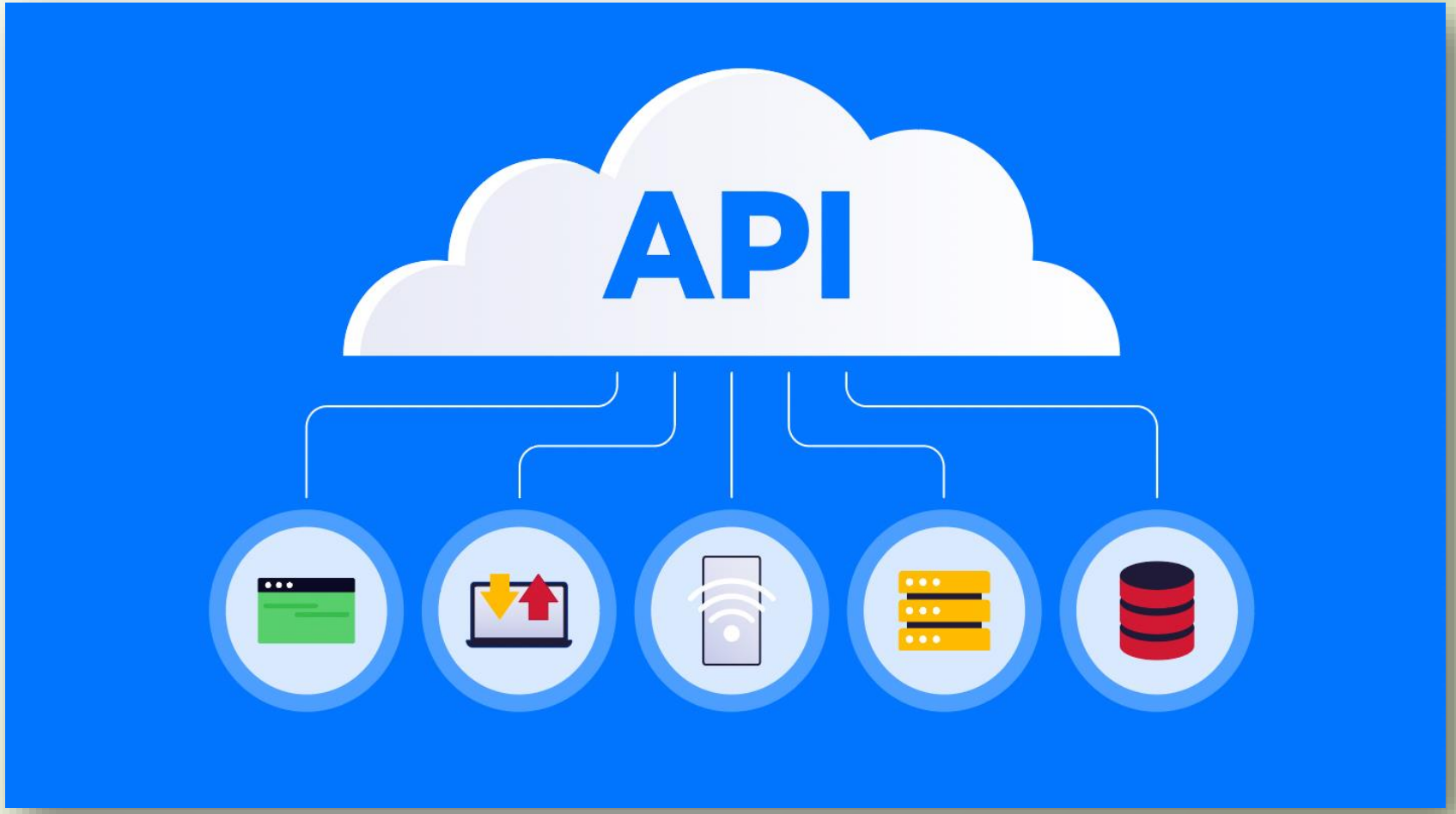
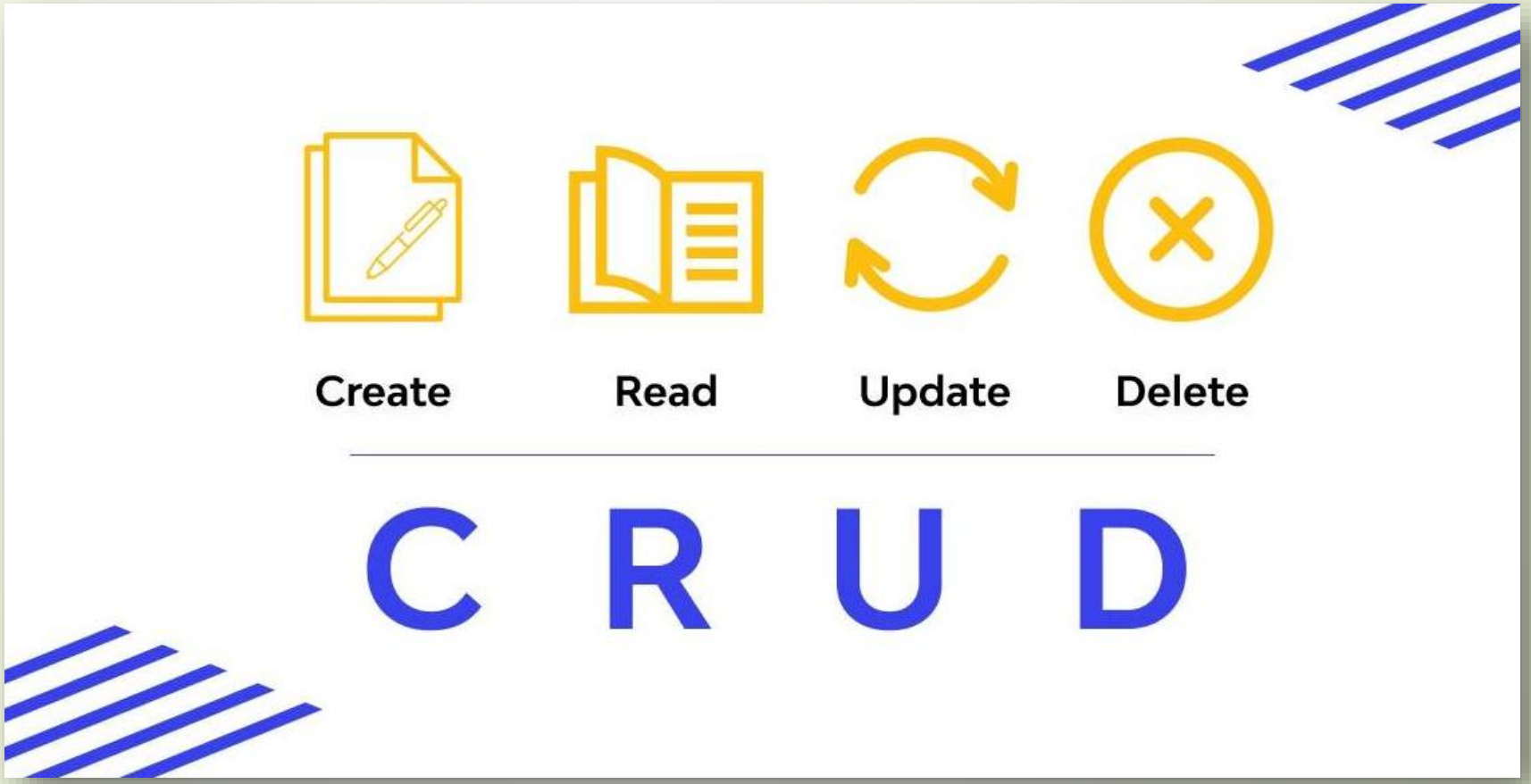
Przetwarzanie rozproszone

Tworzenie internetowego interfejsu API



Tworzenie internetowego interfejsu API

W pełni funkcjonalne Api powinno obsługiwać operacje CRUD (Create Read Update Delete). Czynności te są powszechnie realizowane w powiązaniu z bazami danych.



Tworzenie internetowego interfejsu API

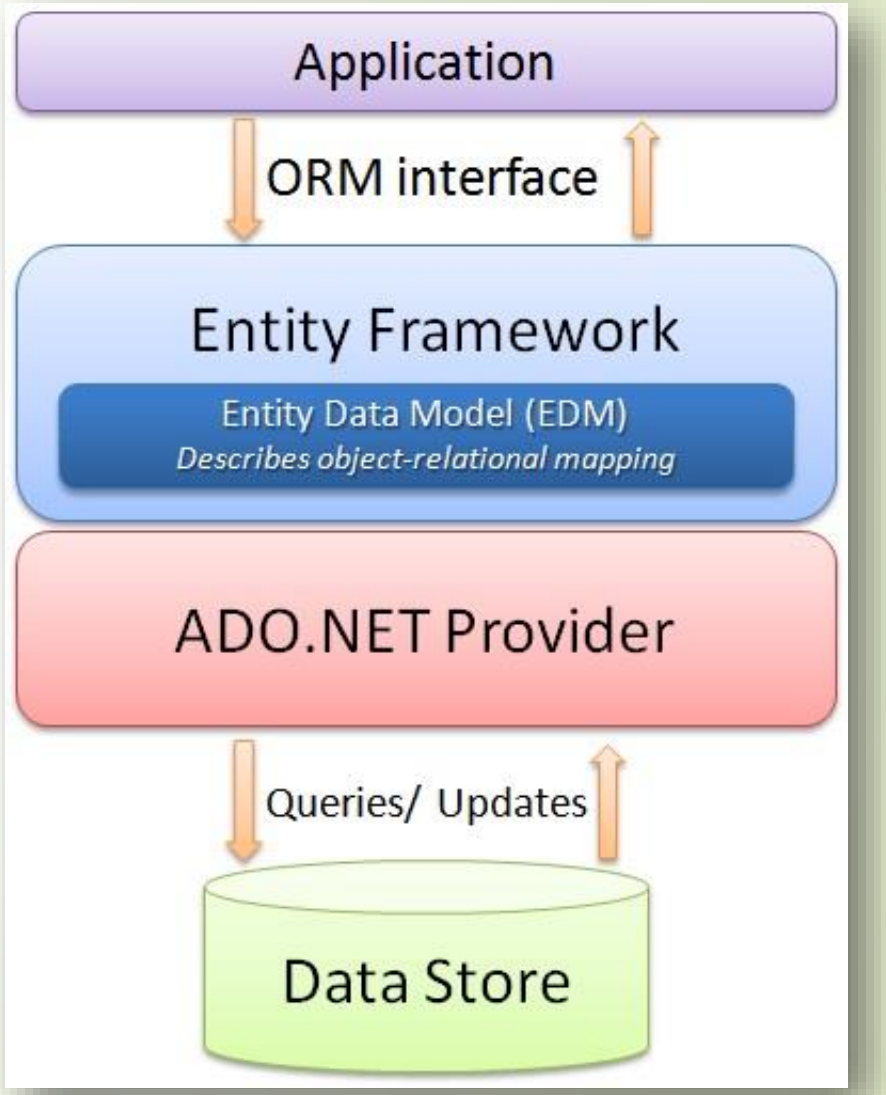
Aby uzyskać powiązanie z bazą danych wykorzystuje się technikę programowania opartą na ORM

ORM (Object Related Mapping) – technika pozwalająca poprzez wykorzystanie odpowiedniego frameworka na rzutowanie struktury obiektowej na strukturę bazodanową. Popularnymi frameworkami realizującymi rozwiązania ORM są Entity, Nhibernate.

Tworzenie internetowego interfejsu API

Zalety ORM:

- Cała logika dostępu do bazy danych może być napisana w języku wyższego poziomu.
- Bazowe dane mogą być zastąpione bez większego narzutu ponieważ cała logika biznesowa dostępu do danych jest na wyższym poziomie.
- Rozwiązanie można powiązać z usługą RestApi



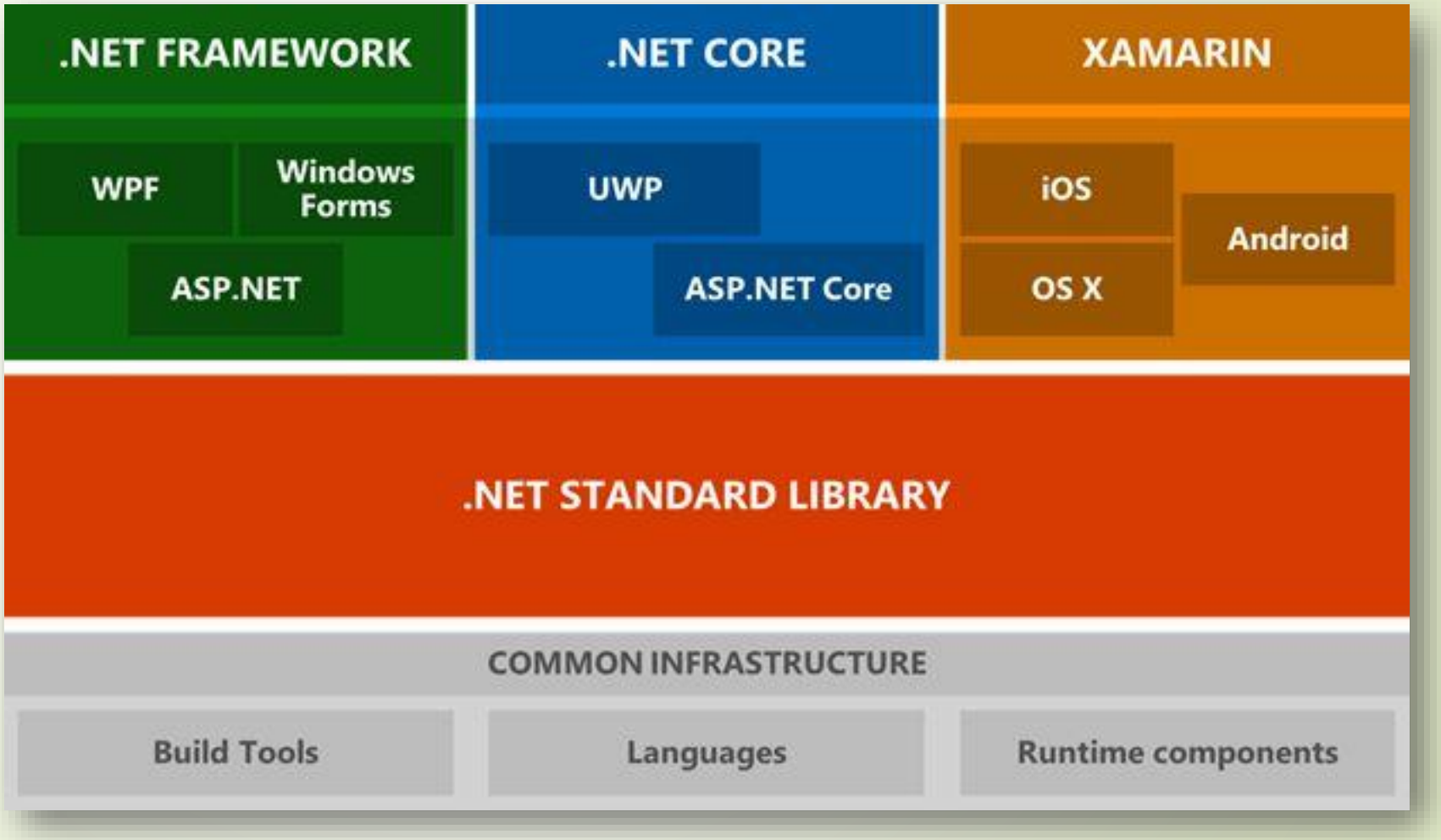
Tworzenie internetowego interfejsu API

Plain Old CLR Object (**POCO**) – specyficzna klasa służącego zazwyczaj tylko do reprezentacji danych. Obiekty POCO są często definiowane przez programistę, kiedy celem jest stworzenie systemu skalowalnego i łatwego do modyfikacji, gdzie istotne jest wyraźne ograniczenie obszarów styku jego własnego kodu z kodem zewnętrznych frameworków.

```
public class Person
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string PhoneNumber { get; set; }
}
```

Tworzenie internetowego interfejsu API

Tworzenie internetowego interfejsu API w środowisku .Net opiera się na technologii ASP.Net Core



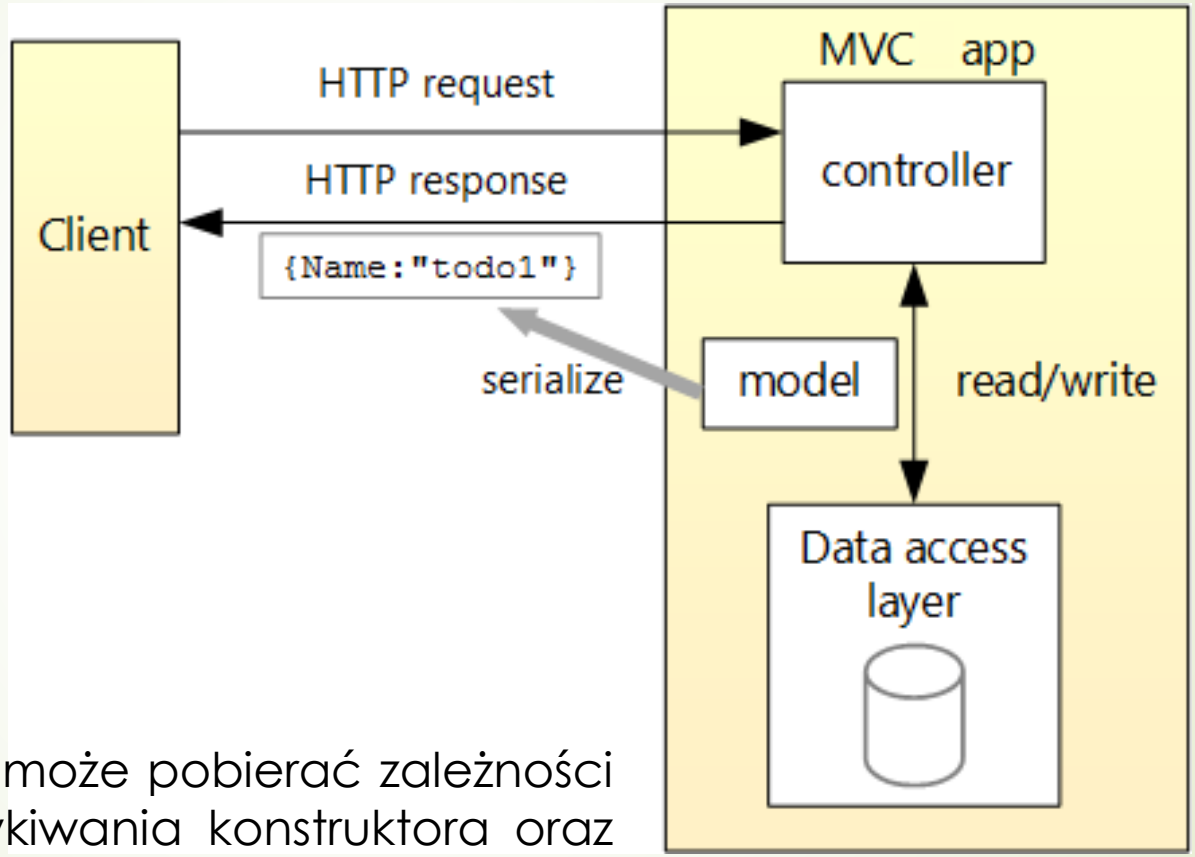
Tworzenie internetowego interfejsu API

ASP.NET Core obsługuje dwa podejścia do tworzenia interfejsów API: podejście oparte na kontrolerze i minimalne interfejsy API. *Kontrolery* w projekcie interfejsu API to klasy pochodzące z *ControllerBase* klasy . Minimalne interfejsy API definiują punkty końcowe z logicznymi procedurami obsługi w wyrażeniach lambd lub metodach.

Minimal API	ASP.NET API
Single file based approach	Class files are separated in Models, Controllers and Views
Granular control on your API	MVC takes more control
Minimum components are required to build API	Relies on Controller, Model and Views
Better performance	Good performance
Easy to learn by new developers	Need to understand the basics like Models, Controllers and Views.

https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio

Tworzenie internetowego interfejsu API



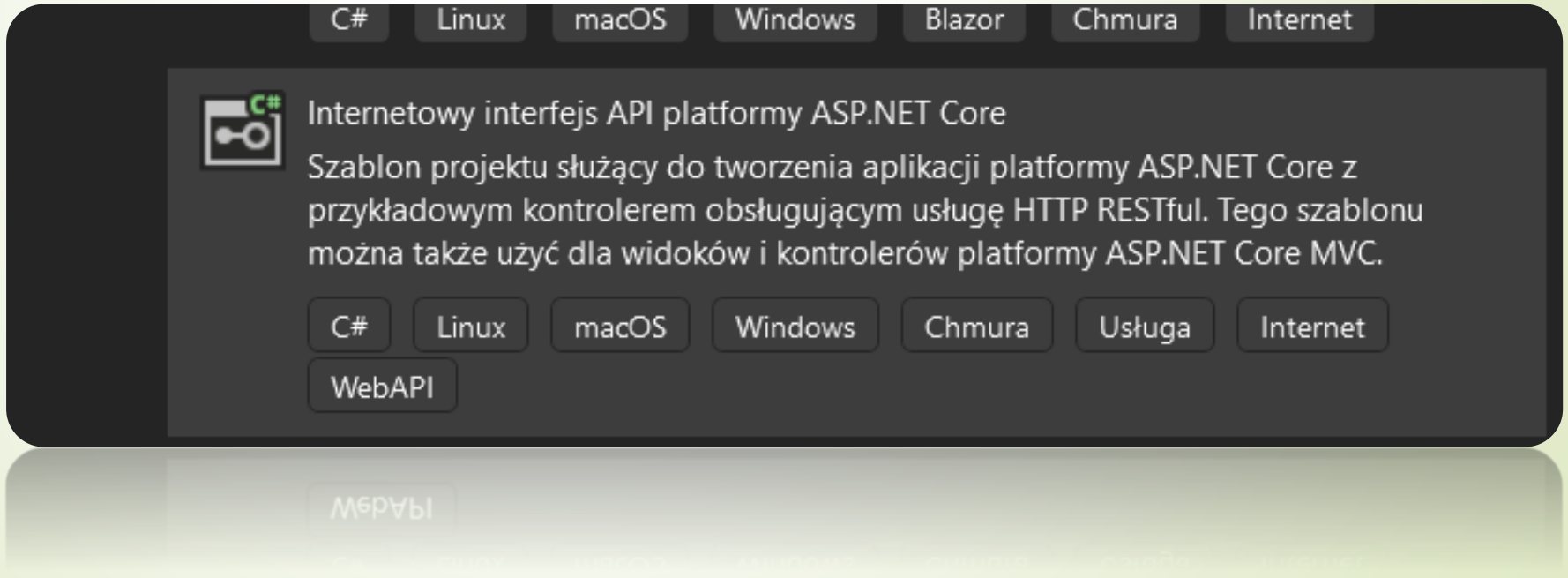
Kontroler jest klasą, która może pobierać zależności za pośrednictwem wstrzykiwania konstruktora oraz wykorzystywać wzorce projektowe.

Tworzenie internetowego interfejsu API

Interfejs API	opis	Treść wniosku	Treść odpowiedzi
`GET /api/todoitems`	Pobieranie wszystkich elementów do wykonania	Brak	Tablica elementów do wykonania
`GET /api/todoitems/{id}`	Pobieranie elementu według identyfikatora	Brak	Element do wykonania
`POST /api/todoitems`	Dodawanie nowego elementu	Element do wykonania	Element do wykonania
`PUT /api/todoitems/{id}`	Aktualizowanie istniejącego elementu	Element do wykonania	Brak
`DELETE /api/todoitems/{id}`	Usuwanie elementu	Brak	Brak

Tworzenie internetowego interfejsu API

Typ projektu jaki należy wybrać w środowisku Visual Studio 2022



Tworzenie internetowego interfejsu API

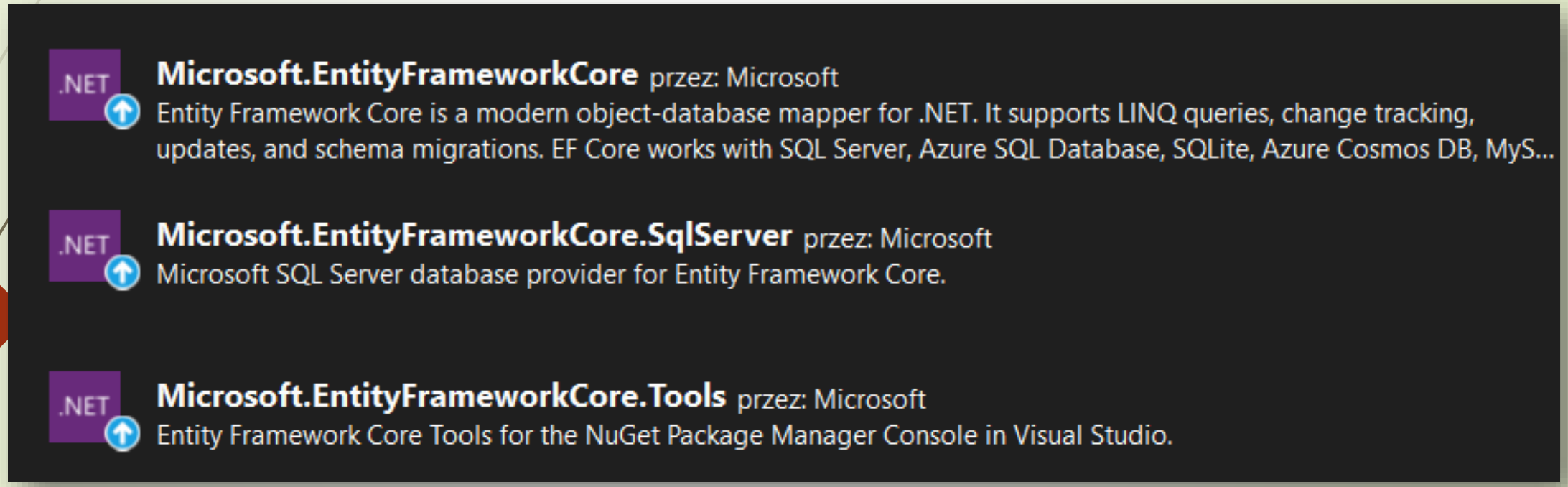
Po utworzeniu projektu należy utworzyć klasę POCO modelu

```
namespace TodoApi.Models;

public class TodoItem
{
    public long Id { get; set; }
    public string? Name { get; set; }
    public bool IsComplete { get; set; }
}
```

Tworzenie internetowego interfejsu API

W celu wykorzystania podejścia ORM należy doinstalować (menadżer pakietów Nugget) framework np.Entity



Tworzenie internetowego interfejsu API

Następnie utworzyć klasę dziedziczącą z DbContext reprezentującą strukturę bazy danych

```
using Microsoft.EntityFrameworkCore;
using MyRestApi.Models;

namespace MyRestApi.DbServices
{
    Odwołania: 7
    public class AppDbContext:DbContext
    {
        Odwołania: 0
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

        Odwołania: 6
        public DbSet<PersonModel> Persons { get; set; }

    }
}
```

Tworzenie internetowego interfejsu API

W pliku appsetings.json dołączamy connection string zawierający m.in. Informację o lokalizacji bazy danych

```
"ConnectionStrings": {
  "PersonConnection":
"Server=(localdb)\\mssqllocaldb;Database=AppDbContext;Trusted_Connection=True;
MultipleActiveResultSets=true"
},
```

Tworzenie internetowego interfejsu API

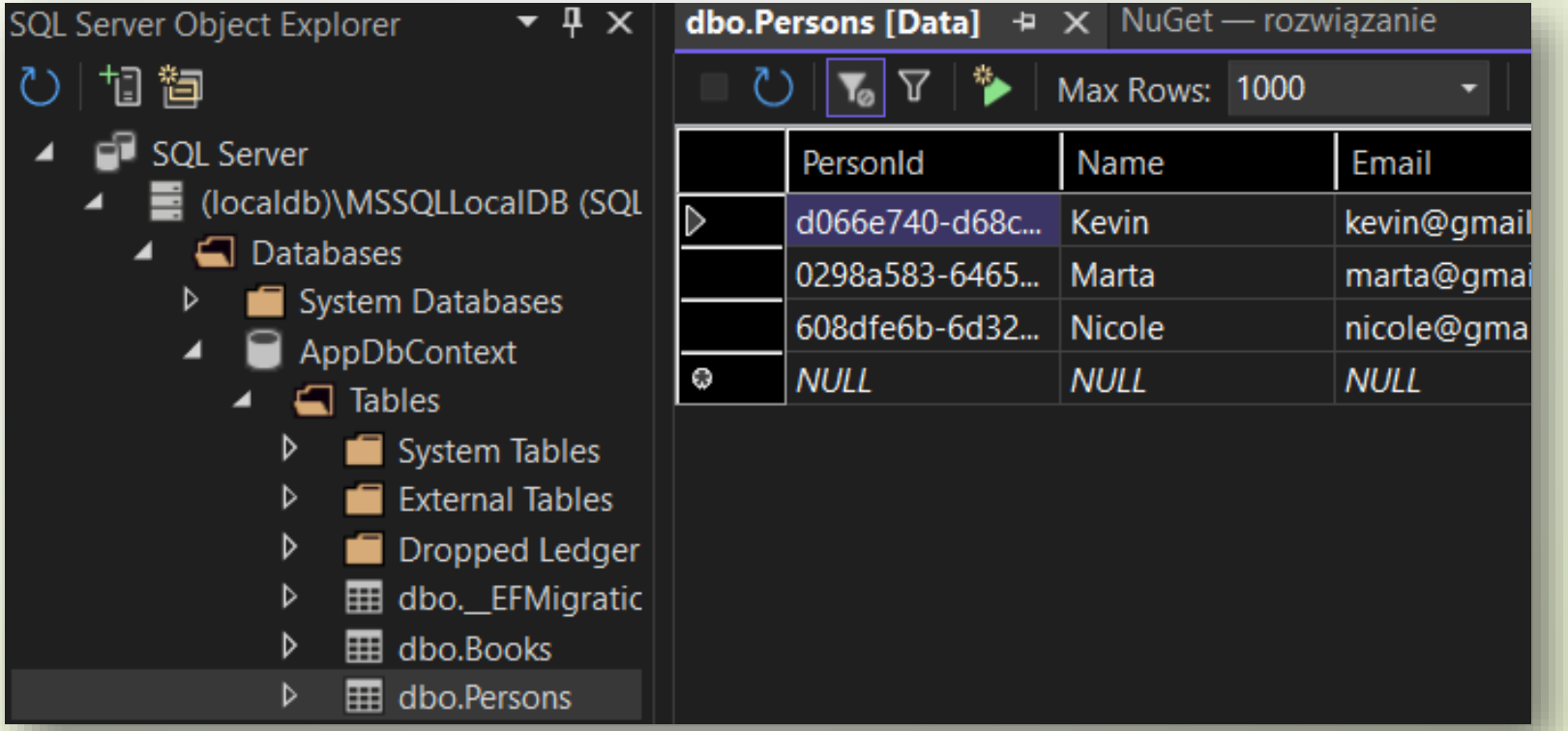
W pliku Program.cs należy zarejestrować bazę danych (w oparciu o wpis w ConnectionString)

```
// Register database
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("PersonConnection")));
```

Tworzenie internetowego interfejsu API

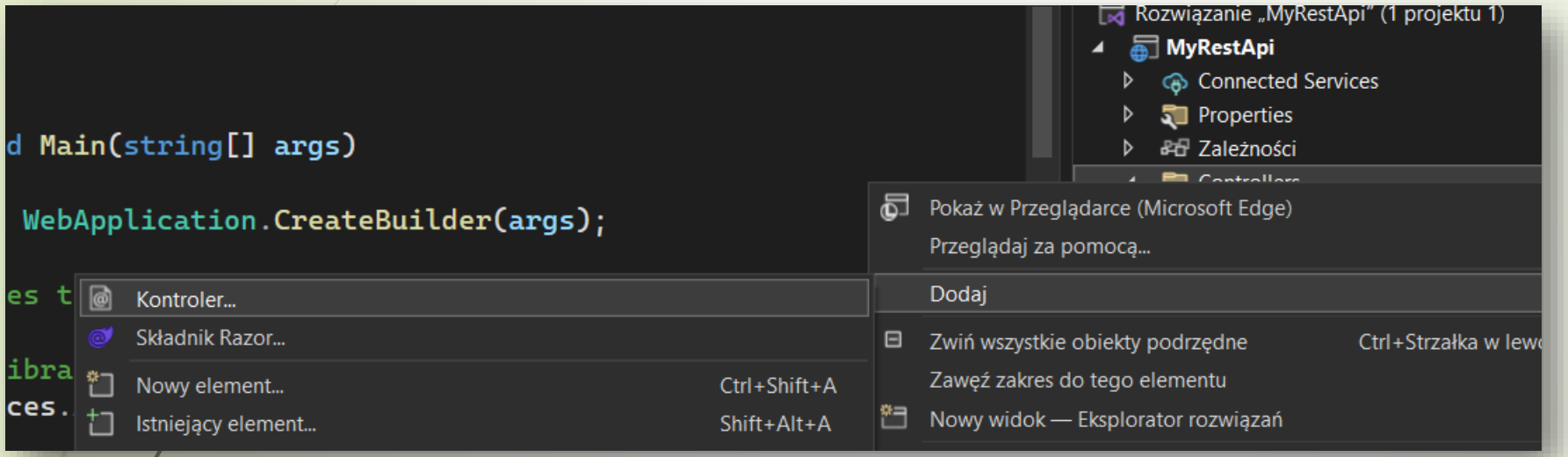
W konsoli menadżera pakietów Nugget wprowadzamy polecenia:

Add-Migration
Update - database



Tworzenie internetowego interfejsu API

Następnym krokiem jest utworzenie kontrolera i odpowiednich metod:




```
[HttpPost]
Odwołania: 0
public async Task<ActionResult<PersonModel>> AddPerson(PersonModel person)
{
    var dbPerson = await personService.AddPerson(person);

    if (dbPerson == null)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, $"{pers
    }

    return CreatedAtAction("GetPerson", new { id = person.PersonId }, perso
}
```

Tworzenie internetowego interfejsu API

Uruchomienie aplikacji: