Notatki do kolokwium – Programowanie aplikacji desktop i mobilnych (.NET, MVC, MVVM, API, Copilot)

Spis treści

1 Programowanie AI i GitHub Copilot w Visual Studio Code **4**
- 1.1 Wprowadzenie . . . 4
- 1.2 Tryb Agent Mode . . . 4
- 1.3 Różnice między klasycznym Copilotem a Agent Mode . . . 4
- 1.4 Bezpieczeństwo: Allow i Deny List . . . 4
- 1.5 Limity zapytań i kontrola wydajności . . . 4
- 1.6 Copilot Instructions i konwencje kodowania . . . 5
- 1.7 Tryby czatu i personalizacja . . . 5
- 1.8 Snooze Completions – wstrzymanie podpowiedzi . . . 5
- 1.9 Coding Agents i GitHub Actions . . . 5
- 1.10 Statystyki użycia i analiza produktywności . . . 5
- 1.11 Diagram działania agenta Copilot . . . 6
- 1.12 MCP (Model Context Protocol) Auto Start . . . 6
- 1.13 Background Agents i sesje w tle . . . 6
- 1.14 Task Lists (To-Do w czacie) . . . 7
- 1.15 Custom Chat Modes – Beast Mode . . . 7
- 1.16 Podsumowanie rozdziału . . . 8
- 1.17 Pytania kontrolne . . . 8

2 Tworzenie i testowanie API REST w .NET **8**
- 2.1 Wprowadzenie do API REST . . . 8
- 2.2 Tworzenie projektu Web API . . . 8
- 2.3 Certyfikaty HTTPS i zaufanie lokalne . . . 8
- 2.4 Testowanie API . . . 9
- 2.5 Czasowniki HTTP i ich zastosowanie . . . 9
- 2.6 Tworzenie modelu danych i DTO . . . 9
- 2.7 Implementacja kontrolera . . . 9
- 2.8 Kody odpowiedzi HTTP . . . 11
- 2.9 Generowanie klienta z OpenAPI / Swagger . . . 11
- 2.10 Autentykacja i autoryzacja . . . 11
- 2.11 Walidacja, idempotencja i paginacja . . . 11
- 2.12 Diagram przepływu zapytania REST API . . . 12
- 2.13 HEAD i OPTIONS - dodatkowe metody HTTP . . . 12
- 2.14 OAuth 2.0 i OpenID Connect – szczegółowy flow . . . 12
  - 2.14.1 Authorization Code + PKCE Flow . . . 12
  - 2.14.2 Konfiguracja providerów . . . 13
  - 2.14.3 Integracja w .NET API . . . 14
- 2.15 Paginacja i filtrowanie . . . 14
- 2.16 Idempotencja metod HTTP . . . 15

2.17 HATEOAS (Hypermedia As The Engine Of Application State) 15
2.18 Podsumowanie rozdziału 16
2.19 Pytania kontrolne 16

3 Architektury MVC i MVVM w aplikacjach .NET 16
3.1 Wprowadzenie 16
3.2 Wzorzec MVC – Model, View, Controller 16
3.3 Wzorzec MVVM – Model, View, ViewModel 17
3.4 Diagram przepływu MVVM 18
3.5 Porównanie MVC i MVVM 18
3.6 Binding i konwertery danych 19
3.7 Walidacja danych 19
3.8 Zalety MVVM 19
3.9 Wady MVVM 19
3.10 Podsumowanie rozdziału 19
3.11 Pytania kontrolne 20

4 Aplikacje WPF, MAUI i Dependency Injection w .NET 20
4.1 Wprowadzenie 20
4.2 Struktura projektu WPF 20
4.3 Struktura projektu MAUI 20
4.4 Dependency Injection (DI) 21
4.5 Zalety DI 21
4.6 Cykl życia aplikacji MAUI 22
4.7 Diagram cyklu życia aplikacji MAUI 22
4.8 Tworzenie usług i integracja z API 22
4.9 Powiązanie z interfejsem w MAUI 23
4.10 MAUI Shell Navigation 24
4.10.1 Rejestracja tras 24
4.10.2 Nawigacja z parametrami 24
4.10.3 Odbieranie parametrów 24
4.10.4 Shell Flyout i TabBar 25
4.11 MAUI Essentials – dostęp do funkcji platformy 25
4.11.1 Connectivity – sprawdzanie połączenia 25
4.11.2 Geolocation – lokalizacja 26
4.11.3 DeviceInfo – informacje o urządzeniu 26
4.12 Kompilacja warunkowa – kod specyficzny dla platformy 26
4.13 Różnice WPF vs MAUI – rozszerzone 27
4.14 Podsumowanie rozdziału 27
4.15 Pytania kontrolne 27

5 Testy jednostkowe, typowe błędy i ściąga z komend .NET 28
5.1 Testy jednostkowe w .NET 28
5.1.1 Tworzenie projektu testowego 28
5.1.2 Przykład prostego testu 28
5.1.3 Testy kontrolerów API 28
5.2 Testy z wykorzystaniem DI i Mocków 29
5.3 Testy UI i integracyjne 29
5.4 Typowe błędy na kolokwium 29
5.5 Ściąga z komend .NET CLI 29
5.6 Skróty klawiszowe w Visual Studio Code 30
5.7 Pytania kontrolne 30

5.8 Podsumowanie końcowe 30

1 Programowanie AI i GitHub Copilot w Visual Studio Code

1.1 Wprowadzenie

GitHub Copilot to asystent programisty wykorzystujący sztuczną inteligencję (AI) do generowania kodu, sugerowania rozwiązań, tworzenia komentarzy i automatyzowania pracy w środowisku Visual Studio Code (VS Code). Opiera się na dużych modelach językowych (LLM) wytrenowanych na kodzie źródłowym z GitHuba i publicznych repozytoriów.

1.2 Tryb Agent Mode

W trybie **Agent Mode** Copilot działa jak inteligentny współprogramista. Użytkownik może wydawać mu polecenia w języku naturalnym, a agent samodzielnie wykonuje operacje, np.:

- generowanie plików i klas,
- kompilowanie i uruchamianie projektu,
- tworzenie testów jednostkowych,
- wykonywanie zadań administracyjnych (np. Git, CI/CD),
- analizowanie błędów i sugerowanie poprawek.

Agenty mogą pracować w tle, komunikować się przez *Copilot Chat*, a także wykonywać proste zadania autonomicznie, np. „stwórz endpoint API dla klasy `Product`".

1.3 Różnice między klasycznym Copilotem a Agent Mode

- **Klasyczny Copilot** – podpowiada linie lub bloki kodu na podstawie kontekstu w edytorze.
- **Agent Mode** – działa jak interaktywny chatbot, który rozumie intencje użytkownika, może wywoływać komendy, analizować pliki projektu i działać w wielu krokach.

1.4 Bezpieczeństwo: Allow i Deny List

Aby uniknąć ryzyka wykonania niepożądanych poleceń systemowych, Agent Mode korzysta z list:

- **Allow List** – lista komend, które agent może wykonać (np. `dotnet build`, `dotnet test`).
- **Deny List** – komendy, które są blokowane (np. `rm -rf`, `sudo`).

Administrator lub użytkownik może te listy konfigurować lokalnie w ustawieniach VS Code.

1.5 Limity zapytań i kontrola wydajności

W ustawieniach Copilota można ustawić:

- **Max Requests** – maksymalną liczbę równoległych zapytań do modelu,
- **Response Timeout** – czas oczekiwania na odpowiedź.

W projektach zespołowych pozwala to uniknąć spowolnień i nadmiernego zużycia zasobów API.

1.6 Copilot Instructions i konwencje kodowania

Copilot może generować kod zgodny z przyjętymi standardami:

- `PascalCase` dla klas i metod,
- `camelCase` dla zmiennych,
- komentarze XML i dokumentacja metod.

Można też dopisać własne instrukcje w pliku `.copilot.json`, np.:

Listing 1: Przykład konfiguracji Copilota w JSON

```
{
  "style": "use PascalCase for classes, camelCase for variables",
  "language": "C#",
  "comment_format": "XML"
}
```

1.7 Tryby czatu i personalizacja

Copilot pozwala przełączać tzw. **Chat Modes**, które zmieniają styl i ton pracy agenta:

- `Default Mode` – standardowy styl pomocy,
- `Beast Mode` – agresywne sugestie optymalizacji,
- `Explain Mode` – tłumaczenie działania kodu linia po linii.

1.8 Snooze Completions – wstrzymanie podpowiedzi

Użytkownik może czasowo wyłączyć podpowiedzi AI, jeśli pracuje nad kodem manualnie lub nie chce rozproszeń. Przykład: „Snooze for 10 minutes" – wstrzymuje automatyczne uzupełnianie przez 10 minut.

1.9 Coding Agents i GitHub Actions

Copilot integruje się z GitHub Actions, co pozwala automatyzować procesy CI/CD:

- testowanie i budowanie projektu po zatwierdzeniu PR,
- analiza błędów i propozycje poprawek w pipeline,
- automatyczne generowanie dokumentacji i changelogów.

Agent może np. po wykonaniu komendy `commit` zaproponować opis PR i wygenerować testy do nowych metod.

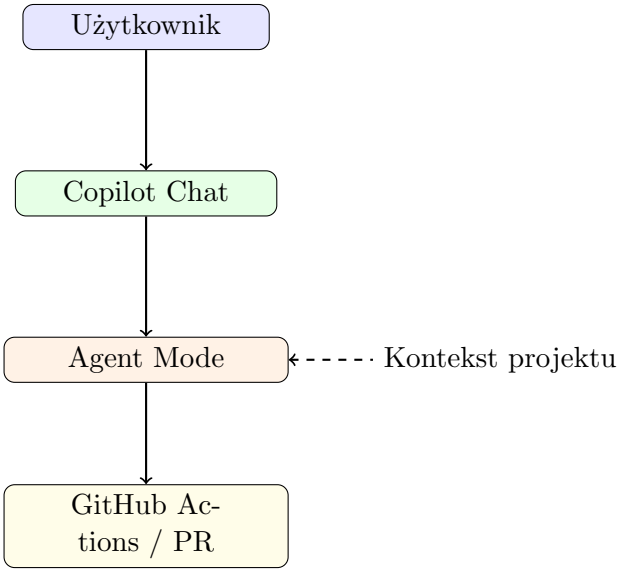
1.10 Statystyki użycia i analiza produktywności

Copilot rejestruje dane o efektywności (lokalnie, bez wysyłania kodu):

- liczba akceptowanych sugestii,
- średnia długość wygenerowanego kodu,
- czas reakcji modelu.

Z raportów wynika, że średnio 46–55% kodu w projektach C# i TypeScript generowanych z pomocą Copilota jest akceptowane przez programistów.

1.11 Diagram działania agenta Copilot



1.12 MCP (Model Context Protocol) Auto Start

MCP to protokół rozszerzający możliwości Copilota o dodatkowe źródła kontekstu i narzędzia.

- **Rola MCP** – dostarcza Copilotowi dodatkowe informacje z zewnętrznych źródeł (bazy danych, API, dokumentacja),
- **Konfiguracja** – Auto Start mode: `always`, `ask`, `never`,
- **Serwery MCP** – rozszerzenia dostarczające kontekst (np. dostęp do lokalnej dokumentacji projektu),
- **Zarządzanie** – Extensions → MCP servers w VS Code.

Przykład użycia:

- MCP może automatycznie dostarczać Copilotowi aktualną dokumentację API projektu,
- Umożliwia dostęp do niestandardowych narzędzi deweloperskich (linters, formattery),
- Pozwala na integrację z bazą wiedzy zespołu.

1.13 Background Agents i sesje w tle

Background Agents pozwalają na delegowanie długotrwałych zadań do wykonania w tle, podczas gdy programista kontynuuje pracę.

- **Agent Sessions (UI integration)** – panel w VS Code pokazujący aktywne sesje agentów,
- **Przypisywanie zadań** – można utworzyć issue w GitHub i przypisać je do Copilota,
- **Automatyczne PR** – agent tworzy pull request z rozwiązaniem w tle,
- **Kontynuacja pracy** – programista może pracować lokalnie, podczas gdy agent działa równolegle.

Przykładowy workflow:

1. Programista tworzy issue: "Dodać skeleton loading na stronie logowania",
2. Przypisuje issue do Copilota,
3. Agent tworzy nowy branch, implementuje rozwiązanie,
4. Tworzy pull request do review,
5. Programista przegląda i akceptuje/modyfikuje.

1.14 Task Lists (To-Do w czacie)

Copilot może generować listy kroków (task lists) dla złożonych zadań i śledzić ich realizację.

Przykład:

Listing 2: Polecenie w Copilot Chat

```
U ytkownik: "Dodaj stron  z formularzem kontaktowym"

Copilot generuje:
[ ] Utworzy  nowy plik ContactPage.xaml
[ ] Doda  pola: Imi , Email, Wiadomo
[ ] Utworzy  ViewModel z walidacj
[ ] Doda  endpoint API POST /api/contact
[ ] Doda  nawigacj  do strony kontaktu
```

Podczas pracy Copilot automatycznie oznacza wykonane kroki:

x Utworzyć nowy plik ContactPage.xaml

x Dodać pola: Imię, Email, Wiadomość

Utworzyć ViewModel z walidacją (w trakcie)

1.15 Custom Chat Modes – Beast Mode

Beast Mode to przykład niestandardowego trybu czatu, który agresywnie optymalizuje kod.

Konfiguracja w projekcie:

1. Utwórz katalog `.github/chat-modes/`
2. Utwórz plik `beast.json`:

Listing 3: Przykład konfiguracji Beast Mode

```
{
        "name": "Beast Mode",
        "description": "Agresywna optymalizacja kodu",
        "instructions": [
        "Zawsze sugeruj najbardziej wydajne rozwi zanie",
        "U ywaj async/await dla operacji I/O",
        "Optymalizuj pod k tem pami ci i procesora",
        "Sugeruj LINQ zamiast p tli where mo liwe"
        ]
}
```

Użycie: W Copilot Chat wybierz tryb "Beast Mode" i poproś o optymalizację kodu.

1.16 Podsumowanie rozdziału

- Copilot umożliwia pracę w trybie interaktywnym i autonomicznym.
- Agent Mode pozwala wykonywać operacje w środowisku VS Code.
- System zapewnia bezpieczeństwo dzięki Allow/Deny List.
- Integracja z GitHub Actions umożliwia automatyzację CI/CD.
- Wydajność można kontrolować przez limity zapytań.

1.17 Pytania kontrolne

1. Czym różni się klasyczny Copilot od trybu Agent Mode?
2. Jak działa lista Allow/Deny i dlaczego jest istotna?
3. Jakie są przykłady integracji Copilota z GitHub Actions?
4. Jak można ograniczyć liczbę zapytań do modelu?
5. W jaki sposób Copilot wspiera standardy nazewnictwa w C#?

2 Tworzenie i testowanie API REST w .NET

2.1 Wprowadzenie do API REST

REST (Representational State Transfer) to architektura, która definiuje zasady komunikacji między klientem a serwerem przy użyciu protokołu HTTP. W .NET API REST jest zwykle budowane przy użyciu `ASP.NET Core Web API`, co umożliwia szybkie tworzenie usług sieciowych, które komunikują się w formacie JSON.

2.2 Tworzenie projektu Web API

Nowy projekt można utworzyć poleceniem:

Listing 4: Tworzenie projektu Web API w .NET

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

Po uruchomieniu aplikacji interfejs testowy Swagger jest dostępny pod adresem:

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

2.3 Certyfikaty HTTPS i zaufanie lokalne

Domyślnie .NET generuje certyfikat deweloperski HTTPS. Jeżeli przeglądarka zgłasza problem z zaufaniem, można go zarejestrować:

Listing 5: Rejestracja certyfikatu deweloperskiego

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

2.4 Testowanie API

Istnieje wiele sposobów testowania API:

- **Swagger** – interaktywny interfejs do testów,
- **curl** – narzędzie wiersza poleceń,
- **Postman** – aplikacja GUI do testów HTTP,
- **Pliki .http** – zintegrowane z Visual Studio Code,
- **Przeglądarka** – do zapytań GET.

Listing 6: Przykładowe zapytanie curl

```
curl –X GET https://localhost:7294/api/cities
```

2.5 Czasowniki HTTP i ich zastosowanie

Metoda	Opis	Przykład
GET	Pobranie danych	/api/cities
POST	Dodanie nowych danych	/api/cities
PUT	Nadpisanie danych	/api/cities/3
PATCH	Częściowa aktualizacja	/api/cities/3
DELETE	Usunięcie danych	/api/cities/3

2.6 Tworzenie modelu danych i DTO

Model reprezentuje dane domenowe, natomiast **DTO** (Data Transfer Object) służy do przesyłania danych między klientem a API.

Listing 7: Przykładowy model City

```
public class City
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public int Population { get; set; }
}
```

Listing 8: Przykładowy DTO CityDto

```
public class CityDto
{
    public string Name { get; set; } = string.Empty;
    public int Population { get; set; }
}
```

2.7 Implementacja kontrolera

Listing 9: Kontroler CitiesController

```
[ApiController]
[Route("api/[controller]")]
public class CitiesController : ControllerBase
{
```

```
    private static List<City> _cities = new()
    {
        new City { Id = 1, Name = "Warszawa", Population = 1790658 },
        new City { Id = 2, Name = "Krak w", Population = 800653 }
    };

    [HttpGet]
    public ActionResult<IEnumerable<CityDto>> GetCities()
        => Ok(_cities.Select(c => new CityDto
        {
            Name = c.Name,
            Population = c.Population
        }));

    [HttpGet("{id}")]
    public ActionResult<CityDto> GetCity(int id)
    {
        var city = _cities.FirstOrDefault(c => c.Id == id);
        if (city == null)
            return NotFound();
        return Ok(new CityDto { Name = city.Name, Population = city.Population }
    }

    [HttpPost]
    public ActionResult<CityDto> CreateCity(CityDto dto)
    {
        if (_cities.Any(c => c.Name == dto.Name))
            return Conflict("City already exists.");

        var city = new City
        {
            Id = _cities.Max(c => c.Id) + 1,
            Name = dto.Name,
            Population = dto.Population
        };
        _cities.Add(city);
        return CreatedAtAction(nameof(GetCity), new { id = city.Id }, dto);
    }
}
```

Znaczenie kodów statusu HTTP

c

Zakres kodów	Znaczenie	
1xx	Informacyjne (Informational)	Serwer przyjął żądanie i przetwarza j
2xx	Sukces (Success)	Żądanie zostało pomyślnie
3xx	Przekierowanie (Redirection)	Wymaga dodatkowego działania, np. przekierowani
4xx	Błąd po stronie klienta (Client Error)	Coś nie tak z zapytaniem użytkow
5xx	Błąd po stronie serwera (Server Error)	Serwer ma problem z przetworzeniem żądania (500

2.8 Kody odpowiedzi HTTP

Najczęściej używane kody odpowiedzi:

- 200 OK – żądanie wykonano poprawnie,
- 201 Created – obiekt został utworzony,
- 204 No Content – brak treści w odpowiedzi,
- 400 Bad Request – błędne żądanie,
- 401 Unauthorized – brak autoryzacji,
- 403 Forbidden – brak dostępu,
- 404 Not Found – zasób nie istnieje,
- 409 Conflict – konflikt danych,
- 500 Internal Server Error – błąd serwera.

2.9 Generowanie klienta z OpenAPI / Swagger

Swagger umożliwia wygenerowanie klienta API w wielu językach, np. przez narzędzie `NSwag`:

```
nswag openapi2csclient /input:https://localhost:7294/swagger/v1/swagger.json /ou
```

Taki klient może być wykorzystany w aplikacji konsolowej, WPF lub MAUI:

```
var client = new CitiesClient("https://localhost:7294", new HttpClient());
var cities = await client.GetCitiesAsync();
```

2.10 Autentykacja i autoryzacja

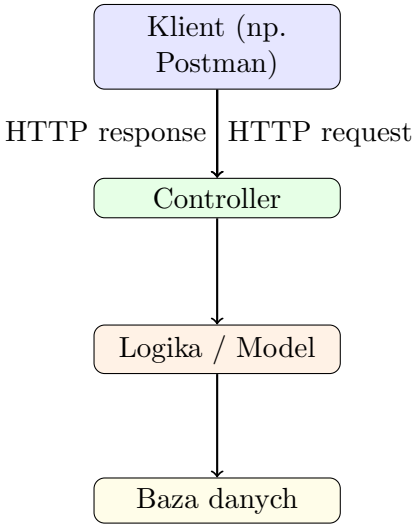
Wspierane są standardy:

- OAuth 2.0,
- OpenID Connect,
- Entra ID (Azure AD),
- Google Identity.

2.11 Walidacja, idempotencja i paginacja

- Walidacja danych wejściowych przy użyciu `DataAnnotations`,
- Idempotencja – operacje powinny dawać ten sam wynik przy wielokrotnym wywołaniu (np. PUT),
- Paginacja – ograniczenie liczby wyników (np. `/api/cities?page=2&size=10`).

2.12 Diagram przepływu zapytania REST API



2.13 HEAD i OPTIONS - dodatkowe metody HTTP

Oprócz podstawowych metod GET, POST, PUT, PATCH i DELETE, REST API wspiera również:

Listing 10: Implementacja HEAD i OPTIONS

```
[HttpHead("{id}")]
public IActionResult CheckCityExists(int id)
{
        var exists = _cities.Any(c => c.Id == id);
        return exists ? Ok() : NotFound();
}

[HttpOptions]
public IActionResult GetCitiesOptions()
{
        Response.Headers.Add("Allow", "GET, POST, PUT, PATCH, DELETE, HE
        return Ok();
}
```

- **HEAD** – sprawdza istnienie zasobu bez pobierania treści (zwraca tylko nagłówki),
- **OPTIONS** – zwraca listę dostępnych metod HTTP dla danego endpointu.

2.14 OAuth 2.0 i OpenID Connect – szczegółowy flow

2.14.1 Authorization Code + PKCE Flow

PKCE (Proof Key for Code Exchange) to rozszerzenie OAuth 2.0 zapewniające bezpieczeństwo dla aplikacji publicznych (SPA, mobile).

Krok 1: Generowanie PKCE

- `code_verifier` – losowy string 43-128 znaków (Base64URL)
- `code_challenge` – Base64URL(SHA256(code_verifier))
- `code_challenge_method` – "S256" (SHA-256)

Krok 2: Żądanie autoryzacji (redirect do przeglądarki)

```
GET https://login.microsoftonline.com/{tenant}/oauth2/v2.0/authorize
?client_id=<APP_ID>
&response_type=code
&redirect_uri=<REDIRECT_URI>
&scope=openid profile email offline_access
&code_challenge=<CHALLENGE>
&code_challenge_method=S256
&state=<RANDOM_STATE>
```

Krok 3: Wymiana kodu na token

```
POST https://login.microsoftonline.com/{tenant}/oauth2/v2.0/token
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=<AUTHORIZATION_CODE>
&redirect_uri=<REDIRECT_URI>
&client_id=<APP_ID>
&code_verifier=<VERIFIER>
```

Odpowiedź:

```
{
        "access_token": "eyJ0eXAiOiJKV1...",
        "id_token": "eyJ0eXAiOiJKV1...",
        "refresh_token": "0.AXoA...",
        "token_type": "Bearer",
        "expires_in": 3600
}
```

2.14.2 Konfiguracja providerów

Google (Google Cloud Console):

- Authorize endpoint: `https://accounts.google.com/o/oauth2/v2/auth`
- Token endpoint: `https://oauth2.googleapis.com/token`
- Scope: `openid email profile`
- Redirect URI: np. `https://localhost:7294/signin-oidc`

Microsoft Entra ID (Azure Portal):

- Tenant ID – z "Directory (tenant) ID"
- Application (client) ID – z "App registrations"
- Authorize endpoint: `https://login.microsoftonline.com/{tenant}/oauth2/v2.0/authorize`
- Token endpoint: `https://login.microsoftonline.com/{tenant}/oauth2/v2.0/token`
- Scope: `openid profile email` (+ `offline_access` dla refresh token)

2.14.3 Integracja w .NET API

Listing 11: Konfiguracja JWT Bearer w Minimal API

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationSchem
.AddJwtBearer(options =>
{
        options.Authority = "https://login.microsoftonline.com/{tenant}/
        options.Audience = "{CLIENT_ID}";
        options.TokenValidationParameters = new TokenValidationParameter
        {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true
        };
});

app.UseAuthentication();
app.UseAuthorization();

// Zabezpieczony endpoint
app.MapGet("/api/secure", [Authorize] () => "Authenticated!")
.RequireAuthorization();
```

2.15 Paginacja i filtrowanie

Listing 12: Przykład paginacji w kontrolerze

```
[HttpGet]
public ActionResult<IEnumerable<CityDto>> GetCities(
[FromQuery] int page = 1,
[FromQuery] int pageSize = 10,
[FromQuery] string? country = null)
{
        var query = _cities.AsQueryable();

        // Filtrowanie
        if (!string.IsNullOrEmpty(country))
        query = query.Where(c => c.Country == country);

        // Paginacja
        var total = query.Count();
        var items = query
        .Skip((page - 1) * pageSize)
        .Take(pageSize)
        .Select(c => new CityDto { Name = c.Name, Country = c.Country })

        Response.Headers.Add("X-Total-Count", total.ToString());
        return Ok(items);
}
```

Przykład użycia: `GET /api/cities?page=2&pageSize=5&country=Poland`

2.16 Idempotencja metod HTTP

Idempotencja oznacza, że wielokrotne wywołanie tej samej operacji daje ten sam efekt co pojedyncze wywołanie.

Metoda	Idempotentna?	Wyjaśnienie
GET	Tak	Wielokrotne pobranie nie zmienia stanu
POST	Nie	Każde wywołanie tworzy nowy zasób
PUT	Tak	Nadpisanie tym samym daje ten sam stan
PATCH	Zależy	Może być idempotentna (zależy od implementacji)
DELETE	Tak	Usunięcie już usuniętego zwraca 404, ale stan się nie zmienia

2.17 HATEOAS (Hypermedia As The Engine Of Application State)

HATEOAS to zasada REST, która polega na zwracaniu w odpowiedzi linków do powiązanych zasobów.

Listing 13: Przykład HATEOAS

```
[HttpGet("{id}")]
public ActionResult<CityWithLinks> GetCity(int id)
{
        var city = _cities.FirstOrDefault(c => c.Id == id);
        if (city == null) return NotFound();

        return Ok(new
        {
                city.Id,
                city.Name,
                city.Country,
                _links = new
                {
                        self = Url.Action(nameof(GetCity), new { id }),
                        update = Url.Action(nameof(UpdateCity), new { id
                        delete = Url.Action(nameof(DeleteCity), new { id
                        all = Url.Action(nameof(GetCities))
                }
        });
}
```

Odpowiedź JSON:

```
{
        "id": 1,
        "name": "Warszawa",
        "country": "Poland",
        "_links": {
                "self": "/api/cities/1",
                "update": "/api/cities/1",
                "delete": "/api/cities/1",
                "all": "/api/cities"
        }
}
```

2.18 Podsumowanie rozdziału

- API REST w .NET jest oparte o architekturę HTTP i format JSON.
- Kontrolery mapują żądania na metody akcji.
- Swagger pozwala testować i generować klienta.
- Kluczowe są poprawne kody HTTP i walidacja danych.
- Uwierzytelnianie i autoryzacja zapewniają bezpieczeństwo aplikacji.

2.19 Pytania kontrolne

1. Jakie są główne metody HTTP używane w REST API?
2. Czym różni się model od DTO?
3. Jak można przetestować API bez użycia Postmana?
4. Jakie są typowe kody błędów HTTP?
5. Do czego służy narzędzie NSwag?

3 Architektury MVC i MVVM w aplikacjach .NET

3.1 Wprowadzenie

Wzorce projektowe MVC (Model–View–Controller) i MVVM (Model–View–ViewModel) są podstawą nowoczesnego tworzenia aplikacji w środowisku .NET. MVC stosuje się głównie w aplikacjach webowych (ASP.NET Core), natomiast MVVM w aplikacjach desktopowych i mobilnych (WPF, MAUI).

3.2 Wzorzec MVC – Model, View, Controller

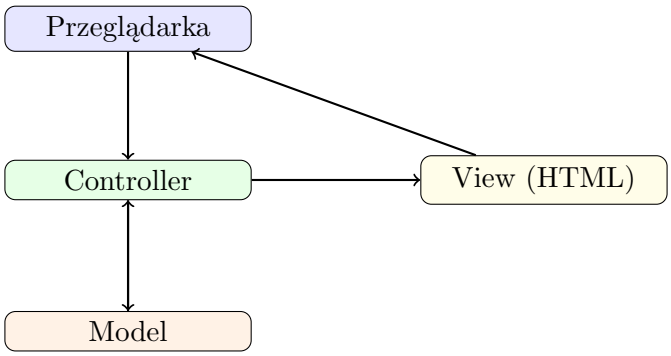
MVC oddziela warstwy aplikacji w celu poprawy czytelności i testowalności kodu.

- **Model** – reprezentuje dane oraz logikę biznesową.
- **View** – odpowiada za prezentację danych użytkownikowi.
- **Controller** – przetwarza żądania, wywołuje odpowiednie metody modelu i przekazuje dane do widoku.

Listing 14: Przykładowy kontroler w ASP.NET Core

```
public class ProductsController : Controller
{
    private readonly IProductService _service;
    public ProductsController(IProductService service)
    {
        _service = service;
    }

    public IActionResult Index()
    {
        var products = _service.GetAll();
```

```
        return View(products);
    }

    public IActionResult Details(int id)
    {
        var product = _service.GetById(id);
        if (product == null)
            return NotFound();
        return View(product);
    }
}
```



3.3 Wzorzec MVVM – Model, View, ViewModel

MVVM jest rozwinięciem koncepcji MVC, zaprojektowanym dla aplikacji z interfejsem graficznym (UI). Oddziela logikę prezentacji (ViewModel) od widoku (View), dzięki czemu można testować logikę bez interakcji z interfejsem użytkownika.

- **Model** – dane aplikacji i logika biznesowa,
- **ViewModel** – warstwa pośrednia między View a Modelem, implementuje `INotifyPropertyChanged`,
- **View** – interfejs użytkownika, najczęściej w XAML.

Listing 15: Przykładowy ViewModel w WPF

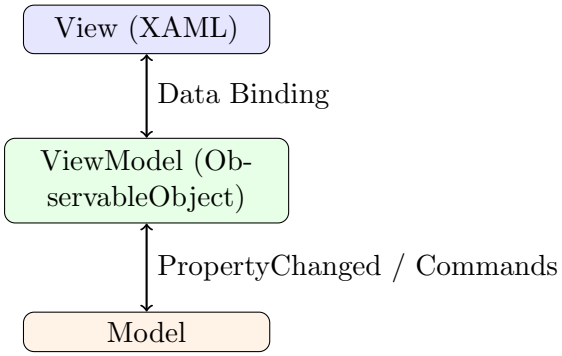
```
public class MainViewModel : INotifyPropertyChanged
{
    private string _message = "Witaj w aplikacji MVVM!";
    public string Message
    {
        get => _message;
        set
        {
            _message = value;
            OnPropertyChanged(nameof(Message));
        }
    }

    public ICommand ChangeMessageCommand { get; }

    public MainViewModel()
```

```
    {
        ChangeMessageCommand = new RelayCommand(ChangeMessage);
    }

    private void ChangeMessage()
    {
        Message = "Tekst został zmieniony!";
    }

    public event PropertyChangedEventHandler? PropertyChanged;
    private void OnPropertyChanged(string propertyName)
        => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyNam
}
```

Listing 16: Fragment widoku XAML (MainWindow.xaml)

```
<Window x:Class="MVVMApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MVVM App" Height="200" Width="400">
    <Grid>
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
            <TextBlock Text="{Binding Message}" FontSize="16" Margin="0,0,0,10"/>
            <Button Content="Zmień tekst" Command="{Binding ChangeMessageCommand
        </StackPanel>
    </Grid>
</Window>
```

3.4 Diagram przepływu MVVM



3.5 Porównanie MVC i MVVM

Cecha	**MVC**	**MVVM**
Zastosowanie	Aplikacje webowe	Aplikacje desktopowe/mobilne (WPF, MAUI)
Warstwa prezentacji	View + Controller	View + ViewModel
Powiązanie danych	Ręczne przekazywanie danych	Data Binding (TwoWay)
Obsługa zdarzeń	Metody kontrolera	Komendy (ICommand)
Testowalność logiki	Średnia	Bardzo wysoka

3.6 Binding i konwertery danych

Binding w MVVM umożliwia dwukierunkową synchronizację danych pomiędzy widokiem a modelem.

Listing 17: Przykład konwertera wartości

```
public class BoolInverseConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, Cultu
        => !(bool)value;

    public object ConvertBack(object value, Type targetType, object parameter, C
        => !(bool)value;
}
```

Listing 18: Zastosowanie konwertera w XAML

```
<Button Content="Wyślij"
        IsEnabled="{Binding IsBusy, Converter={StaticResource BoolInverseConvert
```

3.7 Walidacja danych

Listing 19: Walidacja danych z użyciem DataAnnotations

```
public class User
{
    [Required]
    public string Name { get; set; } = string.Empty;

    [Range(18, 99)]
    public int Age { get; set; }
}
```

3.8 Zalety MVVM

- Łatwe testowanie logiki prezentacji bez UI.
- Spójna architektura z separacją odpowiedzialności.
- Łatwe utrzymanie i rozwój aplikacji.
- Możliwość współdzielenia logiki między platformami (MAUI).

3.9 Wady MVVM

- Większa złożoność początkowa.
- Trudniejsze debugowanie powiązań (bindings).
- Więcej kodu pomocniczego (np. RelayCommand, BaseViewModel).

3.10 Podsumowanie rozdziału

- MVC stosujemy w aplikacjach webowych, MVVM w desktopowych i mobilnych.
- MVVM wprowadza binding, ICommand i testowalność logiki prezentacji.
- Wzorce zapewniają lepszą separację odpowiedzialności i czystszy kod.

3.11 Pytania kontrolne

1. Jakie są trzy główne komponenty MVC i ich funkcje?
2. W jaki sposób MVVM ułatwia testowanie aplikacji?
3. Do czego służy interfejs `INotifyPropertyChanged`?
4. Jak działają konwertery wartości (ValueConverter) w XAML?
5. Jakie są różnice w przepływie danych między MVC a MVVM?

4 Aplikacje WPF, MAUI i Dependency Injection w .NET

4.1 Wprowadzenie

WPF (Windows Presentation Foundation) oraz .NET MAUI (Multi-platform App UI) są frameworkami do tworzenia aplikacji z interfejsem graficznym.

- **WPF** – przeznaczony dla systemu Windows, używa XAML i wzorca MVVM.
- **.NET MAUI** – framework wieloplatformowy (Windows, Android, iOS, macOS), również oparty na XAML i MVVM.

4.2 Struktura projektu WPF

Projekt WPF składa się z plików:

- `App.xaml` – deklaracja zasobów globalnych i punkt wejścia,
- `MainWindow.xaml` – główne okno aplikacji,
- `ViewModels/` – logika prezentacji,
- `Models/` – dane i logika biznesowa.

Listing 20: Fragment App.xaml

```
<Application x:Class="WpfApp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <ResourceDictionary>
            <SolidColorBrush x:Key="PrimaryBrush" Color="#007ACC" />
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

4.3 Struktura projektu MAUI

Projekt MAUI ma podobny układ, ale rozszerzony o platformy:

- `MauiProgram.cs` – główny punkt konfiguracji (DI, serwisy),
- `App.xaml` – konfiguracja wizualna aplikacji,
- `MainPage.xaml` – główny widok,

- katalogi: `Platforms/Android`, `Platforms/iOS`, `Platforms/Windows`.

Listing 21: Plik MauiProgram.cs

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
            });

        // Rejestracja serwis w
        builder.Services.AddSingleton<MainPageViewModel>();
        builder.Services.AddTransient<ApiService>();

        return builder.Build();
    }
}
```

4.4 Dependency Injection (DI)

Dependency Injection to wzorzec, który umożliwia wstrzykiwanie zależności zamiast ich tworzenia bezpośrednio w kodzie. Ułatwia testowanie i utrzymanie kodu, pozwalając wymieniać komponenty bez modyfikacji logiki aplikacji.

Listing 22: Przykład DI w WPF

```
public partial class MainWindow : Window
{
    private readonly IWeatherService _service;

    public MainWindow(IWeatherService service)
    {
        InitializeComponent();
        _service = service;
        DataContext = new MainViewModel(service);
    }
}
```

Listing 23: Rejestracja serwisów w konfiguracji

```
services.AddSingleton<IWeatherService, WeatherService>();
services.AddTransient<MainViewModel>();
```

4.5 Zalety DI

- Redukuje sprzężenie między klasami,
- Umożliwia łatwe testowanie jednostkowe,

- Ułatwia utrzymanie i rozwój aplikacji,
- Pozwala dynamicznie wymieniać implementacje interfejsów.

4.6 Cykl życia aplikacji MAUI

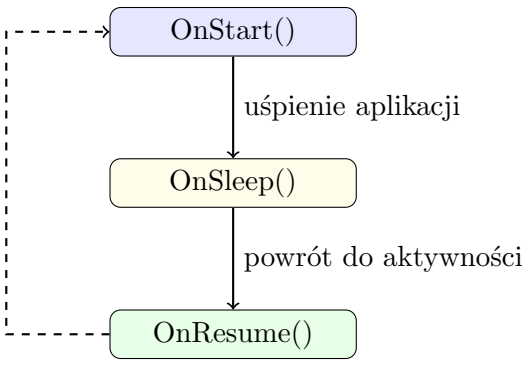
W MAUI aplikacja przechodzi przez kilka etapów cyklu życia:

- `OnStart()` – uruchomienie aplikacji,
- `OnSleep()` – uśpienie (np. minimalizacja),
- `OnResume()` – wznowienie działania.

Listing 24: Cykl życia aplikacji w MAUI

```
protected override void OnStart()
{
    Console.WriteLine("Aplikacja wystartowa a");
}

protected override void OnSleep()
{
    Console.WriteLine("Aplikacja u piona");
}

protected override void OnResume()
{
    Console.WriteLine("Aplikacja wznowiona");
}
```

4.7 Diagram cyklu życia aplikacji MAUI



4.8 Tworzenie usług i integracja z API

Aplikacje WPF i MAUI często komunikują się z API REST poprzez serwisy.

Listing 25: Przykład klasy serwisowej w MAUI

```
public class ApiService
{
    private readonly HttpClient _client = new();

    public async Task<List<CityDto>> GetCitiesAsync()
```

```
    {
        var response = await _client.GetAsync("https://localhost:7294/api/cities
        response.EnsureSuccessStatusCode();
        var json = await response.Content.ReadAsStringAsync();
        return JsonSerializer.Deserialize<List<CityDto>>(json)!;
    }
}
```

Listing 26: Przykład użycia serwisu w ViewModelu

```
public class CitiesViewModel : INotifyPropertyChanged
{
    private readonly ApiService _api;
    public ObservableCollection<CityDto> Cities { get; set; } = new();

    public CitiesViewModel(ApiService api)
    {
        _api = api;
        LoadCitiesCommand = new RelayCommand(async () => await LoadCities());
    }

    public ICommand LoadCitiesCommand { get; }

    private async Task LoadCities()
    {
        var result = await _api.GetCitiesAsync();
        Cities.Clear();
        foreach (var c in result)
            Cities.Add(c);
    }

    public event PropertyChangedEventHandler? PropertyChanged;
}
```

4.9 Powiązanie z interfejsem w MAUI

Listing 27: Fragment MainPage.xaml

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:vm="clr-namespace:MauiApp.ViewModels"
             x:Class="MauiApp.MainPage">

    <ContentPage.BindingContext>
        <vm:CitiesViewModel />
    </ContentPage.BindingContext>

    <CollectionView ItemsSource="{Binding Cities}">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <StackLayout Padding="10">
                    <Label Text="{Binding Name}" FontSize="18" />
```

```
                <Label Text="{Binding Population}" FontSize="14" TextColor="
            </StackLayout>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>

<Button Text="Za aduj miasta" Command="{Binding LoadCitiesCommand}" />
</ContentPage>
```

4.10 MAUI Shell Navigation

Shell to zaawansowany system nawigacji w .NET MAUI, obsługujący routing, Flyout i TabBar.

4.10.1 Rejestracja tras

Listing 28: Rejestracja route w AppShell.xaml.cs

```
public AppShell()
{
        InitializeComponent();

        // Rejestracja tras
        Routing.RegisterRoute("details", typeof(DetailsPage));
        Routing.RegisterRoute("edit", typeof(EditPage));
}
```

4.10.2 Nawigacja z parametrami

Listing 29: Nawigacja do strony szczegółów

```
// W ViewModelu lub code-behind
await Shell.Current.GoToAsync("details", new Dictionary<string, object>
{
        {"Item", selectedItem},
        {"ItemId", selectedItem.Id}
});
```

4.10.3 Odbieranie parametrów

Listing 30: Odbieranie parametru w DetailsPage

```
[QueryProperty(nameof(Item), "Item")]
[QueryProperty(nameof(ItemId), "ItemId")]
public partial class DetailsPage : ContentPage
{
        public Item Item { get; set; }
        public int ItemId { get; set; }

        protected override void OnAppearing()
        {
                base.OnAppearing();
                Console.WriteLine($"Otrzymano Item: {Item.Name}, ID: {Ite
```

```
        }
}
```

4.10.4 Shell Flyout i TabBar

Listing 31: Przykład Shell z Flyout

```
<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
x:Class="MyApp.AppShell">

<FlyoutItem Title="Home" Icon="home.png">
<ShellContent Route="home" ContentTemplate="{DataTemplate local:HomePage
</FlyoutItem>

<FlyoutItem Title="Settings" Icon="settings.png">
<ShellContent Route="settings" ContentTemplate="{DataTemplate local:Setti
</FlyoutItem>

<TabBar>
<Tab Title="Browse" Icon="browse.png">
<ShellContent Route="browse" ContentTemplate="{DataTemplate local:BrowseP
</Tab>
<Tab Title="Profile" Icon="profile.png">
<ShellContent Route="profile" ContentTemplate="{DataTemplate local:Profil
</Tab>
</TabBar>
</Shell>
```

4.11 MAUI Essentials – dostęp do funkcji platformy

MAUI Essentials to zestaw API zapewniających dostęp do funkcji urządzeń (geolokalizacja, czujniki, kamera, itp.).

4.11.1 Connectivity – sprawdzanie połączenia

Listing 32: Sprawdzanie dostępu do Internetu

```
var networkAccess = Connectivity.Current.NetworkAccess;
if (networkAccess == NetworkAccess.Internet)
{
        Console.WriteLine("Po czenie z Internetem dost pne");
}
else
{
        Console.WriteLine("Brak po czenia");
}

// Nas uchiwanie zmian
Connectivity.Current.ConnectivityChanged += (sender, args) =>
{
        Console.WriteLine($"Zmiana po czenia: {args.NetworkAccess}");
};
```

4.11.2 Geolocation – lokalizacja

Listing 33: Pobieranie lokalizacji GPS

```
try
{
        var location = await Geolocation.GetLocationAsync(new Geolocation
        {
                DesiredAccuracy = GeolocationAccuracy.Medium,
                Timeout = TimeSpan.FromSeconds(10)
        });

        if (location != null)
        {
                Console.WriteLine($"Lat: {location.Latitude}, Lon: {loca
        }
}
catch (FeatureNotSupportedException)
{
        Console.WriteLine("Geolokalizacja nie jest wspierana na tym urz
}
catch (PermissionException)
{
        Console.WriteLine("Brak uprawnie do lokalizacji");
}
```

4.11.3 DeviceInfo – informacje o urządzeniu

Listing 34: Informacje o urządzeniu

```
Console.WriteLine($"Model: {DeviceInfo.Model}");
Console.WriteLine($"Producent: {DeviceInfo.Manufacturer}");
Console.WriteLine($"Platforma: {DeviceInfo.Platform}");
Console.WriteLine($"Wersja: {DeviceInfo.VersionString}");
Console.WriteLine($"Typ: {DeviceInfo.Idiom}"); // Phone, Tablet, Desktop
```

4.12 Kompilacja warunkowa – kod specyficzny dla platformy

W MAUI można pisać kod specyficzny dla danej platformy przy użyciu dyrektyw preprocesora.

Listing 35: Kod warunkowy dla Android i iOS

```
public void ShowNativeAlert()
{
        #if ANDROID
        Android.Widget.Toast.MakeText(
        Android.App.Application.Context,
        "To jest Android",
        Android.Widget.ToastLength.Short
        ).Show();
        #elif IOS
        var alert = UIKit.UIAlertController.Create(
        "Informacja",
```

```
            "To jest iOS",
            UIKit.UIAlertControllerStyle.Alert
            );
            alert.AddAction(UIKit.UIAlertAction.Create("OK", UIKit.UIAlertAc
            // Wywietl alert
            #elif WINDOWS
            System.Diagnostics.Debug.WriteLine("To jest Windows");
            #endif
    }
```

Dostępne dyrektywy:

- `#if ANDROID` – kod tylko dla Android,
- `#if IOS` – kod tylko dla iOS,
- `#if WINDOWS` – kod tylko dla Windows,
- `#if MACCATALYST` – kod dla macOS (Catalyst).

4.13 Różnice WPF vs MAUI – rozszerzone

Cecha	**WPF**	**MAUI**
Platformy	Tylko Windows	Android, iOS, Windows, macOS
Projekt	Klasyczny (.csproj)	Single Project
Customizacja UI	Dependency Properties	Handlers
Dostęp do natywnych API	Ograniczony (P/Invoke)	MAUI Essentials + kod warunkowy
Lifecycle	OnStartup, OnExit	OnStart, OnSleep, OnResume
Hot Reload	Desktopowy	Wieloplatformowy (emulatory + urządzenia)
Uprawnienia	Nie wymagane	Wymagane (kamera, lokalizacja, etc.)

4.14 Podsumowanie rozdziału

- WPF i MAUI używają XAML i wzorca MVVM.
- Dependency Injection umożliwia niezależność komponentów.
- Cykl życia aplikacji MAUI obejmuje start, uśpienie i wznowienie.
- Serwisy wstrzykuje się do ViewModeli i komunikują z API.

4.15 Pytania kontrolne

1. Jakie są główne różnice między WPF a .NET MAUI?
2. Do czego służy plik `MauiProgram.cs`?
3. Jak działa Dependency Injection i jakie ma zalety?
4. Jak wygląda cykl życia aplikacji w MAUI?
5. Jak ViewModel komunikuje się z serwisem API?

5 Testy jednostkowe, typowe błędy i ściąga z komend .NET

5.1 Testy jednostkowe w .NET

Testy jednostkowe (unit tests) pozwalają zweryfikować działanie pojedynczych metod lub komponentów bez uruchamiania całej aplikacji.

5.1.1 Tworzenie projektu testowego

Listing 36: Tworzenie projektu testowego w .NET

```
dotnet new xunit -n ShopAPI.Tests
dotnet add ShopAPI.Tests reference ShopAPI.Api
```

5.1.2 Przykład prostego testu

Listing 37: Przykładowy test jednostkowy

```
public class MathServiceTests
{
    [Fact]
    public void Add_ReturnsSum()
    {
        var service = new MathService();
        var result = service.Add(2, 3);
        Assert.Equal(5, result);
    }
}
```

5.1.3 Testy kontrolerów API

Testy kontrolerów można wykonywać z wykorzystaniem pakietu `Microsoft.AspNetCore.Mvc.Testing`.

Listing 38: Przykład testu kontrolera CitiesController

```
public class CitiesControllerTests : IClassFixture<WebApplicationFactory<Program>
{
    private readonly HttpClient _client;

    public CitiesControllerTests(WebApplicationFactory<Program> factory)
    {
        _client = factory.CreateClient();
    }

    [Fact]
    public async Task GetCities_ReturnsSuccess()
    {
        var response = await _client.GetAsync("/api/cities");
        response.EnsureSuccessStatusCode();
        var json = await response.Content.ReadAsStringAsync();
        Assert.Contains("Warszawa", json);
    }
}
```

5.2 Testy z wykorzystaniem DI i Mocków

Do testowania zależności stosuje się tzw. **mocki** – sztuczne implementacje interfejsów. Biblioteki takie jak `Moq` pozwalają emulować zachowanie serwisów w testach.

Listing 39: Przykład testu z użyciem Moq

```
var mockService = new Mock<IWeatherService>();
mockService.Setup(s => s.GetTemperature()).Returns(25);

var vm = new WeatherViewModel(mockService.Object);
Assert.Equal("25 C", vm.DisplayTemperature);
```

5.3 Testy UI i integracyjne

- **Testy UI** – np. przy użyciu frameworka `Playwright` lub `Selenium`,
- **Testy integracyjne** – sprawdzają współdziałanie komponentów aplikacji,
- **Testy wydajnościowe** – badają czas odpowiedzi API.

5.4 Typowe błędy na kolokwium

- Mylenie `MVC` z `MVVM` – brak zrozumienia roli ViewModelu.
- Brak implementacji `INotifyPropertyChanged` – dane nie aktualizują się w interfejsie.
- Zapominanie o `await` przy wywołaniu metod asynchronicznych.
- Niepoprawna rejestracja serwisów w DI (np. brak `AddSingleton`).
- Zła konfiguracja routingu w kontrolerze (błędny atrybut `[Route]`).
- Niewłaściwe kody HTTP w odpowiedziach (np. zwracanie 200 zamiast 201).
- Użycie `code-behind` zamiast MVVM w WPF/MAUI.
- Błędny binding (np. literówka w nazwie właściwości w XAML).
- Zapomniany `PropertyChanged.Invoke()` – UI nie aktualizuje danych.
- Brak testów jednostkowych dla metod serwisowych.

5.5 Ściąga z komend .NET CLI

Listing 40: Najważniejsze komendy .NET CLI

```
dotnet new webapi -n MyApi          # Tworzenie nowego projektu Web API
dotnet new maui -n MyApp            # Tworzenie nowej aplikacji MAUI
dotnet run                          # Uruchomienie projektu
dotnet build                        # Kompilacja projektu
dotnet test                         # Uruchomienie test w
dotnet restore                      # Przywr cenie pakiet w NuGet
dotnet add package <nazwa>          # Dodanie paczki NuGet
dotnet watch run                    # Automatyczne od wie anie aplikacji
dotnet clean                        # Czyszczenie kompilacji
dotnet publish -c Release           # Publikacja projektu w trybie Release
```

5.6 Skróty klawiszowe w Visual Studio Code

- `Ctrl + ‘` – otwarcie terminala,
- `Ctrl + Shift + P` – paleta poleceń,
- `Ctrl + K, Ctrl + C` – komentowanie kodu,
- `Ctrl + F5` – uruchomienie aplikacji,
- `Alt + Shift + F` – formatowanie kodu.

5.7 Pytania kontrolne

1. Jakie są główne etapy cyklu życia testu jednostkowego?
2. Do czego służy biblioteka Moq?
3. Czym różni się test jednostkowy od integracyjnego?
4. Jakie błędy najczęściej pojawiają się przy implementacji MVVM?
5. Jakie są podstawowe komendy .NET CLI przy tworzeniu i testowaniu projektu?

5.8 Podsumowanie końcowe

- Testy jednostkowe zapewniają stabilność i niezawodność aplikacji.
- DI pozwala na łatwe testowanie komponentów dzięki mockom.
- Znajomość cyklu życia aplikacji i wzorców architektonicznych to klucz do zdania kolokwium.
- Warto pamiętać o poprawnych kodach HTTP, bindingach i walidacji danych.

„Dobry kod to taki, który nie wymaga komentarzy, a jego testy mówią wszystko.”