ASP.NET Core – Minimal API

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

13 février 2026

Plan

Plan

6. Endpoints Minimal API : GET / POST
7. Validation : DataAnnotations + erreurs
8. Middleware CORS
9. Circular references
   - `[JsonIgnore]`
   - `IgnoreCycles`
10. `AutoMapper`

ASP.NET Core

Minimal API

- Introduites avec **.NET 6** : style “endpoint-first”.
- Alternative aux **Controllers Web API** pour des APIs simples / microservices.
- Moins de classes, moins de “boilerplate” : endpoints dans `Program.cs`.

ASP.NET Core

Création d'un projet **Web API** avec **Visual Studio Community 2026**

- Allez dans `Fichier > Nouveau > Projet`
- Dans la zone de recherche, saisissez `web api`
- Sélectionner `API Web ASP.NET Core`
- Remplir le champs `Nom` par `CoursMinimalApi`
- Décocher `Utiliser des contrôleurs`
- Validez et attendre la fin de création du projet

ASP.NET Core

Création du projet avec une commande

```
dotnet new web -n CoursMinimalApi cd CoursMinimalApi
```

ASP.NET Core

Utiliser `NuGet` pour Télécharger les dépendances

- Faire clic droit sur `Dépendances` dans l'`Explorateur de solution`
- Choisir `Gérer les packages NuGet`
- Aller dans l'onglet `Parcourir` et chercher **Scalar.AspNetCore**
- Choisir la dernière version stable et installer
- Accepter, attendre la fin de l'installation

ASP.NET Core

Structure minimale d'une Minimal API si vous utilisez Scalar

```
using Scalar.AspNetCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddOpenApi();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
    app.MapScalarApiReference("/docs");
}

app.UseHttpsRedirection();

app.Run();
```

ASP.NET Core

Rappel : Entité vs DTO

- **Entité** : modèle persistant (EF Core) : structure DB.
- **DTO** : modèle d'échange API : ce qu'on expose au client.
- En Minimal API, on conserve la même bonne pratique : **ne pas exposer directement l'entité**.

ASP.NET Core

Exemple : Entités (Models)

```
namespace CoursMinimalApi.Models;

public class Personne
{
    public int Num { get; set; }
    public string? Nom { get; set; }
    public string? Prenom { get; set; }
    public int Age { get; set; }
    public ICollection<Adresse> Adresses { get; set; } = [];
}

public class Adresse
{
    public int Id { get; set; }
    public string? Rue { get; set; }
    public string? Ville { get; set; }
    public string? CodePostal { get; set; }

    public ICollection<Personne> Personnes { get; set; } = [];

}
```

ASP.NET Core

DTO (Contracts)

```
namespace CoursWebApi.Contracts;

public record PersonneDto(
    int Num,
    string? Nom,
    string? Prenom,
    int Age
);

public record CreatePersonneDto
{
    public string Nom { get; init; }
    public string Prenom { get; init; }
    public int Age { get; init; }
}
```

ASP.NET Core

Service + Injection de dépendances

- Même principe qu'avec les controllers : on sort la logique métier hors des endpoints.
- On injecte le service directement dans le handler Minimal API.

ASP.NET Core

Interface

```
namespace CoursMinimalApi.Interfaces;

public interface IPersonneService
{
    Task<List<Personne>> GetAllAsync();
    Task<Personne?> GetByIdAsync(int id);
    Task<Personne> CreateAsync(Personne entity);
}
```

ASP.NET Core

Contenu de `PersonneService.cs`

```
namespace CoursMinimalApi.Services;

public class PersonneService : IPersonneService
{
    private readonly List<Personne> _data = [
        new Personne { Num = 1, Nom = "Wick", Prenom = "John", Age = 45 },
        new Personne { Num = 2, Nom = "Dalton", Prenom = "Jack", Age = 40 }
    ];

    public Task<List<Personne>> GetAllAsync()
        => Task.FromResult(_data);

    public Task<Personne?> GetByIdAsync(int id)
        => Task.FromResult(_data.FirstOrDefault(p => p.Num == id));

    public Task<Personne> CreateAsync(Personne entity)
    {
        var nextId = _data.Count == 0 ? 1 : _data.Max(p => p.Num) + 1;
        entity.Num = nextId;
        _data.Add(entity);
        return Task.FromResult(entity);
    }
}
```

ASP.NET Core

Déclarer le service dans `Program.cs`

```
builder.Services.AddScoped<IPersonneService, PersonneService>();
```

ASP.NET Core

GET : /api/personnes (à ajouter dans `Program.cs`)

```
app.MapGet("/api/personnes", async (IPersonneService service) =>
{
    var personnes = await service.GetAllAsync();

    var dto = personnes.Select(p => new PersonneDto(
        p.Num, p.Nom, p.Prenom, p.Age
    ));

    return Results.Ok(dto);
});
```

ASP.NET Core

GET : /api/personnes/id

```
app.MapGet("/api/personnes/{id:int}", async (int id, IPersonneService
   service) =>
{
    var p = await service.GetByIdAsync(id);
    if (p is null) return Results.NotFound();

    var dto = new PersonneDto(p.Num, p.Nom, p.Prenom, p.Age);
    return Results.Ok(dto);
});
```

ASP.NET Core

POST : /api/personnes

```
app.MapPost("/api/personnes", async (CreatePersonneDto input,
   IPersonneService service) =>
{
    var entity = new Personne
    {
        Nom = input.Nom,
        Prenom = input.Prenom,
        Age = input.Age
    };

    var created = await service.CreateAsync(entity);

    var dto = new PersonneDto(created.Num, created.Nom, created.Prenom,
       created.Age);
    return Results.Created($"/api/personnes/{dto.Num}", dto);
});
```

ASP.NET Core

Une deuxième solution

```
app.MapPost("/api/personnes", async (IPersonneService _service, [FromBody] PersonneCreateDto
   dto) =>
{
    var entity = new Personne
    {
        Nom = input.Nom,
        Prenom = input.Prenom,
        Age = input.Age
    };

    var created = await service.CreateAsync(entity);
    var dto = new PersonneDto(created.Num, created.Nom, created.Prenom, created.Age);
    return Results.CreatedAtRoute("GetById", new { id = entity.Num }, _mapper.Map<
      PersonneResponseDto>(entity));
});
```

ASP.NET Core

Une deuxième solution

```
app.MapPost("/api/personnes", async (IPersonneService _service, [FromBody] PersonneCreateDto
   dto) =>
{
    var entity = new Personne
    {
        Nom = input.Nom,
        Prenom = input.Prenom,
        Age = input.Age
    };

    var created = await service.CreateAsync(entity);
    var dto = new PersonneDto(created.Num, created.Nom, created.Prenom, created.Age);
    return Results.CreatedAtRoute("GetById", new { id = entity.Num }, _mapper.Map<
      PersonneResponseDto>(entity));
});
```


Sans oublier d'associer un nom à GET : /api/personnes/id

```
app.MapGet("/api/personnes/{id:int}", async (int id, IPersonneService service) =>
{
    var p = await service.GetByIdAsync(id);
    if (p is null) return Results.NotFound();

    var dto = new PersonneDto(p.Num, p.Nom, p.Prenom, p.Age);
    return Results.Ok(dto);
}).WithName("GetById");
```

ASP.NET Core

Validation en Minimal API

- Contrairement à `[ApiController]`, la validation n'est pas automatique par défaut.
- On peut utiliser **DataAnnotations** et valider explicitement.
- Retour d'erreurs : `Results.ValidationProblem(...)`.

ASP.NET Core

DTO avec DataAnnotations

```
using System.ComponentModel.DataAnnotations;

namespace CoursMinimalApi.Contracts;

public record CreatePersonneDto
{
    [Required]
    public string Nom { get; init; } = null!;

    [MinLength(3), MaxLength(30)]
    public string Prenom { get; init; } = null!;

    [Range(0, 150)]
    public int Age { get; init; }
}
```

ASP.NET Core

Validation explicite dans l'endpoint POST

```
using System.ComponentModel.DataAnnotations;

app.MapPost("/api/personnes", async (CreatePersonneDto input, IPersonneService service) =>
{
    var ctx = new ValidationContext(input);
    var results = new List<ValidationResult>();

    if (!Validator.TryValidateObject(input, ctx, results, validateAllProperties: true))
    {
        var errors = results
            .GroupBy(r => r.MemberNames.FirstOrDefault() ?? "")
            .ToDictionary(
                g => g.Key,
                g => g.Select(r => r.ErrorMessage ?? "Erreur").ToArray()
            );

        return Results.ValidationProblem(errors);
    }

    var entity = new Personne { Nom = input.Nom, Prenom = input.Prenom, Age = input.Age };
    var created = await service.CreateAsync(entity);

    var dto = new PersonneDto(created.Num, created.Nom, created.Prenom, created.Age);
    return Results.Created($"/api/personnes/{dto.Num}", dto);
});
```

ASP.NET Core

Exercice

Depuis un projet **Angular**, consommer l'endpoint `GET /api/personnes`.

ASP.NET Core

Activer CORS dans `Program.cs`

```
builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(policy =>
        policy.WithOrigins("http://localhost:4200")
              .AllowAnyHeader()
              .AllowAnyMethod()
              .AllowCredentials()
    );
});

var app = builder.Build();

app.UseHttpsRedirection();

app.UseCors();
```

ASP.NET Core

Dans `Models`, créons une deuxième classe POCO `Adresse`

```
namespace CoursWebApi.Models;

public class Adresse
{
    public int Id { get; set; }
    public string? Rue { get; set; }
    public string? Ville { get; set; }
    public string? CodePostal { get; set; }

    public ICollection<Personne> Personnes { get; set; } = [];
}
```

ASP.NET Core

Modifions la classe `Personne` pour que l'association avec `Adresse` soit bidirectionnelle

```
namespace CoursWebApi.Models

public class Personne
{
    public int Num { get; set; }
    public string? Nom { get; set; }
    public string? Prenom { get; set; }
    public int Age { get; set; }

    public ICollection<Adresse> Adresses { get; set; } = [];
}
```

ASP.NET Core

Modifions également le constructeur de `PersonneService`

```
public class PersonneService : IPersonneService
{
    private readonly List<Personne> _personnes;

    public PersonneService()
    {
        Adresse adresse = new() { Rue = "paradis", Ville = "Marseille", CodePostal = "13006" };
        var personne1 = new Personne { Num = 1, Nom = "Wick", Prenom = "John", Age = 45,
          Adresses = { adresse } };
        var personne2 = new Personne { Num = 2, Nom = "Dalton", Prenom = "Jack", Age = 40 };
        var personne3 = new Personne { Num = 3, Nom = "Maggio", Prenom = "Sophie", Age = 20 };
        adresse.Personnes.Add(personne1);
        _personnes = [personne1, personne2, personne3];
    }

    // + le code précédent

}
```

ASP.NET Core

Modifions également `PersonneDto`

```
using CoursWebApi.Models;

namespace CoursWebApi.Contracts;

// DTO pour lecture
public record PersonneDto(int Id, string? Nom, string? Prenom, int Age, ICollection<Adresse>
   Adresses);
```

ASP.NET Core

Mettons à jour les actions de `PersonnesController` **utilisant** `PersonneDto`

```
[HttpGet]
public async Task<ActionResult<IEnumerable<PersonneDto>>> GetAll()
{
    var list = await _service.GetAllAsync();
    var dto = list.Select(p => new PersonneDto(p.Num, p.Nom, p.Prenom, p.Age,     p.Adresses));
    return Ok(dto);
}
```

ASP.NET Core

Mettons à jour les actions de `PersonnesController` utilisant `PersonneDto`

```
[HttpGet]
public async Task<ActionResult<IEnumerable<PersonneDto>>> GetAll()
{
    var list = await _service.GetAllAsync();
    var dto = list.Select(p => new PersonneDto(p.Num, p.Nom, p.Prenom, p.Age,     p.Adresses));
    return Ok(dto);
}
```


Relancez le projet et allez à `https://localhost:<port>/api/personnes` et vérifiez l'erreur suivante

```
System.Text.Json.JsonException: A possible object cycle was detected.
```

ASP.NET Core

Remarque

En essayant de consulter la liste des personnes (avec leurs adresses respectives), on a une boucle infinie (circular reference) car l'association est désormais bidirectionnelle.

ASP.NET Core

Remarque

En essayant de consulter la liste des personnes (avec leurs adresses respectives), on a une boucle infinie (circular reference) car l'association est désormais bidirectionnelle.

Cycles : 2 stratégies

Stratégie	**Avantage**	**Limite**
`[JsonIgnore]`	contrôle fin, explicite	perte d'info / couplage sérialisation
`IgnoreCycles`	rapide, global	moins explicite, peut masquer un design

ASP.NET Core

Dans `Adresse`, décorons `Personnes` avec l'attribut `[JsonIgnore]` pour éviter sa sérialisation avec les adresses

```
using System.Text.Json.Serialization;

namespace CoursWebApi.Models;

public class Adresse
{
    public int Id { get; set; }
    public string? Rue { get; set; }
    public string? Ville { get; set; }
    public string? CodePostal { get; set; }

    [JsonIgnore]
    public ICollection<Personne> Personnes { get; set; } = [];
}
```

Relancez le projet et allez à `https://localhost:7067/api/personnes` et vérifiez le résultat suivant


```
[
  {
    "num": 1,
    "nom": "Wick",
    "prenom": "John",
    "age": 45,
    "adresses": [
      {
        "id": 0,
        "rue": "paradis",
        "ville": "Marseille",
        "codePostal": "13006"
      }
    ]
  },
  {
    "num": 2,
    "nom": "Dalton",
    "prenom": "Jack",
    "age": 40,
    "adresses": []
  },
  {
    "num": 3,
    "nom": "Maggio",
    "prenom": "Sophie",
    "age": 20,
    "adresses": []
  }
]
```

ASP.NET Core

Deuxième solution : configurer globalement `Program.cs`

```
// Pour casser les références circulaires
builder.Services.ConfigureHttpJsonOptions(options =>
{
    options.SerializerOptions.ReferenceHandler = ReferenceHandler.
       IgnoreCycles;
});
```

Relancez le projet et allez à `https://localhost:7067/api/personnes` et vérifiez le résultat suivant


```
[
  {
    "num": 1,
    "nom": "Wick",
    "prenom": "John",
    "age": 45,
    "adresses": [
      {
        "id": 0,
        "rue": "paradis",
        "ville": "Marseille",
        "codePostal": "13006",
        "personnes": [
          {
            "num": 1,
            "nom": "Wick",
            "prenom": "John",
            "age": 45,
            "adresses": null
          }
        ]
      }
    ]
  },
  {
    "num": 2,
    "nom": "Dalton",
    "prenom": "Jack",
    "age": 40,
    "adresses": []
  },
  ...
]
```

ASP.NET Core

Le problème : Mapping manuel

Le mapping manuel (DTO $\leftrightarrow$ Entité) devient répétitif et difficile à maintenir.

ASP.NET Core

Le problème : Mapping manuel

Le mapping manuel (DTO $\leftrightarrow$ Entité) devient répétitif et difficile à maintenir.

Pourquoi utiliser AutoMapper ?

- **Réduction du code boilerplate**.
- **Maintenance facilitée** : règles centralisées.
- **Évolutivité** : ajout de champs plus simple.

ASP.NET Core

Installation (CLI)

```
dotnet add package AutoMapper
```

ASP.NET Core

Définissons une classe `PersonneProfile` qui hérite de `Profile`

```
using AutoMapper;
using CoursWebApi.Contracts;
using CoursWebApi.Models;

namespace ProjetWebApiEf.Mappers;

public class PersonneProfile: Profile
{
    public PersonneProfile()
    {
        // Source => Destination
        CreateMap<PersonneCreateDto, Personne>();
    }
}
```

ASP.NET Core

Et si les propriétés ont un nom différent (`Id` et `Num` par exemple)

```
public class PersonneProfile: Profile
{
    public PersonneProfile()
    {
        // Source => Destination
         CreateMap<PersonneCreateDto, Personne>();
         CreateMap<PersonneUpdateDto, Personne>()
             .ForMember(
                dest => dest.Num,
                opt => opt.MapFrom(src => src.Id)
             );
         CreateMap<Personne, PersonneResponseDto>()
            .ForMember(
                dest=> dest.Id,
                opt => opt.MapFrom(src => src.Num)
             );
    }
}
```

ASP.NET Core

Déclarer AutoMapper dans `Program.cs`

```
builder.Services.AddAutoMapper(typeof(PersonneProfile));
```

ASP.NET Core

Utiliser AutoMapper dans un endpoint POST

```
app.MapPost("/api/personnes/am",
    async (CreatePersonneDto input, IPersonneService service, IMapper
       mapper) =>
{
    var entity = mapper.Map<Personne>(input);

    var created = await service.CreateAsync(entity);

    var dto = mapper.Map<PersonneDto>(created);

    return Results.Created($"/api/personnes/{dto.Num}", dto);
});
```