości
- RelayCommand handlers

3.6 Commands vs Event Handlers

3.6.1 RelayCommand

```
public partial class MainViewModel : ObservableObject
{
    [RelayCommand]
    private async Task Save()
    {
        IsBusy = true;
        try
        {
            await Task.Delay(1000); // Symulacja
        }
        finally
        {
            IsBusy = false;
        }
    }
}

// W XAML
<Button Command="{Binding SaveCommand}" Text="Zapisz" />
```

3.6.2 CanExecute

```
public partial class MainViewModel : ObservableObject
{
    [ObservableProperty]
    private bool isBusy;

    [RelayCommand(CanExecute = nameof(IsNotBusy))]
    private async Task Save()
    {
        // ...
    }

    private bool IsNotBusy => !IsBusy;
}
```

3.6.3 ICommand vs Event Handlers

ICommand	**Event Handlers**
Testowalne	Trudne do testowania
CanExecute (enable/disable)	Brak warunkowego disablowania
Data binding	Require code-behind
MVVM friendly	Tight coupling

3.7 Model vs DTO vs Entity

3.7.1 Entity

Klasa reprezentująca wiersz w bazie danych, zawiera ID.

3.7.2 DTO

Obiekt transferu danych między warstwami, odseparowany od bazy.

3.7.3 Model

Obiekt używany w aplikacji, może być kombinacją Entity i DTO.

3.7.4 Przykład

```
// Entity
public class User
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string PasswordHash { get; set; }
}

// DTO
public class UserDto
{
    public int Id { get; set; }
    public string Email { get; set; }
}

// ViewModel
public class UserViewModel : ObservableObject
{
    [ObservableProperty]
    private string email;

    public void LoadFrom(UserDto dto)
    {
        Email = dto.Email;
    }
}
```

3.8 Walidacja

3.8.1 ValidationAttributes

```
public class ContactViewModel : ObservableObject
{
    [ObservableProperty]
    [Required(ErrorMessage = "Imi  jest wymagane")]
    [MinLength(2, ErrorMessage = "Min 2 znaki")]
    private string firstName;

    [ObservableProperty]
    private string validationMessage;

    [RelayCommand]
    private void Save()
    {
        var context = new ValidationContext(this);
        var results = new List<ValidationResult>();

        if (! Validator.TryValidateObject(this, context,
        results, true))
        {
            ValidationMessage = string.Join("\n",
            results.Select(r => r.ErrorMessage));
        }
    }
}
```

3.9 ValueConverter

3.9.1 Przykład - BoolInverseConverter

```
public class BoolInverseConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
    object parameter, CultureInfo culture)
    {
        return !(bool)value;
    }

    public object ConvertBack(object value, Type targetType,
    object parameter, CultureInfo culture)
    {
        return !(bool)value;
    }
}

// ResourceDictionary
<ResourceDictionary
xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
xmlns:local="clr-namespace:MyApp">
<local:BoolInverseConverter x:Key="BoolInverse" />
```

```
    </ResourceDictionary>

    // XAML
    <Button Text="Zapisz"
    IsEnabled="{Binding IsBusy,
        Converter={StaticResource BoolInverse}}" />
```

3.10 Dependency Injection

3.10.1 MAUI

```
    // MauiProgram.cs
    var builder = MauiApp.CreateBuilder();
    builder
    .UseMauiApp<App>()
    .ConfigureFonts(fonts =>
    {
        fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
    })
    .ConfigureServices(services =>
    {
        services.AddSingleton<MainViewModel>();
        services.AddTransient<MainPage>();
    });

    // MainPage.xaml. cs
    public MainPage(MainViewModel vm)
    {
        InitializeComponent();
        BindingContext = vm;
    }
```

3.11 Testy jednostkowe

3.11.1 Projekt testowy

```
    dotnet new nunit -n DemoTests
    dotnet add DemoTests/DemoTests.csproj reference \
    DemoMaui/DemoMaui. csproj
```

3.11.2 Przykładowy test

```
    [TestFixture]
    public class MainViewModelTests
    {
        [Test]
```

```
        public void FullName_Updates_OnFirstOrLastChange()
        {
            var vm = new MainViewModel();
            vm.FirstName = "Anna";
            vm.LastName = "Nowak";

            Assert. That(vm.FullName, Is.EqualTo("Anna Nowak"));
        }

        [Test]
        public async Task Save_SetsBusyTrue()
        {
            var vm = new MainViewModel();

            Assert.That(vm.IsBusy, Is.False);

            var task = vm.SaveCommand.ExecuteAsync(null);

            Assert.That(vm.IsBusy, Is.True);

            await task;

            Assert.That(vm.IsBusy, Is.False);
        }
    }
```

3.12 Nawigacja

3.12.1 MAUI Shell

```
    // AppShell.xaml. cs
    public partial class AppShell : Shell
    {
        public AppShell()
        {
            InitializeComponent();
            Routing.RegisterRoute("details", typeof(DetailsPage));
        }
    }

    // MainViewModel.cs
    [RelayCommand]
    private async Task SelectContact(Contact contact)
    {
        await Shell.Current.GoToAsync(
        $"details?id={contact.Id}");
    }

    // DetailsPage.xaml. cs
```

```
[QueryProperty(nameof(ContactId), "id")]
public partial class DetailsPage : ContentPage
{
    private int contactId;
    public int ContactId
    {
        get => contactId;
        set => contactId = value;
    }
}
```

3.13 Stylizacja

3.13.1 ResourceDictionary

```
<!-- Resources/Styles. xaml -->
<ResourceDictionary
xmlns="http://schemas.microsoft.com/dotnet/2021/maui">

<Color x:Key="PrimaryColor">#512BD4</Color>
<Color x:Key="SecondaryColor">#DFD8F7</Color>

<Style TargetType="Button" ApplyToDerivedTypes="True">
<Setter Property="BackgroundColor"
Value="{StaticResource PrimaryColor}" />
<Setter Property="Padding" Value="10,5" />
<Setter Property="FontSize" Value="14" />
</Style>
</ResourceDictionary>
```

3.13.2 Light/Dark Mode

```
<Label Text="Hello"
TextColor="{AppThemeBinding Light=Black,
    Dark=White}" />
```

3.14 MAUI vs WPF

3.14.1 Porównanie

Aspekt	**MAUI**	**WPF**
Platformy	Android, iOS, Windows, macOS	Windows
Projekt	Single Project	Classic Desktop
Handlers	Handler architecture	Dependency Properties
Hot Reload	Wieloplatformowy	Desktopowy

3.14.2 Lifecycle MAUI

- **OnStart**: Aplikacja uruchomiona
- **OnSleep**: Aplikacja przechodzi w tło
- **OnResume**: Aplikacja wznowiona z tła

3.14.3 Uprawnienia na mobilnych

```
// AndroidManifest.xml
<uses-permission android:name="android.permission. CAMERA" />

// Kod
var status = await Permissions.CheckStatusAsync<Permissions.
   Camera>();
if (status != PermissionStatus.Granted)
{
    status = await Permissions.RequestAsync<Permissions.Camera
       >();
}
```

3.15 Pytania kontrolne

1. Jaka jest różnica między MVC a MVVM?
2. Co to jest Data Binding i jakie są jego tryby?
3. Jak działa ObservableProperty w MVVM Toolkit?
4. Dlaczego RelayCommand jest lepszy niż event handler?
5. Co to jest DTO i dlaczego oddzielamy go od Entity?
6. Jak zaimplementować walidację w MVVM?
7. Jakie są korzyści z Dependency Injection?
8. Jakie są główne różnice między MAUI a WPF?
9. Jak obsługiwać uprawnienia na urządzeniach mobilnych?

Rozdział 4

Tworzenie aplikacji Blazor

4.1 Wstęp

Blazor to framework do tworzenia aplikacji webowych w .NET. Umożliwia tworzenie interaktywnych UI przy użyciu C zamiast JavaScript.

4.2 Dlaczego Blazor?

4.2.1 Korzyści

- **Jeden zablon**: Obsługuje wiele trybów renderowania
- **SEO + szybkość**: SSR dla pierwszego ładowania
- **Wspólny kod**: Model, walidacja, logika w . NET (brak duplikacji JS/TS)
- **Progressive Enhancement**: Stopniowe dodawanie interaktywności

4.3 Tryby renderowania

4.3.1 Static SSR (Server-Side Rendering)

Opis Strona jest renderowana na serwerze, brak interaktywności

Użycie Content, marketing, blog, landing page

Zalety Minimalny rozmiar, SEO, szybkie ładowanie

Wady Brak interaktywności

```
// Program.cs - nie dodajemy interactive
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorComponents();

var app = builder.Build();
app.MapRazorComponents<App>();
app.Run();
```

4.3.2 Interactive Server

Opis Logika po stronie serwera, komunikacja przez WebSocket

Użycie Backoffice, aplikacje z małą ilością użytkowników

Zalety Mały transfer, szybkie TTFB, pełna moc serwera

Wady Połączenia stałe, ograniczona skalowalność

```
// Component.razor
@rendermode InteractiveServer

@page "/counter"

<h1>Counter</h1>
<p>Current count: @count</p>
<button @onclick="IncrementCount">Click me</button>

@code {
    private int count = 0;

    private void IncrementCount()
    {
        count++;
    }
}
```

4.3.3 Interactive WebAssembly

Opis Logika w przeglądarce, brak zależności od serwera

Użycie Aplikacje wymagające offline, PWA

Zalety Skalowalność, offline, PWA, brak roundtrip

Wady Większy initial payload, cold start

```
// Program.cs - dodajemy WebAssembly
builder.Services.AddRazorComponents()
.AddInteractiveWebAssemblyComponents();
```

4.3.4 Interactive Auto

Opis Pierwszy raz Server (szybki), potem przełącza na WebAssembly

Użycie Aplikacje wymagające szybkiego TTFB i skalowalności

Zalety Najlepsze z obu światów

Wady Większa złożoność

```
@rendermode InteractiveAuto
```

4.4 Stream Rendering

4.4.1 Problem bez stream renderingu

Użytkownik widzi białą stronę, aż wszystkie dane będą pobrane.

4.4.2 Stream rendering - rozwiązanie

1. Szkielet strony zaraz
2. Placeholder "Ładowanie..."
3. Dane wypływają stopniowo

4.4.3 Implementacja

```
@page "/products"
@rendermode InteractiveServer

<h1>Products</h1>

@if (products == null)
{
    <p>  adowanie  ...</p>
}
else
{
    @foreach (var product in products)
    {
        <p>@product.Name</p>
    }
}

@code {
    private List<Product> products;

    protected override async Task OnInitializedAsync()
    {
        await Task.Delay(1500); // Symulacja
        products = new List<Product>
        {
            new Product { Name = "Product 1" },
            new Product { Name = "Product 2" }
        };
    }
}
```

4.5 Struktura projektu Blazor Web App

4.5.1 Komponenty

Server Host, API, SSR

Client WebAssembly (komponenty interaktywne)

Shared Modele, DTO, interfejsy usług

4.5.2 Program. cs - Server

```
var builder = WebApplicationBuilder.CreateBuilder(args);

builder.Services
.AddRazorComponents()
.AddInteractiveServerComponents()
.AddInteractiveWebAssemblyComponents();

builder.Services.AddScoped<GameService>();

var app = builder.Build();

app.UseStaticFiles();
app.UseRouting();
app.MapRazorComponents<App>()
.AddInteractiveServerRenderMode()
.AddInteractiveWebAssemblyRenderMode();

app.Run();
```

4.6 Formularze

4.6.1 EditForm

```
@page "/contacts/edit"
@rendermode InteractiveServer

<h1>Nowy kontakt</h1>

<EditForm Model="contact" OnValidSubmit="HandleSubmit">
<DataAnnotationsValidator />
<ValidationSummary />

<div>
<label>Imi :</label>
<InputText @bind-Value="contact.FirstName" />
</div>
```

```
    <div>
    <label>Email:</label>
    <InputEmail @bind-Value="contact.Email" />
    </div>

    <button type="submit">Zapisz</button>
    </EditForm>

    @code {
        private Contact contact = new();

        private async Task HandleSubmit()
        {
            await contactService.SaveAsync(contact);
        }
    }
```

4.6.2 SSR static formularze

```
    @page "/contact"

    <h1>Kontakt</h1>

    <form method="post">
    <input type="hidden" name="FormName" value="Contact" />

    <div>
    <label>Imi :</label>
    <input type="text" name="firstName" required />
    </div>

    <button type="submit">Wy lij </button>
    </form>

    @code {
        [SupplyParameterFromForm(FormName = "Contact")]
        private string FirstName { get; set; }

        public async Task OnPostAsync()
        {
            if (FirstName != null)
            {
                // Przetw rz dane
            }
        }
    }
```

4.7 Walidacja

4.7.1 Data Annotations

```
public class Contact
{
    [Required(ErrorMessage = "Imi  jest wymagane")]
    [MinLength(2, ErrorMessage = "Min 2 znaki")]
    public string FirstName { get; set; }

    [Required]
    [EmailAddress]
    public string Email { get; set; }
}
```

4.7.2 Custom validator

```
public class CustomValidator : ComponentBase
{
    private EditContext editContext;

    [CascadingParameter]
    private EditContext EditContext
    {
        get => editContext;
        set
        {
            editContext = value;
            editContext?.AddAsyncValidator(this);
        }
    }

    public async Task ValidateAsync(
    ValidationContext context)
    {
        // Custom validacja
    }
}
```

4.8 EF Core w Blazor

4.8.1 DbContext

```
public class GameContext : DbContext
{
    public DbSet<Game> Games { get; set; }

    protected override void OnConfiguring(
```

```
        DbContextOptionsBuilder options)
        {
            options. UseSqlServer(
            "Server=localhost;Database=Games;.. .");
        }

        protected override void OnModelCreating(
        ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Game>(). HasData(
            new Game { Id = 1, Title = "Elden Ring" },
            new Game { Id = 2, Title = "Baldur's Gate 3" }
            );
        }
    }
```

4.8.2 Migracje

```
    dotnet ef migrations add InitialCreate
    dotnet ef database update
```

4.8.3 Rejestracja w DI

```
    builder.Services.AddDbContext<GameContext>(options =>
    options.UseSqlServer(
    builder.Configuration.GetConnectionString("Default")));
```

4.9 Service Layer

4.9.1 Server - bezpośredni dostęp do DbContext

```
    public interface IGameService
    {
        Task<List<Game>> GetAllAsync();
        Task<Game> GetByIdAsync(int id);
        Task CreateAsync(Game game);
    }

    public class GameService : IGameService
    {
        private readonly GameContext context;

        public GameService(GameContext context)
        {
            this.context = context;
        }
```

```
        public async Task<List<Game>> GetAllAsync()
        {
            return await context.Games.ToListAsync();
        }

        public async Task CreateAsync(Game game)
        {
            context.Games.Add(game);
            await context.SaveChangesAsync();
        }
    }
```

4.9.2 Client - HttpClient

```
    public class ClientGameService : IGameService
    {
        private readonly HttpClient httpClient;

        public ClientGameService(HttpClient httpClient)
        {
            this.httpClient = httpClient;
        }

        public async Task<List<Game>> GetAllAsync()
        {
            return await httpClient.GetFromJsonAsync<List<Game>>(
            "/api/games");
        }

        public async Task CreateAsync(Game game)
        {
            await httpClient.PostAsJsonAsync("/api/games", game);
        }
    }
```

4.10 Komponenty CRUD

4.10.1 Lista z stream renderingiem

```
    @page "/games"
    @rendermode InteractiveServer
    @inject IGameService gameService

    <h1>Games</h1>

    <button @onclick="NavigateToCreate">Add New</button>

    @if (games == null)
```

```
    {
        <p>Loading...</p>
    }
    else
    {
        <table>
        <thead>
        <tr>
        <th>Title</th>
        <th>Actions</th>
        </tr>
        </thead>
        <tbody>
        @foreach (var game in games)
        {
            <tr>
            <td>@game.Title</td>
            <td>
            <button @onclick='() => NavigateToEdit(game. Id)'>
            Edit
            </button>
            <button @onclick='() => Delete(game.Id)'>
            Delete
            </button>
            </td>
            </tr>
        }
        </tbody>
        </table>
    }

    @code {
        private List<Game> games;

        protected override async Task OnInitializedAsync()
        {
            games = await gameService.GetAllAsync();
        }

        private void NavigateToCreate()
        {
            // Navigate to create page
        }

        private async Task Delete(int id)
        {
            await gameService.DeleteAsync(id);
            games = await gameService.GetAllAsync();
        }
    }
```

4.10.2 Edycja - Auto render mode

```
@page "/games/edit"
@page "/games/edit/{id:int}"
@rendermode InteractiveAuto
@inject IGameService gameService

<h1>@(game.Id == 0 ? "Add" : "Edit") Game</h1>

<EditForm Model="game" OnValidSubmit="Save">
<DataAnnotationsValidator />
<ValidationSummary />

<InputText @bind-Value="game.Title"
placeholder="Title" />

<button type="submit">Save</button>
</EditForm>

@code {
    [Parameter]
    public int Id { get; set; }

    private Game game = new();

    protected override async Task OnInitializedAsync()
    {
        if (Id > 0)
        {
            game = await gameService.GetByIdAsync(Id);
        }
    }

    private async Task Save()
    {
        if (game.Id == 0)
        await gameService.CreateAsync(game);
        else
        await gameService.UpdateAsync(game);
    }
}
```

4.11 PWA (WebAssembly)

4.11.1 Service Worker

- Cache zasobów offline
- Synchronizacja w tle
- Push notyfikacje

4.11.2 Włączenie PWA

Blazor WebAssembly automatycznie obsługuje PWA. Dodaj manifest i service worker.

4.12 Pre-rendering

4.12.1 Włączone pre-rendering

```
@rendermode new InteractiveServerRenderMode(prerender: true)
```

4.12.2 Wyłączone pre-rendering

```
@rendermode new InteractiveServerRenderMode(prerender: false)
```

4.13 Pytania kontrolne

1. Jakie są główne tryby renderowania w Blazor?
2. Kiedy wybrać Static SSR, a kiedy Interactive?
3. Co to jest stream rendering i jakie są jego korzyści?
4. Jak zaimplementować formularz w Blazor?
5. Jakie są różnice między EditForm a HTML form?
6. Jak zintegrować EF Core z Blazor?
7. Co to jest PWA i kiedy go używać?
8. Jaka jest różnica między pre-rendering włączonym a wyłączonym?
9. Jak obsługiwać walidację w formularzach?

Rozdział 5

Tworzenie aplikacji rozproszonych - . NET Aspire

5.1 Wstęp

. NET Aspire to framework do tworzenia i zarządzania aplikacjami rozproszonymi. Upraszcza orkestrację wielu usług, obsługuje monitoring i deployment.

5.2 Problemy przy wielu usługach

5.2.1 Bez Aspire

- Ręczne uruchamianie wielu projektów
- Różne porty i konfiguracje u różnych developerów
- Brak spójnej obserwowalności
- Ręczne tworzenie connection stringów
- Ręczne uruchamianie baz danych (Docker)
- Różne konfiguracje DEV vs PROD

5.2.2 Z Aspire

- Deklaratywny opis zasobów (AppHost)
- Service Discovery (identyfikatory zamiast URL)
- Integracje (Redis, PostgreSQL, Service Bus)
- Automatyczne health checks i telemetria
- Resilience (retry, timeout, circuit breaker)
- Dashboard do monitorowania
- Wdrożenie na Azure Container Apps lub Kubernetes

5.3 Kluczowe pojęcia

5.3.1 AppHost

Projekt, który definiuje wszystkie zasoby i ich relacje.

5.3.2 Service Defaults

Wspólne rozszerzenia dla wszystkich usług.

5.3.3 Service Discovery

Automatyczne odkrywanie usług bez podawania portów.

5.4 Health Checks

5.4.1 /health

Zwraca zdrowie usługi razem z zależnościami (baza, cache).

5.4.2 /alive

Zwraca tylko, czy proces żyje.

5.5 Telemetria

5.5.1 Logs

Ustrukturyzowane logi zapisywane w Dashboard.

5.5.2 Distributed Traces

Śledzenie żądania przez wiele usług.

5.5.3 Metrics

HTTP metrics, GC stats, custom metrics.

5.6 Resilience - Polly

5.6.1 Domyślnie wbudowane

- Retry z jitterem
- Timeout
- Circuit breaker

5.7 Komponenty

5.7.1 Redis

Pamięć podręczna dla rozproszonego cache'a.

5.7.2 PostgreSQL

Baza danych dla aplikacji rozproszonej.

5.7.3 Service Bus

Komunikacja między usługami.

5.8 Pytania kontrolne

1. Jakie są główne problemy przy zarządzaniu wieloma usługami?
2. Co to jest AppHost w . NET Aspire?
3. Jak Service Discovery ułatwia zarządzanie usługami?
4. Jakie są różnice między /health a /alive?
5. Jakie metryki zbiera Telemetria?
6. Co to jest Polly i jakie strategie oferuje?
7. Jak integrować Redis z . NET Aspire?
8. Jak monitorować aplikacje rozproszone?