ASP.NET Core : Web 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

Plan

1. Introduction
2. Création d'un projet Web API
3. Contrôleur Web API
   - `[ApiController]`
   - `[HttpGet]`
   - `[HttpPost]`
   - `[HttpDelete]`
   - `[HttpPut]`
   - `[FromBody]`
   - `[FromQuery]`
   - `[FromRoute]`

Plan

4. Gestion d'erreurs
   - `Created()`
   - `NotFound()` et `Ok()`
   - `NoContent()`
   - `BadRequest()` et `Accepted()`
   - `[ProducesResponseType]`
   - `[ApiConventionType]`
5. Choix du format XML/JSON
   - `[FormatFilter]`
   - `[Produces("application/json")]`

Plan

6. Mise en forme des données de la réponse
7. Circular references
8. Validation de données
9. Middleware CORS

ASP.NET Core

Service web (ou **WS** pour **W**eb **S**ervice)

- Programme = { fonctionnalités exposées en temps réel et sans intervention humaine }.
- Accessible via internet ou intranet.
- Indépendant de tout système d'exploitation.
- Indépendant de tout langage de programmation.
- Utilisant un système standard d'échange (**XML** ou **JSON**), ces messages sont généralement transportés par des protocoles internet connus **HTTP** (ou autres comme **FTP**, **SMTP**...).
- Pouvant communiquer avec d'autres **WS**.

ASP.NET Core

Technologies autour de **WS**

- **HTTP** (Hypertext Transfer Protocol) : le protocole, connu, utilisé par le World Wide Web et inventé par **Roy Fiedling**.
- **REST** (Representational State Transfer) : une architecture de services Web, créée aussi par **Roy Fielding** en 2000 dans sa thèse de doctorat.
- **SOAP** (Simple object Access Protocol) : un protocole, défini par Microsoft et IBM ensuite standarisé par **W3C**, permettant la transmission de messages entre objets distants (physiquement distribués).
- **WSDL** (Web Services Description Language) : est un langage de description de service web utilisant le format **XML** (standardisé par le **W3C** depuis 2007).
- **UDDI** (Universal Description, Discovery and Integration) : un annuaire de WS.

HTTP vs **REST** (Representational State Transfer) vs **RESTful**

- Les **API REST** sont basées sur le protocole **HTTP** (architecture client/serveur) et utilisent le concept de ressource.
- Une ressource $\Rightarrow$ URL unique qui permet de l'atteindre.
- L'API REST utilise donc des méthodes suivantes pour l'échange de données client et serveur
  - `GET` pour la récupération,
  - `POST` pour l'ajout,
  - `DELETE` pour la suppression,
  - `PUT` pour la modification,
  - ...
- Plusieurs formats possibles pour les données échangées : texte, **XML**, **JSON**...
- **RESTful** : adjectif qui désigne de services Web basée sur une architecture **REST**.

ASP.NET Core

WS avec **ASP.NET**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

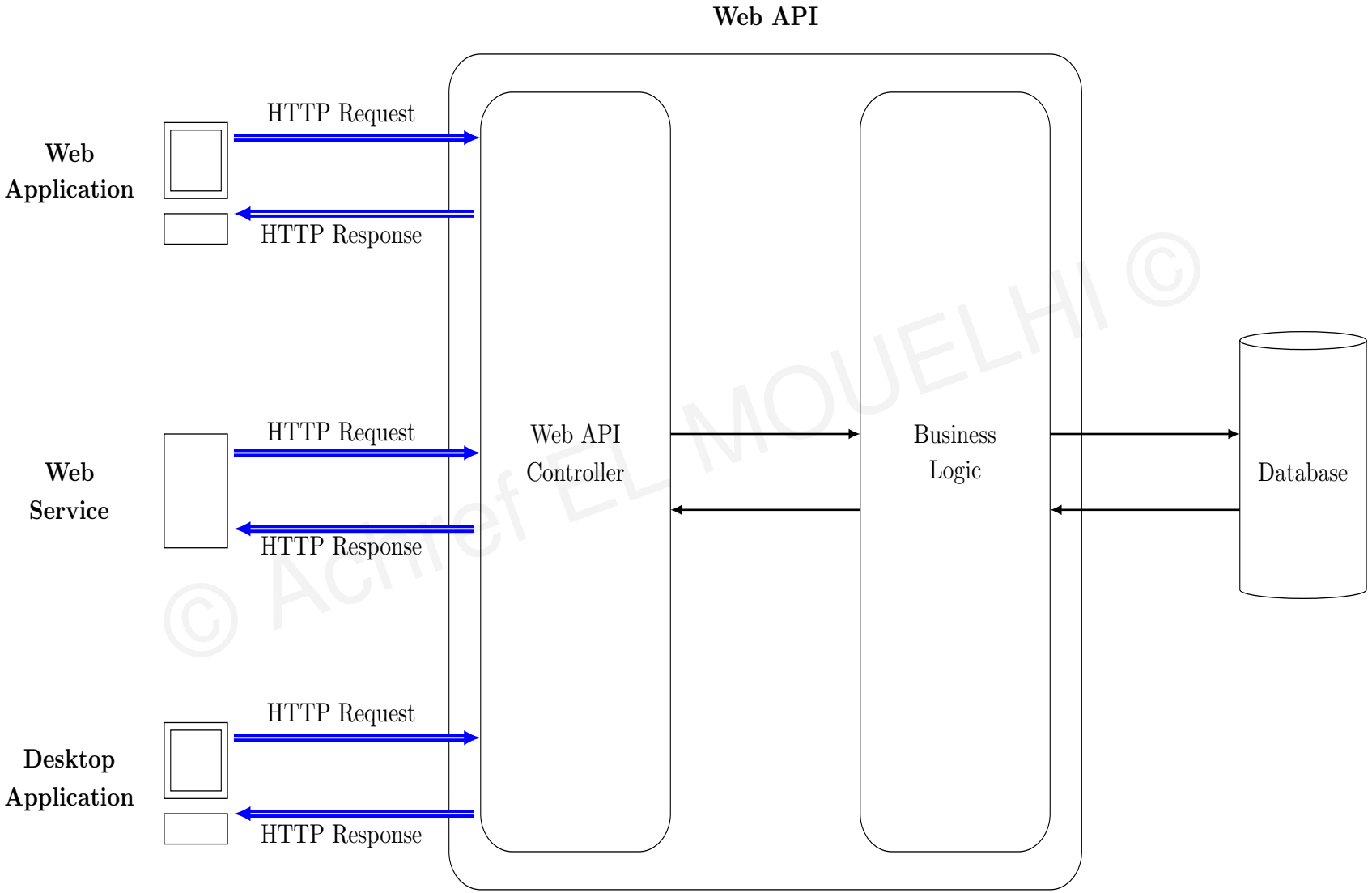
ASP.NET Core

WS avec **ASP.NET**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

Ceci est l'objet de ce chapitre.

ASP.NET Core



ASP.NET Core

Contrôleur **MVC**

- Classe qui hérite de `Controller`.
- Ses actions retournent une vue ou redirigent vers une autre action.

Contrôleur **Web API**

- Classe qui hérite de `BaseController`.
- Ses actions retournent souvent des données sous format **JSON** ou **XML**.

ASP.NET Core

Création d'un projet **Web API** avec **Visual Studio Community 2022**

- Allez dans `Fichier > Nouveau > Projet`
- Dans la zone de recherche, saisissez `web api`
- Sélectionner `API Web ASP.NET Core`
- Remplir le champs `Nom` par `CoursWebApi`
- Gardez les options par défaut
- Validez et attendre la fin de création du projet

ASP.NET Core

Structure du projet

- Une application Web API $\simeq$ une application **MVC** sans **Views**
  - `Controllers` : pour les contrôleurs
  - `Models` : pour les classes modèles
- `Connected Services` : liste des services Web consommés par l'application.
- `Properties/launchSettings.json` : fichier de configuration utilisé par **Visual Studio** au démarrage de l'application.
- `appsettings.json` : fichier de configuration pouvant contenir les variables globales, les données de connexion à la base de données...
- `Dépendances` : dossier contenant les librairies indispensables pour le démarrage de l'application.
- `Program.cs` : point d'entrée d'une application **ASP.NET Core**.

ASP.NET Core

Dans `Program.cs`, plusieurs middlewares chargés et aussi Swagger

```
using CoursWebApi.Interfaces;
using CoursWebApi.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

ASP.NET Core

Swagger UI ?

- Générateur de documentation au format **JSON** pour les services **REST**
- Permettant également de visualiser la documentation des ressources exposées
- Documentation officielle : `https://swagger.io/tools/swagger-ui/`

ASP.NET Core

Étapes

- Préparer le Modèle
- Créer le contrôleur Web API
- Envoyer de requêtes à ce contrôleur Web API

ASP.NET Core

Créons un premier POCO `Personne` dans `Models`

```
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; }
    }
}
```

ASP.NET Core

Dans `Interfaces`**, créons l'interface** `IPersonneService`

```
using CoursWebApi.Models;

namespace CoursWebApi.Interfaces
{
    public interface IPersonneService
    {
        List<Personne> GetAll();
        Personne? GetOneById(int id);
        Personne Add(Personne personne);
    }
}
```

Dans `Services`**, créons une classe** `PersonneService` **qui implémente** `IPersonneService`

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;

namespace CoursWebApi.Services
{
    public class PersonneService : IPersonneService
    {
        public List<Personne> Personnes { get; set; }
        public PersonneService()
        {
            Personnes = new List<Personne>()
            {
                new Personne(){ Num = 1, Nom = "Wick", Prenom = "John", Age = 45},
                new Personne(){ Num = 2, Nom = "Dalton", Prenom = "Jack", Age = 40},
                new Personne(){ Num = 3, Nom = "Maggio", Prenom = "Sophie", Age = 20},
            };
        }
        public List<Personne> GetAll()
        {
            return Personnes;
        }
        public Personne? GetOneById(int id)
        {
            return Personnes.Find(elt => elt.Num == id);
        }
        public Personne Add(Personne personne)
        {
            Personnes.Add(personne);
            return personne;
        }
    }
}
```

ASP.NET Core

Dans `Program.cs`, préparons l'injection de dépendance de l'interface `IPersonneService`

```
using CoursWebApi.Interfaces;
using CoursWebApi.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddSingleton<IPersonneService, PersonneService>();
var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

ASP.NET Core

Contrôleur Web API

- Un des composants du modèle **MVC**
- Une classe **C#**
  - ayant l'attribut `[ApiController]`
  - héritant de la classe `ControllerBase` : classe mère des contrôleurs **MVC** (sans support pour les vues)
- Chaque action du contrôleur doit
  - avoir comme nom un verbe **HTTP** : `Get`, `Post`, `Put`, `Delete`...
  - être décorée par `[HttpVerb]` : `Verb` à remplacer par `Get`, `Post`, `Put`, `Delete`...

ASP.NET Core

L'attribut `[ApiController]` permet d'avoir

- le routage par attribut
- les messages **HTTP** 400
- les attributs de liaison : `[FromBody]`, `[FromRoute]`...
- la validation du modèle
- ...

ASP.NET Core

Remarque

Par défaut, un contrôleur **REST** encode les données au format **JSON**.

ASP.NET Core

Remarque

Par défaut, un contrôleur **REST** encode les données au format **JSON**.

JSON : **J**ava**S**cript **O**bject **N**otation

- Objet **JavaScript**,
- Clés et valeurs de type `string` entourées par des guillemets.

ASP.NET Core

Créons le contrôleur

- Faire clic droit sur le répertoire `Controllers`
- Aller dans `Ajouter > Contrôleur`
- Dans `API`, sélectionner `Contrôleur d'API Vide`
- Cliquer sur `Ajouter`
- Renommer en `PersonnesController` puis Valider

ASP.NET Core

Code généré

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PersonnesController : ControllerBase
    {
    }
}
```

ASP.NET Core

Commençons par injecter le service `IPersonneService` dans `PersonnesController`

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PersonnesController : ControllerBase
    {
        private readonly IPersonneService personneService;
        public PersonnesController(IPersonneService personneService)
        {
            this.personneService = personneService;
        }
    }
}
```

Définissons maintenant une méthode qui nous permet de retourner tout le contenu de la liste `Personnes`

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PersonnesController : ControllerBase
    {
        private readonly IPersonneService personneService;
        public PersonnesController(IPersonneService personneService)
        {
            this.personneService = personneService;
        }
        public IEnumerable<Personne> Get()
        {
            return personneService.GetAll();
        }
    }
}
```

ASP.NET Core

Explication

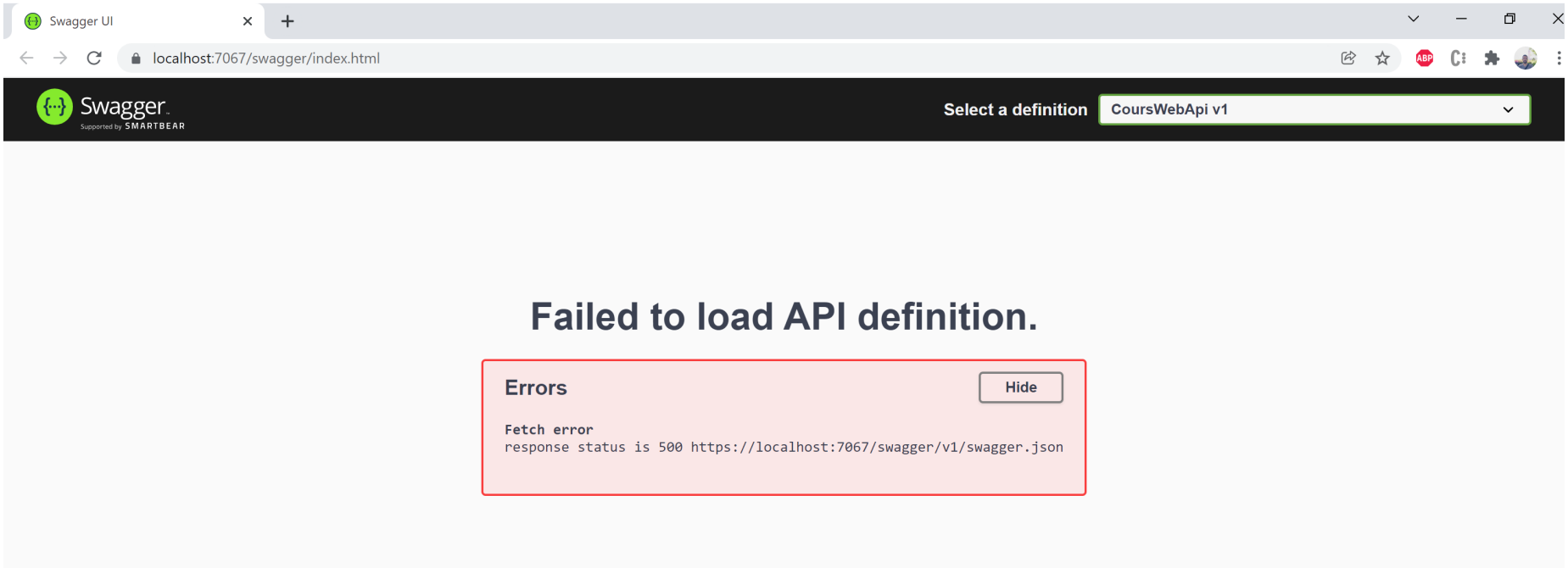
- Toutes les actions de ce contrôleur sont accessibles via la route : `localhost:$<$port$>$/api/personnes`.
- Il est possible de modifier la route pour une action donnée.
- Par défaut, une action est accessible via la méthode **HTTP** `GET`.
- Dans une application **ASP.NET Core 6**, le résultat retourné par un contrôleur **Web API** est au format **JSON**.
- Il est possible d'utiliser un autre format comme **XML**...

ASP.NET Core

Lancez le projet et vérifiez qu'en allant à `api/personnes` le résultat suivant est affiché

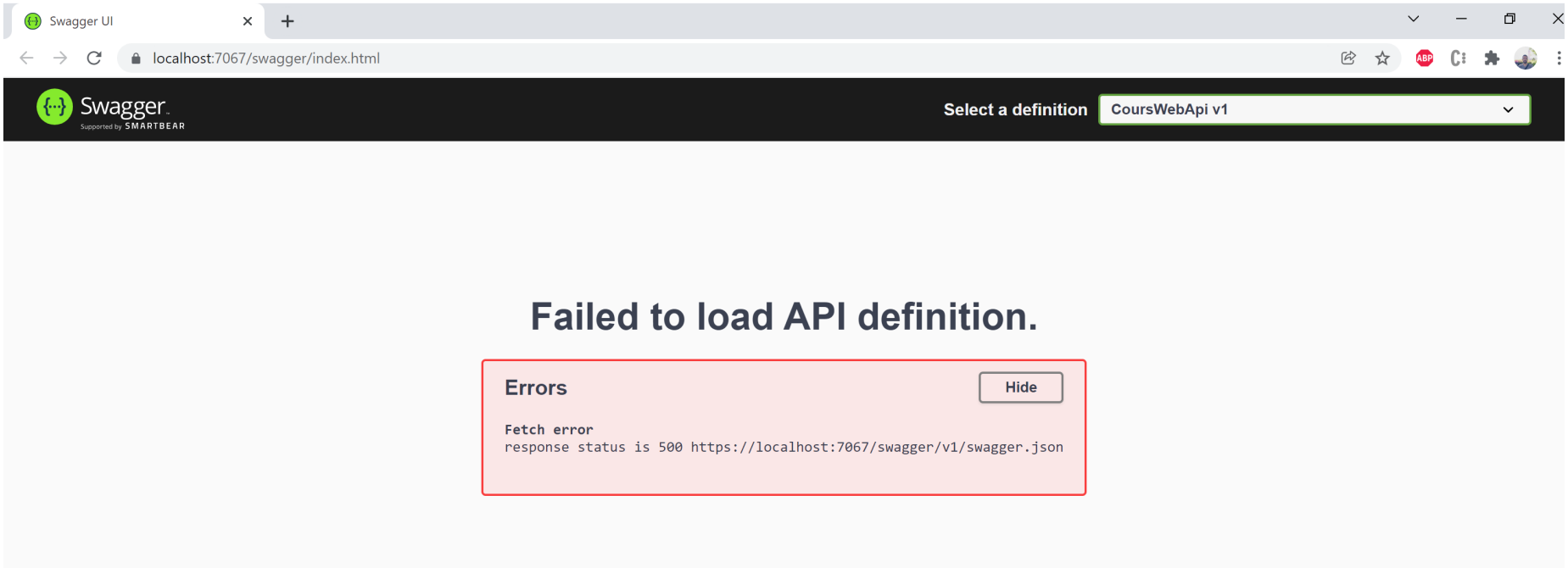

```
[
    {
        "num": 1,
        "nom": "Wick",
        "prenom": "John",
        "age": 45
    },
    {
        "num": 2,
        "nom": "Dalton",
        "prenom": "Jack",
        "age": 40
    },
    {
        "num": 3,
        "nom": "Maggio",
        "prenom": "Sophie",
        "age": 20
    }
]
```

ASP.NET Core



Problème avec Swagger

ASP.NET Core



Problème avec Swagger

Explication

Swagger a besoin qu'on précise explicite le verbe **HTTP** de cette méthode.

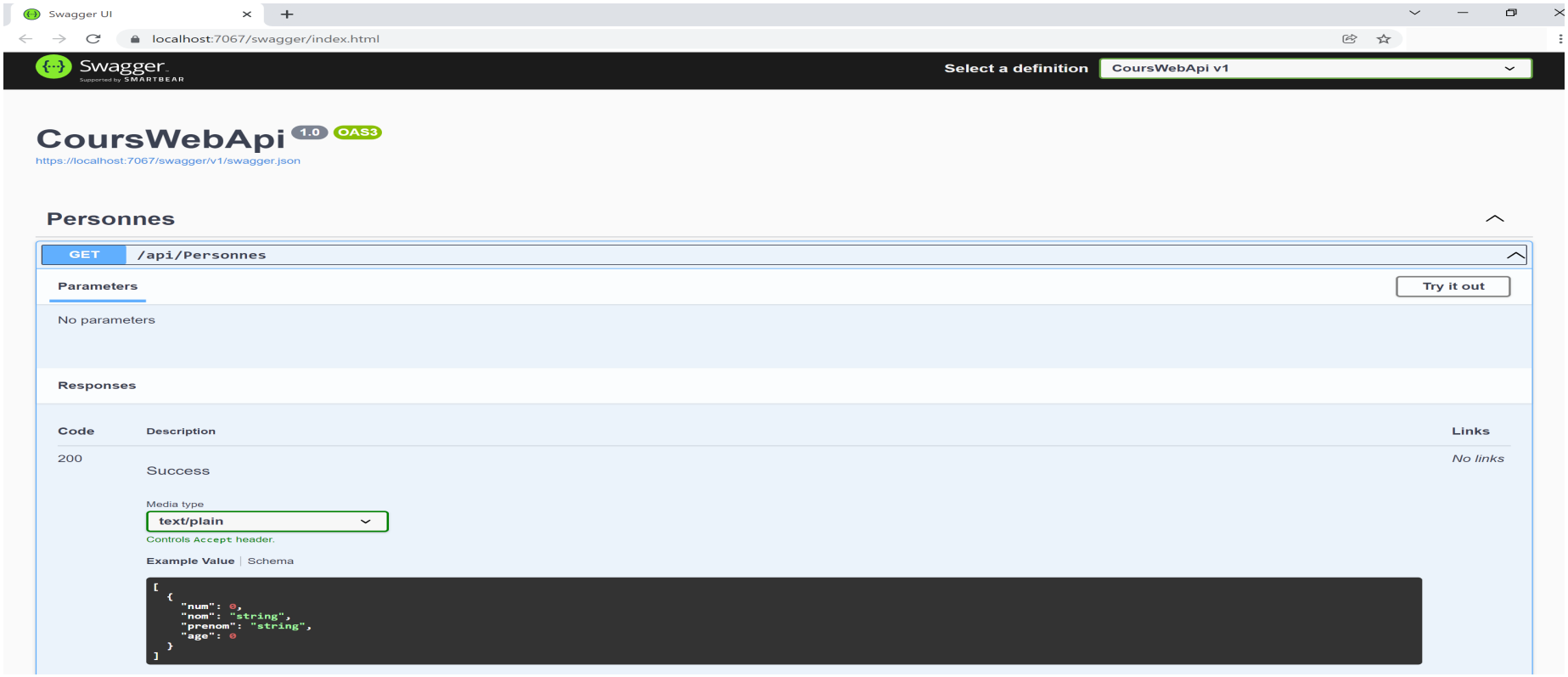
Ajoutons l'attribut [HttpGet]

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PersonnesController : ControllerBase
    {
        private readonly IPersonneService personneService;
        public PersonnesController(IPersonneService personneService)
        {
            this.personneService = personneService;
        }
        [HttpGet]
        public IEnumerable<Personne> Get()
        {
            return personneService.GetAll();
        }
    }
}
```

ASP.NET Core

Relancez le projet et vérifiez que le problème est résolu



ASP.NET Core

Ajoutons une méthode qui permet de récupérer une seule personne selon son identifiant

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PersonnesController : ControllerBase
    {
        private readonly IPersonneService personneService;
        public PersonnesController(IPersonneService personneService)
        {
            this.personneService = personneService;
        }
        [HttpGet]
        public IEnumerable<Personne> Get()
        {
            return personneService.GetAll();
        }
        [HttpGet("{id}")]
        public Personne? Get(int id)
        {
            return personneService.GetOneById(id);
        }
    }
}
```

ASP.NET Core

Pour tester, allez à `/personnes/1` et vérifiez le résultat suivant


```
{
    "num": 1,
    "nom": "Wick",
    "prenom": "John",
    "age": 45
}
```

ASP.NET Core

Définissons l'action qui permet l'ajout d'une nouvelle personnne

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PersonnesController : ControllerBase
    {
        // le code précédent +
        [HttpPost]
        public Personne? Post(Personne personne)
        {
            Console.WriteLine(personne.Nom);
            return personneService.Add(personne);
        }
    }
}
```

ASP.NET Core

Deux solutions pour tester

- **Swagger**
- Un simulateur de client **REST** comme **Postman**

ASP.NET Core

Utilisation de **Postman**

- Aller sur internet et chercher **Postman**,
- Télécharger puis installer l'application,
- Démarrer l'application.

ASP.NET Core

Utilisation de **Postman**

- Aller sur internet et chercher **Postman**,
- Télécharger puis installer l'application,
- Démarrer l'application.

Il existe plusieurs autres tel que ARC (pour Advanced REST Client)...

ASP.NET Core

Pour ajouter une nouvelle personne

- Utilisez **Postman** en précisant la méthode `POST`
- Saisissez l'URL de notre ressource `https://localhost:7067/api/personnes`
- Dans `Headers`, saisissez `Content-Type` comme `Key` et `application/json` comme `Value`
- Ensuite cliquez sur `Body`, cochez `raw`, choisissez `JSON (application/json)` et saisissez des données sous format `JSON` correspondant à l'objet personne à ajouter

```
{
    "num": 4,
    "nom": "El Mouelhi",
    "prenom": "Achref",
    "age": 37
}
```

- Cliquez sur `Send`

ASP.NET Core

Exercice 1

En se basant sur le nouveau contenu de l'interface `IPersonneService` donné dans le slide suivant :

- Implémentez les méthodes manquantes dans `PersonneService`.
- Créez puis testez, avec **Postman** ou **Swagger**, deux actions associées aux verbes **HTTP** `PUT` et `DELETE` qui permettront de modifier et supprimer une personne.
  - L'action `Put` accepte deux paramètres, l'identifiant et l'objet à modifier.
  - L'action `Delete` accepte un seul paramètre : l'identifiant de la personne à supprimer.

ASP.NET Core

Nouveau contenu de `IPersonneService`

```
using CoursWebApi.Models;

namespace CoursWebApi.Interfaces
{
    public interface IPersonneService
    {
        List<Personne> GetAll();
        Personne? GetOneById(int id);
        Personne Add(Personne personne);
        bool Delete(int id);
        Personne? Update(Personne personne);
    }
}
```

ASP.NET Core

Suppression : méthode à ajouter dans le service

```
public bool Delete(int id)
{
    return Personnes.Remove(GetOneById(id));
}
```

ASP.NET Core

Suppression : méthode à ajouter dans le service

```
public bool Delete(int id)
{
    return Personnes.Remove(GetOneById(id));
}
```

Suppression : action à ajouter dans le contrôleur

```
[HttpDelete("{id}")]
public bool Delete(int id)
{
    return personneService.Delete(id);
}
```

ASP.NET Core

Deux solutions pour tester

- Choisir le verbe `DELETE`
- Saisir l'URL : `https://localhost:7067/api/personnes/1`

ASP.NET Core

Modification : méthode à ajouter dans le service

```
public Personne? Update(Personne personne)
{
    var p = GetOneById(personne.Num);
    if (p != null)
    {
        p = personne;
        return personne;
    }
    return null;
}
```

ASP.NET Core

Modification : méthode à ajouter dans le service

```
public Personne? Update(Personne personne)
{
    var p = GetOneById(personne.Num);
    if (p != null)
    {
        p = personne;
        return personne;
    }
    return null;
}
```


Modification : action à ajouter dans le contrôleur

```
[HttpPut("{id}")]
public Personne Put(int id, Personne personne)
{
    return personneService.Update(personne);
}
```

ASP.NET Core

Pour modifier une personne

- Utilisez **Postman** en précisant la méthode `PUT`
- Saisissez l'URL de notre ressource `https://localhost:7067/api/personnes/1`
- Dans le `Headers`, saisissez `Content-Type` comme `Key` et `application/json` comme `Value`
- Ensuite cliquez sur `Body`, cochez `raw`, choisissez `JSON (application/json)` et saisissez des données sous format `JSON` correspondant à l'objet personne à ajouter

```
{
    "num": 1,
    "nom": "El Mouelhi",
    "prenom": "Achref",
    "age": 37
}
```

- Cliquez sur `Send`

ASP.NET Core

Question

Comment **API Web** fait pour savoir s'il faut rechercher les paramètres d'une action dans le body ou l'URL de la requête ?

ASP.NET Core

Question

Comment **API Web** fait pour savoir s'il faut rechercher les paramètres d'une action dans le body ou l'URL de la requête ?

Réponse

- Si le paramètre est de type primitif (`int`, `bool`, `double`...), **API Web** essaie d'obtenir la valeur à partir de l'URL de la requête **HTTP**.
- Si le paramètre est de type complexe (objet personnalisé comme par exemple : Personne), **API Web** essaie de lire la valeur à partir du corps de la requête **HTTP**.

ASP.NET Core

Remarques

- Pour récupérer un type primitif passé dans le corps de la requête **HTTP**, alors vous devez ajouter `[FromBody]` avant le paramètre de l'action du contrôleur **Web API**.
- Pour récupérer un type complexe dans l'URL, alors vous devez ajouter
  - soit `[FromUri]` (**Web API 2**)
  - soit `[FromQuery]` (**ASP.NET Core MVC**)

  avant le paramètre de l'action du contrôleur **Web API**.

ASP.NET Core

[FromRoute] vs [FromUri] et [FromQuery]

- [FromUri] et [FromQuery] permettent de récupérer les paramètres de requêtes (situés après ? écrits sous la forme clé=valeur)
- [FromRoute] permet de récupérer les variables de chemin (situés avant ?)

ASP.NET Core

Suppression : méthode à ajouter dans le service

```
[HttpDelete]
public bool Delete([FromBody] int id)
{
    return personneService.Delete(id);
}
```

ASP.NET Core

Suppression : méthode à ajouter dans le service

```
[HttpDelete]
public bool Delete([FromBody] int id)
{
    return personneService.Delete(id);
}
```


Pour tester

- Choisir le verbe `DELETE`
- Saisir l'URL : `https://localhost:7067/api/personnes`
- Dans `Body`, spécifiez l'identifiant de la personne à supprimer.

ASP.NET Core

Pour récupérer un paramètre situé après `?`, on utilise

`[FromQuery]`

```
[HttpDelete]
public bool Delete([FromQuery] int id)
{
    return personneService.Delete(id);
}
```

ASP.NET Core

Pour récupérer un paramètre situé après `?`, on utilise `[FromQuery]`

```
[HttpDelete]
public bool Delete([FromQuery] int id)
{
    return personneService.Delete(id);
}
```


Pour tester

- Choisir le verbe `DELETE`
- Saisir l'URL : `https://localhost:7067/api/personnes?id=1`

ASP.NET Core

Pour récupérer plusieurs paramètres dans un objet, on utilise `[FromRoute]`

```
[HttpPost("{Num}/{Nom}/{Prenom}/{Age}")]
public Personne? PostPersonne([FromRoute] Personne personne)
{
    Console.WriteLine(personne.Nom);
    return personneService.Add(personne);
}
```

ASP.NET Core

Pour récupérer plusieurs paramètres dans un objet, on utilise `[FromRoute]`

```
[HttpPost("{Num}/{Nom}/{Prenom}/{Age}")]
public Personne? PostPersonne([FromRoute] Personne personne)
{
    Console.WriteLine(personne.Nom);
    return personneService.Add(personne);
}
```


Pour tester

- Choisir le verbe `POST`
- Saisir l'URL : `https://localhost:7067/api/personnes/4/elmouelhi/achref/37`

ASP.NET Core

Gestion d'erreurs

- Si nous envoyons une requête **HTTP** de type **GET** à l'URL `https://localhost:7067/api/personnes/1`, nous obtiendrons un objet **JSON** contenant des informations sur la personne ayant le numéro `1`.
- Si nous demandons des informations sur une personne qui n'existe pas (par exemple numéro `10000`), nous n'obtiendrons pas de réponse mais un état **200 OK**.
- Ce type de retour peut avoir des graves conséquences à la réception.
- Nous préférons avoir un code plus approprié.

ASP.NET Core

Une action d'un contrôleur **Web API** peut retourner

- un type primitif ou objet (pas de gestion d'erreurs ni les états **HTTP**).
- un `ActionResult` si l'action retourne un seul type et donc un seul état **HTTP**.
- un `IActionResult` recommandé par **Microsoft** lorsque plusieurs `ActionResult` types de retour sont possibles dans une action..

ASP.NET Core

Modifions la méthode `POST` qui ajoute une nouvelle personne

```
[HttpPost("{Num}/{Nom}/{Prenom}/{Age}")]
public ActionResult<Personne> PostPersonne([FromRoute] Personne
    personne)
{
    personneService.Add(personne);
    return Created("/personnes", personne);
}
```

ASP.NET Core

Commençons par modifier la méthode `Get` qui retourne une personne selon l'identifiant

```
[HttpGet("{id}")]
public IActionResult Get(int id)
{
    var personne = personneService.GetOneById(id);
    if (personne == null)
    {
        return NotFound();
    }
    return Ok(personne);
}
```

ASP.NET Core

Modifier la méthode `Delete` qui supprime une personne selon l'identifiant

```
[HttpDelete]
public IActionResult Delete([FromQuery] int id)
{
    if (!personneService.Delete(id))
    {
        return NotFound();
    }
    return NoContent();
}
```

ASP.NET Core

Modifions la méthode `Put` qui modifie une personne selon l'identifiant

```
[HttpPut("{id}")]
public IActionResult Put(int id, Personne personne)
{
    if (id != personne.Num)
    {
        return BadRequest();
    }
    if (personneService.Update(personne) == null)
    {
        return NotFound();
    }
    return Accepted();
}
```

ASP.NET Core

On peut aussi utiliser l'attribut `[ProducesResponseType]`

```
[HttpPost("{Num}/{Nom}/{Prenom}/{Age}")]
[ProducesResponseType(typeof(Personne), StatusCodes.Status201Created)]
public Personne Post([FromRoute] Personne personne)
{
    return personneService.Add(personne);
}
```

ASP.NET Core

On peut aussi utiliser l'attribut `[ProducesResponseType]`

```
[HttpPost("{Num}/{Nom}/{Prenom}/{Age}")]
[ProducesResponseType(typeof(Personne), StatusCodes.Status201Created)]
public Personne Post([FromRoute] Personne personne)
{
    return personneService.Add(personne);
}
```

Ou aussi

```
[HttpPost("{Num}/{Nom}/{Prenom}/{Age}")]
[ProducesResponseType(StatusCodes.Status201Created)]
public Personne Post([FromRoute] Personne personne)
{
    return personneService.Add(personne);
}
```

ASP.NET Core

Remarque

L'utilisation de l'attribut `[ProducesResponseType]` permet de l'intégrer dans la documentation de **Swagger**

ASP.NET Core

En consultant la documentation Swagger de cette action, le seul retour possible est `200 - success`

```
[HttpDelete]
public IActionResult Delete([FromQuery] int id)
{
    if (!personneService.Delete(id))
    {
        return NotFound();
    }
    return NoContent();
}
```

ASP.NET Core

En consultant la documentation Swagger de cette action, le seul retour possible est `200 - success`

```
[HttpDelete]
public IActionResult Delete([FromQuery] int id)
{
    if (!personneService.Delete(id))
    {
        return NotFound();
    }
    return NoContent();
}
```

Ainsi, on voit, dans la documentation, les deux codes possibles : `204` **et** `404`

```
[HttpDelete]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status204NoContent)]
public IActionResult Delete([FromQuery] int id)
{
    if (!personneService.Delete(id))
    {
        return NotFound();
    }
    return NoContent();
}
```

ASP.NET Core

[ApiConventionType(typeof(DefaultApiConventions))]

- **ASP.NET Core** définit un ensemble de conventions pour les actions d'un contrôleur.
- Objectif
  - éviter les multi-décorations des actions individuelles avec `[ProducesResponseType]`,
  - faciliter l'extraction de la documentation.
- Pour cela, on peut utiliser
  - `[ApiConventionType]`, ou
  - `[ApiConventionMethod]`.

ASP.NET Core

Pour appliquer les conventions sur toutes les actions d'un contrôleur, on ajoute `[ApiConventionType(typeof(DefaultApiConventions))]`

```
namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [ApiConventionType(typeof(DefaultApiConventions))]
    public class PersonnesController : ControllerBase
```

ASP.NET Core

Ainsi, comme si on a ajouté les trois `[ProducesResponseType]` **suivants à l'action** `Put`

```
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[HttpPut("{id}")]
public IActionResult Put(int id, Personne personne)
{
    if (id != personne.Num)
    {
        return BadRequest();
    }
    if (personneService.Update(personne) == null)
    {
        return NotFound();
    }
    return NoContent();
}
```

ASP.NET Core

Remarque

Si, malgré la présence de l'attribut `[ApiConventionType(typeof(DefaultApiConventions))]` sur le contrôleur, on ajoute des `[ProducesResponseType]` sur une action, alors c'est les `[ProducesResponseType]` qui seront inclus dans la documentation de l'action.

ASP.NET Core

Pour appliquer les conventions sur une action particulière (`Put` par exemple), on utilise `[ApiConventionMethod]`

```
[ApiConventionMethod(typeof(DefaultApiConventions),nameof(
   DefaultApiConventions.Put))]
[HttpPut("{id}")]
public IActionResult Put(int id, Personne personne)
{
    if (id != personne.Num)
    {
        return BadRequest();
    }
    if (personneService.Update(personne) == null)
    {
        return NotFound();
    }
    return NoContent();
}
```

ASP.NET Core

Dans `Program.js`, ajoutons la ligne suivant pour activer la génération de données au format XML

```
using CoursWebApi.Interfaces;
using CoursWebApi.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddSingleton<IPersonneService, PersonneService>();

// Pour activer la sérialisation XML
builder.Services.AddControllers().AddXmlSerializerFormatters();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

ASP.NET Core

Pour récupérer les données au format **XML**

- Utilisez **Postman** en précisant la méthode `GET`
- Saisissez l'URL de notre ressource `https://localhost:7067/api/personnes`
- Dans le `Headers`, saisissez `Accept` comme `Key` et `application/xml` comme `Value`
- Cliquez sur `Send`

ASP.NET Core

Résultat

```
<ArrayOfPersonne xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <Personne>
        <Num>1</Num>
        <Nom>Wick</Nom>
        <Prenom>John</Prenom>
        <Age>45</Age>
    </Personne>
    <Personne>
        <Num>2</Num>
        <Nom>Dalton</Nom>
        <Prenom>Jack</Prenom>
        <Age>40</Age>
    </Personne>
    <Personne>
        <Num>2</Num>
        <Nom>Maggio</Nom>
        <Prenom>Sophie</Prenom>
        <Age>20</Age>
    </Personne>
</ArrayOfPersonne>
```

ASP.NET Core

Remarque

Il est aussi possible de spécifier le format souhaité dans l'URL de la requête **HTTP**.

ASP.NET Core

Commençons par décorer le contrôleur par l'attribut `[FormatFilter]`

```
namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [FormatFilter]
    public class PersonnesController : ControllerBase
```

ASP.NET Core

Commençons par décorer le contrôleur par l'attribut `[FormatFilter]`

```
namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [FormatFilter]
    public class PersonnesController : ControllerBase
```

Modifions une action du contrôleur en ajoutant un paramètre `format`

```
[HttpGet("{id}.{format?}")]
public IActionResult Get(int id)
{
    var personne = personneService.GetOneById(id);
    if (personne == null)
    {
        return NotFound();
    }
    return Ok(personne);
}
```

ASP.NET Core

En allant à `/personnes/1`

```
{
    "num": 1,
    "nom": "Wick",
    "prenom": "John",
    "age": 45
}
```

En allant à `/personnes/1.json`

```
{
    "num": 1,
    "nom": "Wick",
    "prenom": "John",
    "age": 45
}
```

En allant à `/personnes/1.xml`

```
<Personne xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
  xsd="http://www.w3.org/2001/XMLSchema">
    <Num>1</Num>
    <Nom>Wick</Nom>
    <Prenom>John</Prenom>
    <Age>45</Age>
</Personne>
```

ASP.NET Core

[Produces("application/json")]

- Permet de forcer une ou toutes les actions du contrôleur à retourner des réponses au format **JSON**.
- Peut être appliqué sur un contrôleur ou une action.

ASP.NET Core

Ainsi toutes les actions retourneront uniquement des données au format JSON

```
namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [FormatFilter]
    [Produces("application/json")]
    public class PersonnesController : ControllerBase
```

ASP.NET Core

Remarques

- En **C#**, les propriétés sont écrites en **PascalCase**.
- Par défaut, Le sérialiseur **JSON** est configuré en **CamelCase**.
- Il est possible de mettre les clés de la réponse **JSON** d en **Pascal-Case**.

ASP.NET Core

Dans `Program.cs`, ajoutons la ligne suivant pour mettre les clés de la réponse JSON d en PascalCase

```
using CoursWebApi.Interfaces;
using CoursWebApi.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddSingleton<IPersonneService, PersonneService>();
builder.Services.AddControllers().AddXmlSerializerFormatters();
builder.Services.AddControllers()
            .AddJsonOptions(options =>
               options.JsonSerializerOptions.PropertyNamingPolicy = null);
var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

ASP.NET Core

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


```
[
  {
    "Num": 1,
    "Nom": "Wick",
    "Prenom": "John",
    "Age": 45
  },
  {
    "Num": 2,
    "Nom": "Dalton",
    "Prenom": "Jack",
    "Age": 40
  },
  {
    "Num": 3,
    "Nom": "Maggio",
    "Prenom": "Sophie",
    "Age": 20
  }
]
```

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; }

        public Adresse()
        {
            Personnes = new List<Personne>();
        }
    }
}
```

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; }

        public Personne()
        {
            Adresses = new List<Adresse>();
        }
    }
}
```

ASP.NET Core

Modifions également le constructeur de `PersonneService`

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;

namespace CoursWebApi.Services
{
    public class PersonneService : IPersonneService
    {
        public List<Personne> Personnes { get; set; }
        public PersonneService()
        {
            Adresse adresse = new Adresse() { 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 = new List<Personne>() { personne1, personne2, personne3 };
        }

        // + le code précédent

    }
}
```

ASP.NET Core

Remarque

Rien à changer dans le contrôleur.

ASP.NET Core

Remarque

Rien à changer dans le contrôleur.

Relancez le projet et allez à `https://localhost:7067/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 de 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 de personnes (avec leurs adresses respectives), on a une boucle infinie (circular reference) car l'association est désormais bidirectionnelle.

Solution

Pour arrêter la boucle infinie, on peut ajouter l'attribut `[JsonIgnore]` dans la classe `Adresse`.

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; }
        public Adresse()
        {
            Personnes = new List<Personne>();
        }
    }
}
```

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

Exercice

Dans `PersonnesController`, créer deux actions accessible via la méthode **HTTP** `GET`

- `personnes/{IdPersonne}/adresses` retourne la liste d'adresses de la personne ayant l'identifiant `IdPersonne` (on ne retourne pas les autres propriétés `nom`, `prenom`...).
- `personnes/{IdPersonne}/adresses/{IdAdresse}` retourne l'adresse ayant l'identifiant `IdAdresse` de la personne ayant l'identifiant `IdPersonne`.

ASP.NET Core

Validation de données

- Le typage de données (attributs, propriétés) ne suffira pas pour avoir des données cohérentes.
- **ASP.NET** nous propose plusieurs décorateurs pour préciser les contraintes nécessaires pour avoir des données valides.

ASP.NET Core

Dans `Personne`, annotons quelques propriétés avec les décorateurs de validation suivants

```
using System.ComponentModel.DataAnnotations;

namespace CoursWebApi.Models
{
    public class Personne
    {
        public int Num { get; set; }
        [Required]
        public string? Nom { get; set; }
        [StringLength(20)]
        public string? Prenom { get; set; }
        [Range(0, 150)]
        public int Age { get; set; }
        public ICollection<Adresse> Adresses { get; set; }
        public Personne()
        {
            Adresses = new List<Adresse>();
        }
    }
}
```

Pour ajouter une nouvelle personne

- Utilisez **Postman** en précisant la méthode `POST` et en saisissant l'URL `https://localhost:7067/api/personnes`
- Utilisez l'objet suivant dans `Body`

```
{
    "nom": "Messi",
    "prenom": "Lionel",
    "age": 160,
    "adresses": [
        {
            "rue": "austin",
            "ville": "Paris",
            "codePostal": "75000"
        }
    ]
}
```

- Cliquez sur `Send`

ASP.NET Core

Le message d'erreur relatif à la contrainte de validation de la propriété `Age`


```
{
  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "traceId": "00-392df09d12dde2fb4633f9aac83a83f7-e282a9d3a41b4723-00",
  "errors": {
    "Age": [
      "The field Age must be between 0 and 150."
    ]
  }
}
```

ASP.NET Core

Remarque

Dans un contrôleur **Web API**, il n'est pas nécessaire de vérifier explicitement si l'état du modèle est Valide (avec `if (ModelState.IsValid)`). Comme le contrôleur est décoré avec `[ApiController]`, il se charge de vérifier si le modèle est valide et renvoie automatiquement le code 400 avec les erreurs de validation.

ASP.NET Core

Liste complète de tous les décorateurs de validation

`https://docs.microsoft.com/fr-fr/dotnet/api/system.componentmodel.dataannotations?view=net-6.0`

ASP.NET Core

Exercice

Depuis un projet **Angular**, créer des composants et des services qui permettent de consommer l'action `GET` de `PersonnesController`.

ASP.NET Core

En envoyant une requête HTTP de type `GET` depuis Angular, le message d'erreur suivant s'affiche dans la console du navigateur

ASP.NET Core

CORS : **C**ross-**o**rigin **r**esource **s**haring

- C'est une norme **W3C**.
- Ce n'est pas une fonctionnalité de sécurité, il assouplit la sécurité.
- Permet à un serveur de filtrer explicitement certaines demandes Cross-Origin (origine croisée ou différente).

Dans `Program.cs`, utilisons le middleware `UseCors` et précisons ce que l'on souhaite autoriser

```
using CoursWebApi.Interfaces;
using CoursWebApi.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddSingleton<IPersonneService, PersonneService>();
builder.Services.AddControllers();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseCors(options =>
{
    options.WithOrigins("http://localhost:4200")
    .AllowAnyMethod()
    .AllowAnyHeader()
    .AllowCredentials();
});
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Une autre solution consiste à préparer un service qui précise ce que l'on souhaite accepter

```
using CoursWebApi.Interfaces;
using CoursWebApi.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddSingleton<IPersonneService, PersonneService>();
builder.Services.AddControllers();

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "CorsPolicy",
      builder => builder.WithOrigins("http://localhost:4200")
      .AllowAnyMethod()
      .AllowAnyHeader()
      .AllowCredentials());
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Ensuite, chargeons les propriétés dans le middleware `UseCors`

```
using CoursWebApi.Interfaces;
using CoursWebApi.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddSingleton<IPersonneService, PersonneService>();
builder.Services.AddControllers();

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "CorsPolicy",
      builder => builder.WithOrigins("http://localhost:4200")
      .AllowAnyMethod()
      .AllowAnyHeader()
      .AllowCredentials());
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseCors("CorsPolicy");
app.UseAuthorization();
app.MapControllers();
app.Run();
```