

Nous allons réaliser une premier API REST simple sans état (i.e. sans base de données). Cette API va nous permettre de convertir des euros en différentes devises.

Un WS REST est orienté ressources (et non traitement) par défaut (i.e. obtention de données au format JSON, XML, ATOM, etc.). Il s'agit donc ici de gérer des listes de devises avec leur taux de conversion. Cette API sera ensuite testée (tests unitaires).

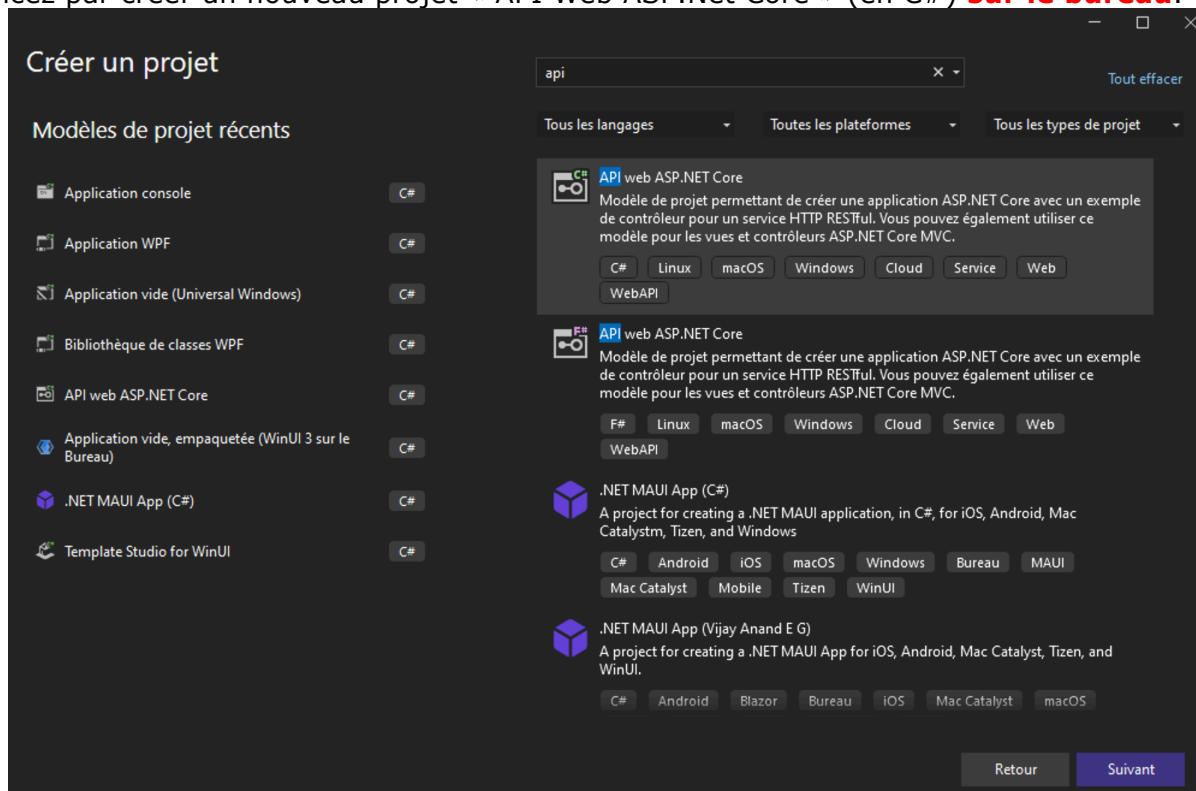
Une application cliente simple (WinUI) permettant de convertir un montant saisi en euros vers la devise sélectionnée (la liste de devises sera alimentée par le WS) sera réalisée en partie 2.

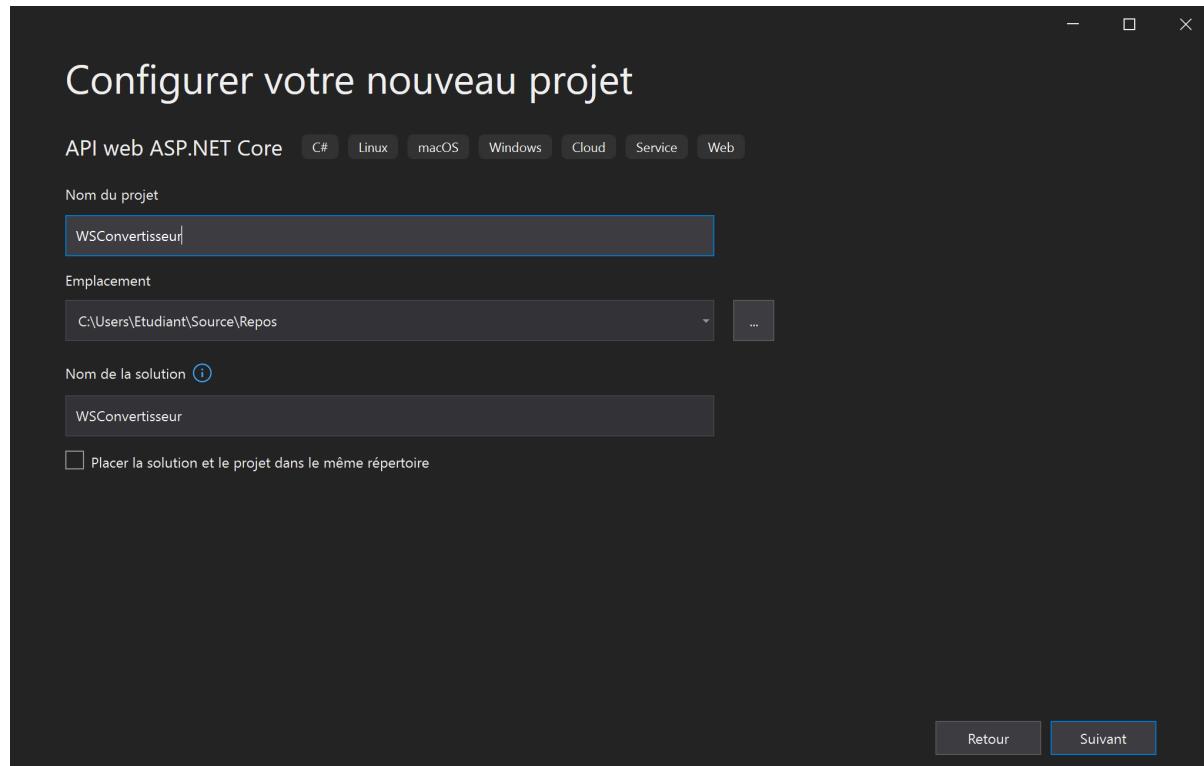
### 1. Création du WS REST

#### 1.1. Création du projet

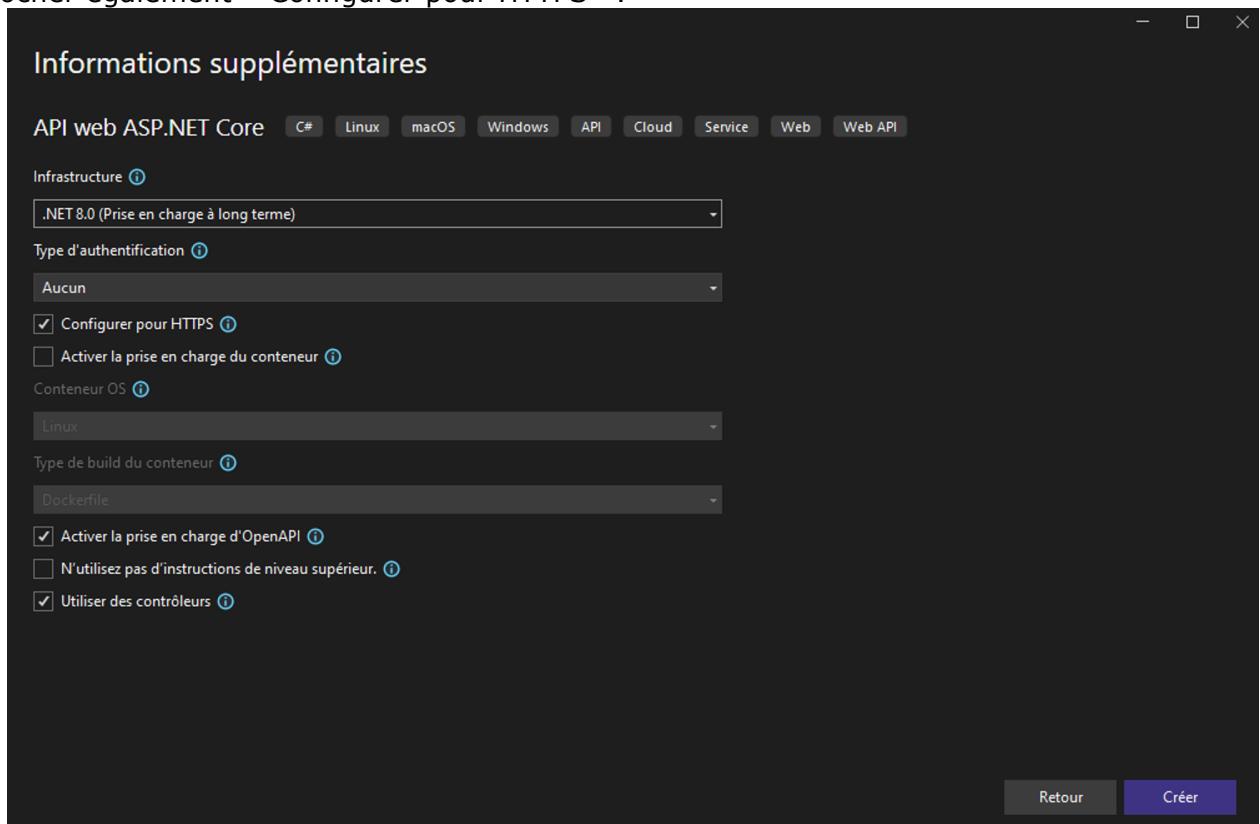
Lancez Visual Studio 2022.

Commencez par créer un nouveau projet « API Web ASP.NET Core » (en C#) **sur le bureau**.





Utiliser le framework **.NET 8** (version LTS). Bien cocher « Activer la prise en charge d'OpenAPI » qui permet de générer la documentation et tester l'API via Swagger :  
Plus d'informations ici : <https://docs.microsoft.com/fr-fr/aspnet/core/tutorials/web-api-help-pages-using-swagger>.  
Bien cocher également « Configurer pour HTTPS ».



Valider.

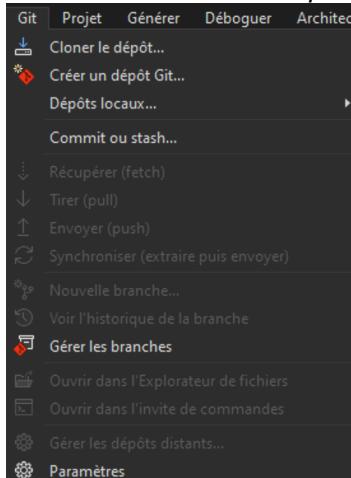
Remarque : si vous décochez l'option « Utiliser des contrôleurs », vous générerez une API REST light, i.e. consommant très peu de ressources. Les API minimales sont notamment utiles si l'application est hébergée sur un Raspberry ou autre.

## 1.2. Versioning

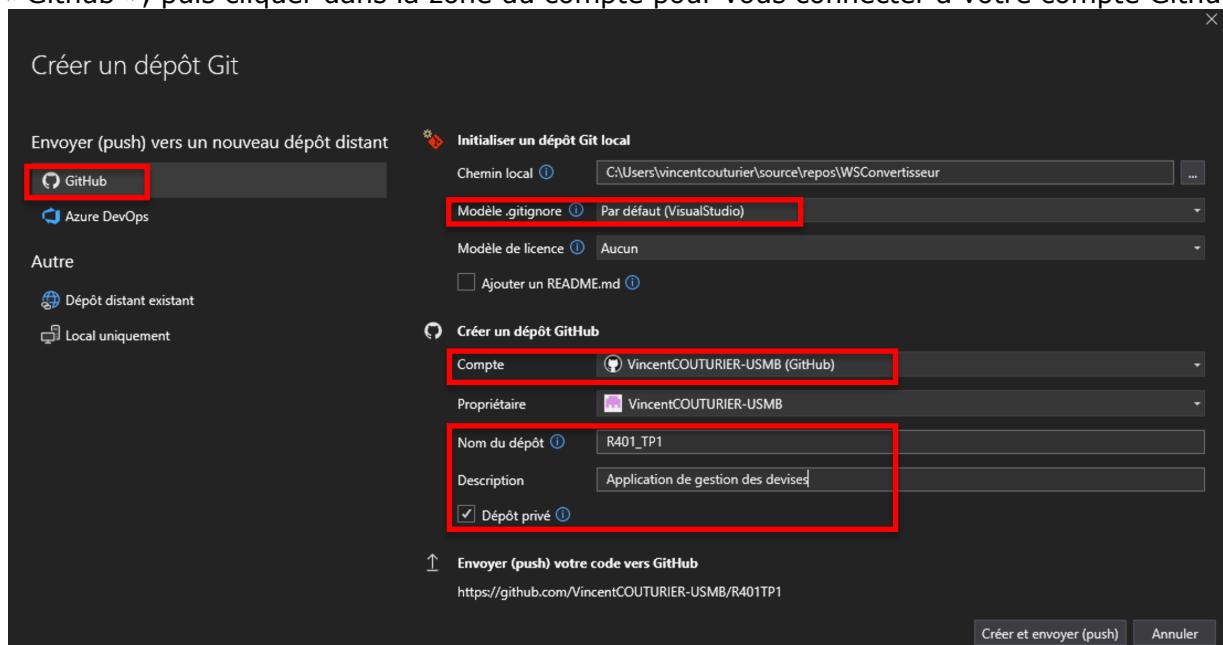
## **Création du compte Github et du dépôt**

Aller sur Github.com et vous créer un compte si vous n'en avez pas encore. Utiliser votre mail @etu.univ-smb.fr

Sous Visual Studio, dans le menu *Git*, sélectionner *Créer un dépôt Git* (dépôt = repository).



Choisir « Github », puis cliquer dans la zone du compte pour vous connecter à votre compte Github.



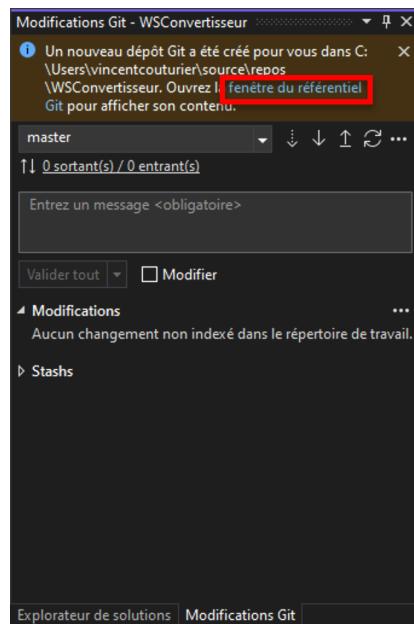
Saisir un nom de dépôt (champ « Nom du dépôt ») et, si vous le souhaitez, une phrase de description (champ « Description »).

Vous pouvez cocher la case « Ajouter un README.md » afin d'ajouter un fichier permettant de présenter le projet.

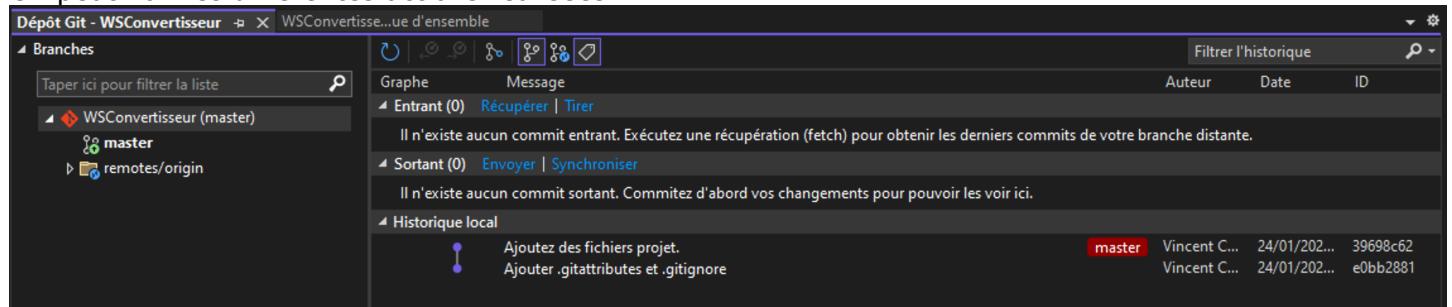
Cocher « Dépôt privé » afin qu'il ne soit pas visible par tout le monde sur Internet (connexion obligatoire). **« Modèle .gitignore » : bien laisser « Par défaut (VisualStudio) » afin de ne pas versionner certains fichiers liés à Visual Studio.**

Cliquer sur le bouton « Créer et envoyer » pour créer le dépôt et y ajouter l'application.

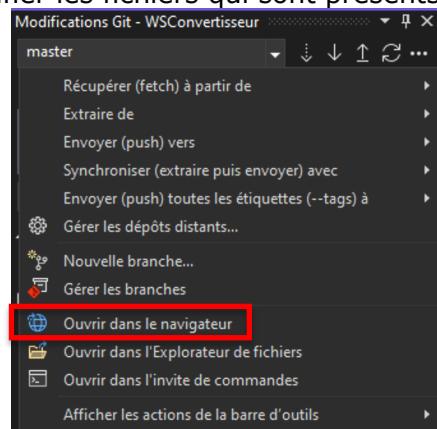
Dans la fenêtre « Modifications Git », cliquer sur le lien « Fenêtre du référentiel Git ».



On peut voir les différentes actions réalisées.



Aller sur votre compte Github et vérifier les fichiers qui sont présents dans le repository (dépôt) créé :



Les fichiers sont bien sûrs identiques à ceux du projet C# dans Visual Studio.

On peut voir les commits, i.e. les validations des modifications réalisées.

La branche « master » est la branche par défaut. C'est sur celle-ci que nous allons envoyer (push) nos modifications de code.

The screenshot shows a GitHub repository page for 'VincentCOUTURIER-USMB / R401\_TP1'. The 'Code' tab is selected. At the top, it shows 'master', '1 branch', '0 tags', 'Go to file', 'Add file', and a green 'Code' button. Below this is a list of commits:

- 0d03c92 2 minutes ago (highlighted with a red box) - VincentCOUTURIER-USMB Ajoutez des fichiers projet.
- 2 minutes ago - WSConvertisseur Ajoutez des fichiers projet.
- 2 minutes ago - .gitattributes Ajouter .gitattributes et .gitignore
- 2 minutes ago - .gitignore Ajouter .gitattributes et .gitignore
- 2 minutes ago - WSConvertisseur.sln Ajoutez des fichiers projet.

At the bottom, there's a blue box with the text 'Add a README with an overview of your project.' and a green 'Add a README' button.

The second screenshot shows the same repository page with a different view of the commit history. It highlights a specific commit from January 24, 2023:

- Ajoutez des fichiers projet. (commit 0d03c92 by VincentCOUTURIER-USMB 3 minutes ago)
- Ajouter .gitattributes et .gitignore (commit a3b2cdb by VincentCOUTURIER-USMB 3 minutes ago)

Below the commits are 'Newer' and 'Older' buttons.

*Les commits constituent les piliers d'une chronologie de projet Git. Les commits peuvent être considérés comme des instantanés ou des étapes importantes dans la chronologie d'un projet Git. Ils sont créés grâce à la commande `git commit` pour capturer l'état d'un projet à un point dans le temps.*

*Les instantanés commités peuvent être considérés comme des versions « sûres » d'un projet. Avant d'exécuter `git commit`, la commande `git add` est utilisée pour promouvoir ou « stager » les changements apportés au projet qui seront stockés dans un commit.*

*Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne.*

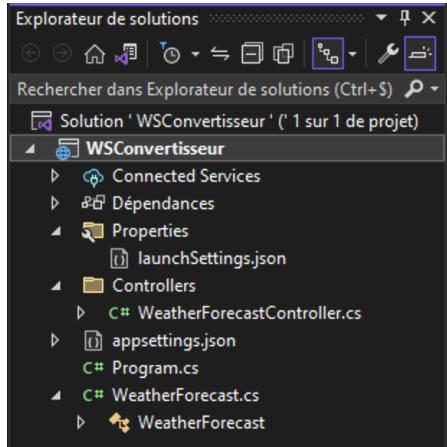
*La branche par défaut dans Git s'appelle master. Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés. À chaque validation, le pointeur de la branche master avance automatiquement.*

### 1.3. Explication des fichiers

L'architecture est basée sur un modèle MVC (Modèle-Vue-Contrôleur). Vous y retrouverez donc des contrôleurs (le dossier est déjà créé) et des modèles (que nous allons créer dans un dossier spécifique). Il n'y aura cependant pas de vues (view) car la vue correspondra à l'application cliente qui accèdera au WS.

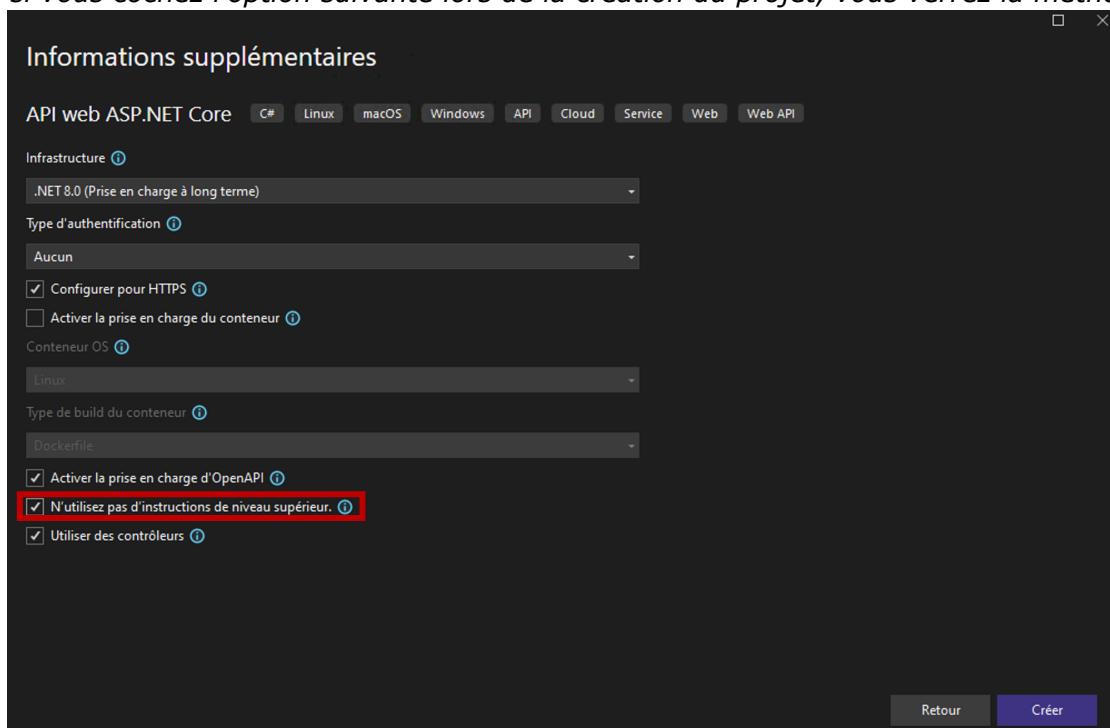
#### **Fichiers générés :**

Principaux fichiers :



- `Program.cs` : ce fichier contient la méthode `Main` (maintenant cachée) contenant toute la logique de bootstrapping.
- `WeatherForecastController.cs` : le contrôleur d'API par défaut (exemple d'application) utilisant la classe métier `WeatherForecast.cs`

*Remarque : si vous cochez l'option suivante lors de la création du projet, vous verrez la méthode `Main()` :*



```

1  namespace WSconvert2
2  {
3      public class Program
4      {
5          public static void Main(string[] args)
6          {
7              var builder = WebApplication.CreateBuilder(args);
8
9              // Add services to the container.
10             builder.Services.AddControllers();
11             // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
12             builder.Services.AddEndpointsApiExplorer();
13             builder.Services.AddSwaggerGen();
14
15             var app = builder.Build();
16
17             // Configure the HTTP request pipeline.
18             if (app.Environment.IsDevelopment())
19             {
20                 app.UseSwagger();
21                 app.UseSwaggerUI();
22             }
23
24             app.UseHttpsRedirection();
25
26             app.UseAuthorization();
27
28             app.MapControllers();
29
30             app.Run();
31
32         }
33     }
34 }
35

```

Program.cs enregistre les services nécessaires à l'API, tels que le service de prise en charge des contrôleur et des fonctionnalités liées à l'API (builder.Services.AddControllers() : nouvelle méthode d'inscription de service MVC sans vue de .NET Core) :

```

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

```

Ces services sont ajoutés au **conteneur d'injection de dépendance** (DI).

Comme nous avons coché l'option « Activer la prise en charge d'OpenAPI », le service Swagger de description et de test du WS est également ajouté au conteneur de DI :

```
builder.Services.AddSwaggerGen();
```

Rappel : l'injection de dépendance correspond au D de SOLID, que tout développeur de programmation objet doit connaître.

Pour plus de détails :

- Injection de dépendance : <https://medium.com/ividata-link/c-linjection-de-dépendances-di-et-l-inversion-de-contrôle-ioc-48dbe76cff6b>
- SOLID : [https://fr.wikipedia.org/wiki/SOLID\\_\(informatique\)](https://fr.wikipedia.org/wiki/SOLID_(informatique))

La gestion de l'authentification (`UseAuthorization`) et la redirection automatique vers le protocole HTTPS `UseHttpsRedirection` (car nous avons laissé cochée l'option  Configurer pour HTTPS) sont ensuite ajoutées au pipeline.

Exécuter le web service avec **http** pour le tester Debug ▾ Any CPU ▾ ▶ http ▾.

Par défaut la documentation de l'API s'affiche (interface Swagger). Il suffit alors de cliquer sur la méthode GET puis sur les boutons « Try it out » et « Execute » pour l'exécuter.

The screenshot shows the Swagger UI interface for the 'WSConvertisseur v1' API. At the top, there's a navigation bar with tabs for 'Swagger UI' and 'localhost:44356/swagger/index.html'. Below the navigation is a header with the 'Swagger' logo and 'Supported by SMARTBEAR'. A dropdown menu 'Select a definition' is set to 'WSConvertisseur v1'. The main content area displays the 'WeatherForecast' endpoint. It shows a 'GET /WeatherForecast' button. Below this, under the 'Schemas' section, the JSON schema for 'WeatherForecast' is displayed:

```
WeatherForecast <pre>v1 OAS3</pre>
{
    date: string($date-time)
    temperatureC: integer($int32)
    temperatureF: integer($int32)
    summary: string
    readOnly: true
    nullable: true
}
```

Un ensemble (tableau) d'enregistrements JSON est affiché en guise de résultat :

## WSConvertisseur 1.0 OAS3

https://localhost:7223/swagger/v1/swagger.json

### WeatherForecast

GET /WeatherForecast

Parameters

No parameters

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7223/WeatherForecast' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7223/WeatherForecast
```

Server response

Code Details

200 Response body

```
[{"date": "2022-09-22T14:51:09.0026557+02:00", "temperatureC": -2, "temperatureF": 29, "summary": "Mild"}, {"date": "2022-09-23T14:51:09.0029818+02:00", "temperatureC": 34, "temperatureF": 93, "summary": "Scorching"}, {"date": "2022-09-24T14:51:09.0029841+02:00", "temperatureC": 47, "temperatureF": 116, "summary": "Cool"}, {"date": "2022-09-25T14:51:09.0029845+02:00", "temperatureC": -16, "temperatureF": 4, "summary": "Mild"}, {"date": "2022-09-26T14:51:09.0029847+02:00", "temperatureC": 26, "temperatureF": 78, "summary": "Mild"}]
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Wed, 21 Sep 2022 12:51:08 GMT
server: Kestrel
```

Responses

Tester l'exécution en **https** Valider le certificat puis l'installer.

Approuver le certificat SSL ASP.NET Core

Ce projet est configuré pour utiliser SSL. Pour éviter les avertissements SSL dans le navigateur, vous pouvez choisir d'approuver le certificat auto-signé généré par ASP.NET Core.

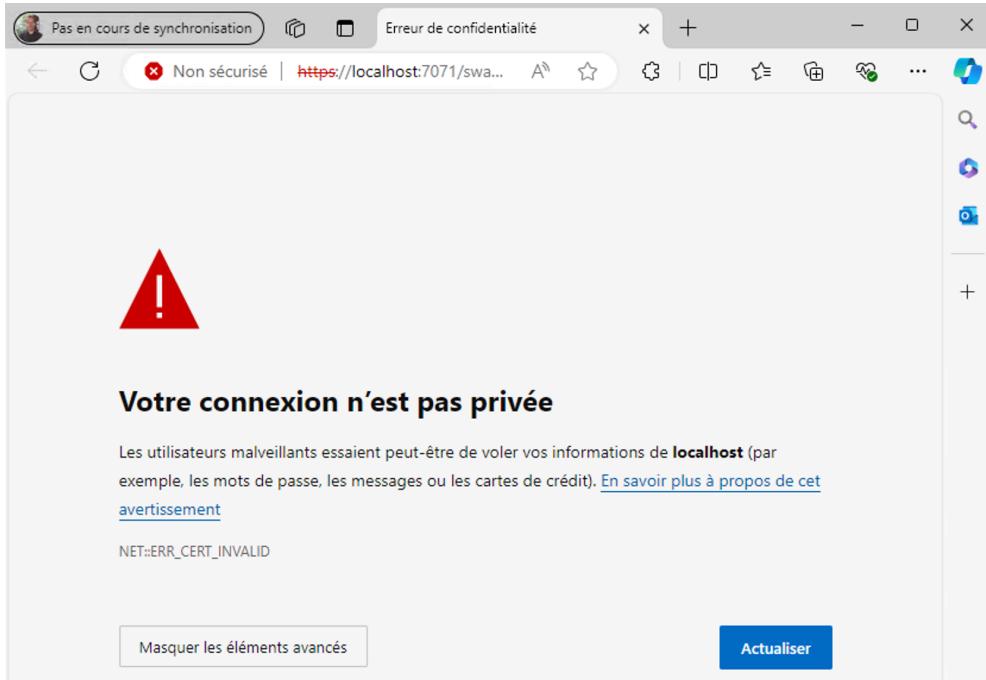
Voulez-vous approuver le certificat SSL ASP.NET Core ?

Ne plus me poser la question

Oui

Non

Il se peut que vous obteniez l'erreur suivante :

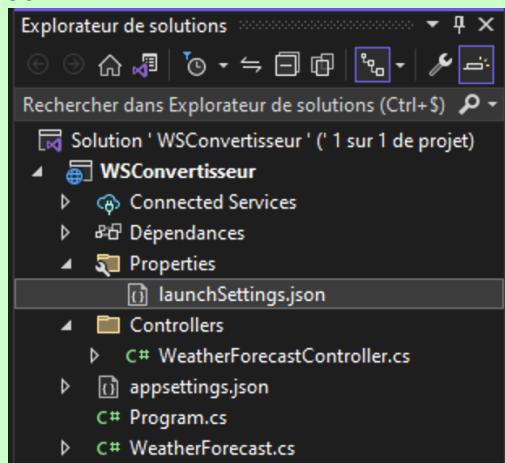


Arrêter et réessayer de le lancer une 2<sup>nde</sup> fois en https.

**Si cela ne fonctionne toujours pas, utiliser l'API en http.**

***En cas de problème d'installation du certificat, il suffit de mettre en commentaire la ligne app.UseHttpsRedirection(); afin de désactiver la redirection vers HTTPS.***

On peut voir et modifier la configuration du lancement de l'API dans Properties/launchSettings.json :



```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:20380",
      "sslPort": 44391
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5294",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

```
"ASPNETCORE_ENVIRONMENT": "Development"  
}  
},  
"https": {  
    "commandName": "Project",  
    "dotnetRunMessages": true,  
    "launchBrowser": true,  
    "launchUrl": "swagger",  
    "applicationUrl": "https://localhost:7071;http://localhost:5294",  
    "environmentVariables": {  
        "ASPNETCORE_ENVIRONMENT": "Development"  
    }  
},  
"IIS Express": {  
    "commandName": "IISExpress",  
    "launchBrowser": true,  
    "launchUrl": "swagger",  
    "environmentVariables": {  
        "ASPNETCORE_ENVIRONMENT": "Development"  
    }  
}  
}
```

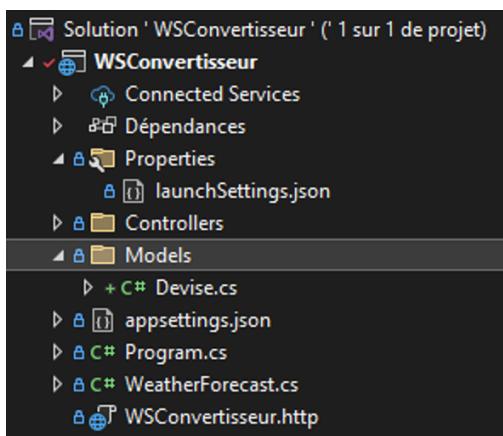
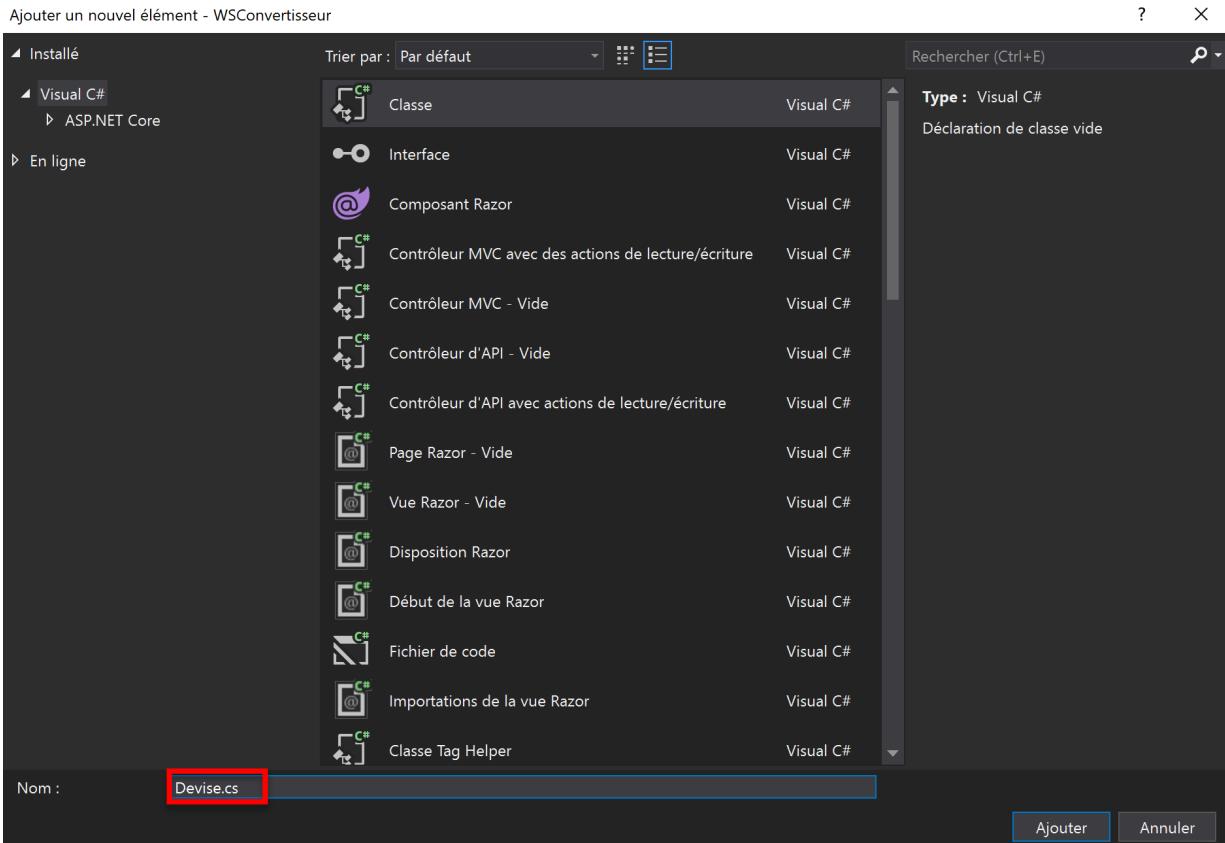
Par exemple, si l'on veut lancer directement le web service (sans passer par swagger) :

```
    "http": {  
        "commandName": "Project",  
        "dotnetRunMessages": true,  
        "launchBrowser": true,  
        "launchUrl": "weatherforecast",  
        "applicationUrl": "http://localhost:5294",  
        "environmentVariables": {  
            "ASPNETCORE_ENVIRONMENT": "Development"  
        }  
    }  
}
```

#### 1.4. Création de la couche M

Ajouter un dossier Models au projet WSConvertisseur.

Dans le dossier Models, ajouter la classe *Model Devise*. Pour cela, dans l'explorateur de la solution puis dossier Models, ajouter une classe (*Ajouter > Classe*).



Créer 3 properties dans la classe :

- Une pour l'Id (int) : Id
- Une pour le Nom de la devise (string?) : NomDevise
- Une pour le Taux (double) : Taux

Rappel : pour faciliter la construction des properties, saisir propfull (snippet) puis tabulation 2 fois.

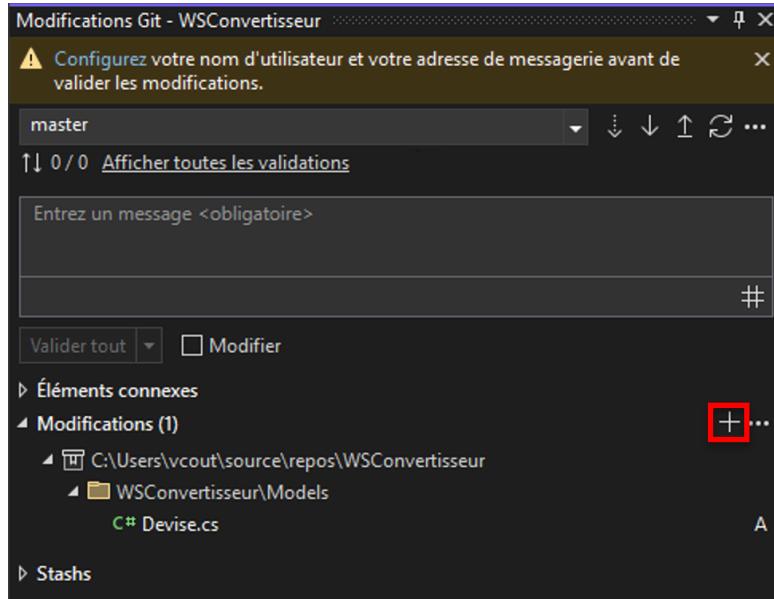
Ajouter un constructeur paramétré et un constructeur sans paramètre.

Rappel : pour générer le constructeur, utiliser le snippet ctor puis tab ou utiliser l'aide à la génération.

On peut voir que le fichier a été ajouté au projet mais n'a pas encore été commité :



Dans la fenêtre « Modifications Git », vous pouvez voir les ajouts (A) et les modifications (M) réalisées :



**Cliquez d'abord sur « Configurez votre nom d'utilisateur... », sinon vous ne pourrez pas commiter.**

Comme indiqué précédemment, un workflow Git consiste essentiellement à modifier des fichiers et à commiter ces changements.

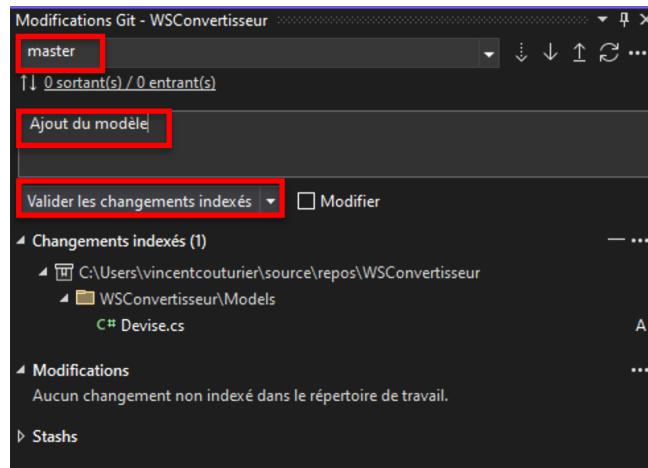
Git effectue le suivi des modifications de fichier dans votre dépôt (local) au fur et à mesure que vous travaillez et sépare les fichiers du dépôt en trois catégories :

- *Fichiers non modifiés* : ces fichiers n'ont pas changé depuis votre dernière validation.
- *Fichiers modifiés* : ces fichiers ont des modifications depuis votre dernière validation, mais vous ne les avez pas encore rendus indexés pour la validation (commit) suivante.
- *Fichiers indexés (staged files)* : ces fichiers ont des modifications qui seront ajoutées à la validation suivante.

Pour indexer les modifications lorsque vous êtes prêt, cliquer sur le bouton « + » sur chaque fichier ou utiliser le bouton « + » en haut de la section « Modifications » pour indexer toutes les modifications.

Lorsque vous indexer une modification, Visual Studio crée une section « Changements indexés », qui contient les modifications indexées. Seules les modifications de cette section sont ajoutées à la prochaine livraison (commit).

Il est maintenant possible de commiter en entrant un message et en cliquant sur le bouton « Valider les changements indexés ».

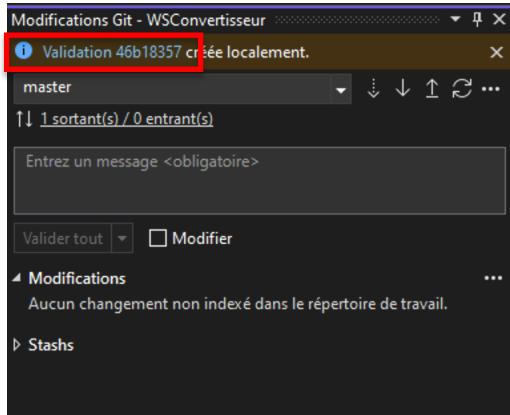


On remarque que l'on committe sur la branche « master » (il s'agit de la branche locale et non distante).

*Remarque : il est possible, avant de commiter, de désindexer les modifications indexées en cliquant sur le bouton « - ».*

Résultat :

Vincent COUTURIER



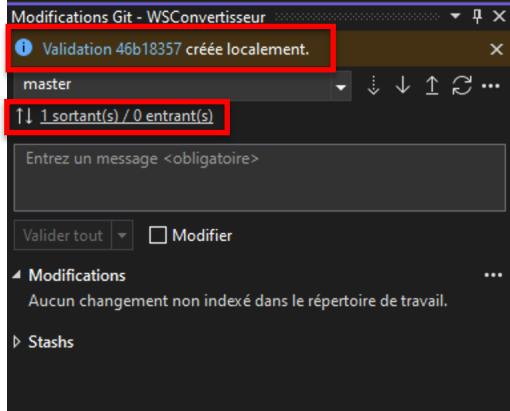
Aller sur Github.com et vérifier si vos fichiers ont été modifiés.

A screenshot of a GitHub repository page for "VincentCOUTURIER-USMB / R401\_WSConvertisseur". The "Code" tab is selected. The commit history shows five commits from "VincentCOUTURIER-USMB" made one hour ago. The commits are: "Ajoutez des fichiers projet.", "WSConvertisseur", ".gitattributes", ".gitignore", and "WSConvertisseur.sln".

Commit	Message	Date
39698c6	Ajoutez des fichiers projet.	1 hour ago
1	WSConvertisseur	1 hour ago
2	.gitattributes	1 hour ago
3	.gitignore	1 hour ago
4	WSConvertisseur.sln	1 hour ago

Ce n'est pas le cas... On remarque aussi que le nombre de commits n'a pas changé (toujours 2).

En effet, dans Visual Studio, il est bien indiqué que le commit a été réalisé sur le dépôt local :

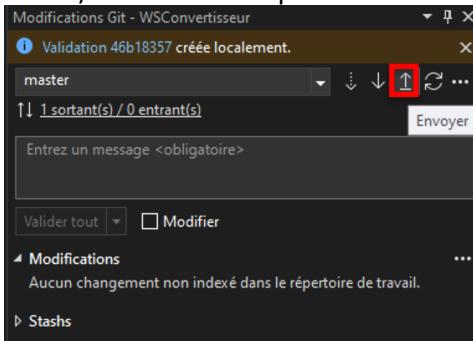


En cliquant sur celui-ci, on peut retrouver toutes les modifications ou ajouts effectués :

Il est possible d'annuler le commit et donc les modifications en cliquant sur « Rétablir » (**NE PAS LE FAIRE**).

On peut aussi visualiser l'historique des commits dans le menu *Git > Voir l'historique de la branche* :

Pour envoyer le code sur le dépôt distant, il suffit de cliquer sur le bouton « Envoyer » :



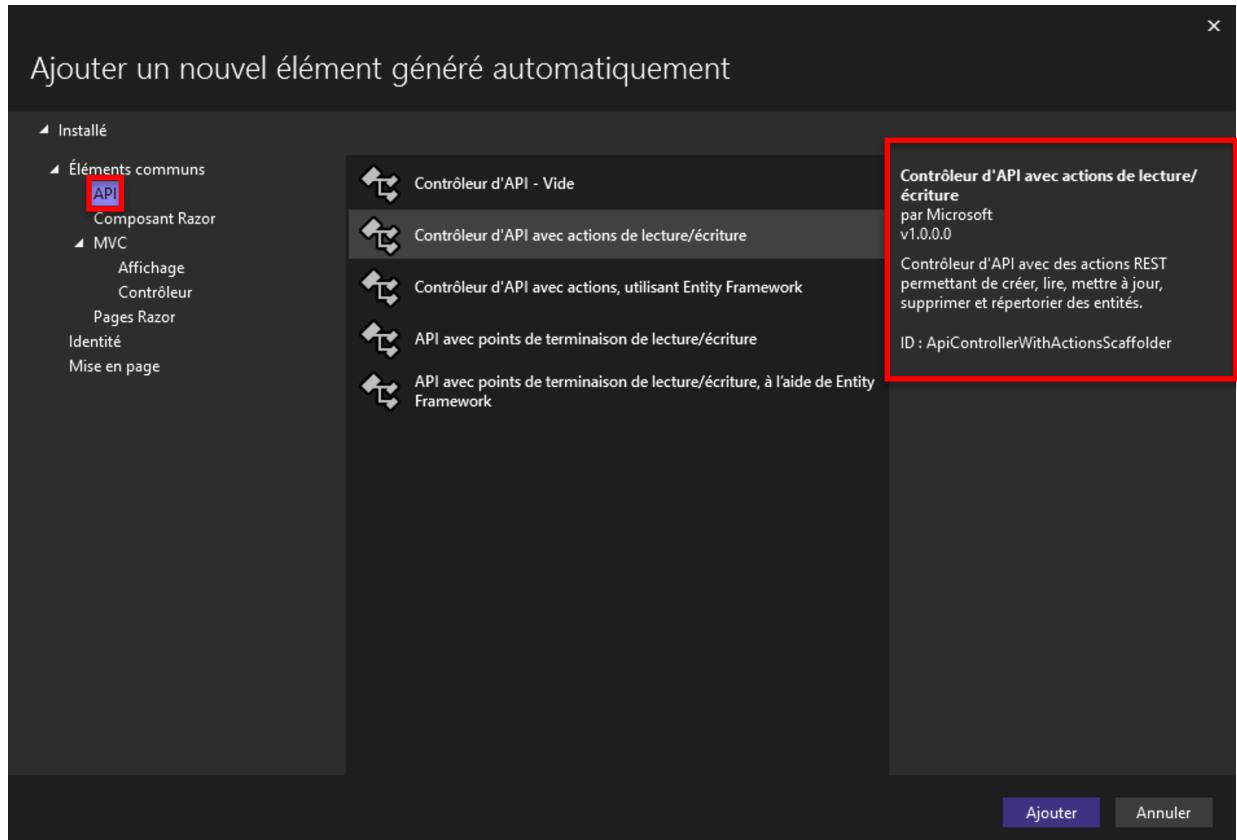
Résultat (mettre à jour l'historique) :

Aller sur [Github.com](https://github.com) et vérifier si vos fichiers ont été modifiés. Normalement, la couche *Model* a été ajoutée au repos distant et vous devriez avoir un commit supplémentaire.

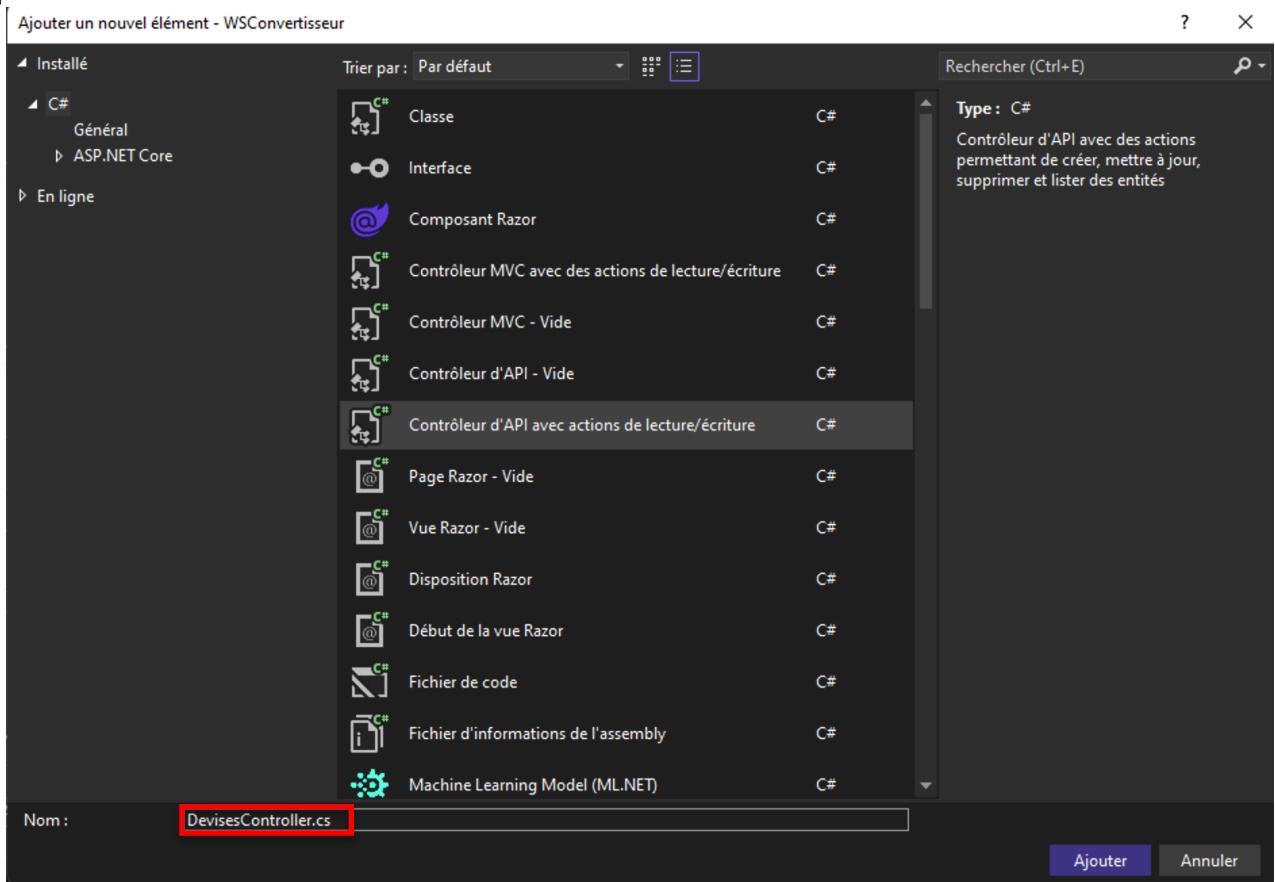
### 1.5. Création de la couche C

Un *contrôleur* est un objet qui gère les requêtes HTTP et crée la réponse HTTP.

Dans le dossier **Controllers** de la solution, ajouter un nouveau contrôleur (*Ajouter > Contrôleur*). Choisir « Contrôleur d'API avec actions de lecture/écriture » car il s'agit d'un contrôleur d'API REST et non d'un contrôleur ASP.NET Core MVC.

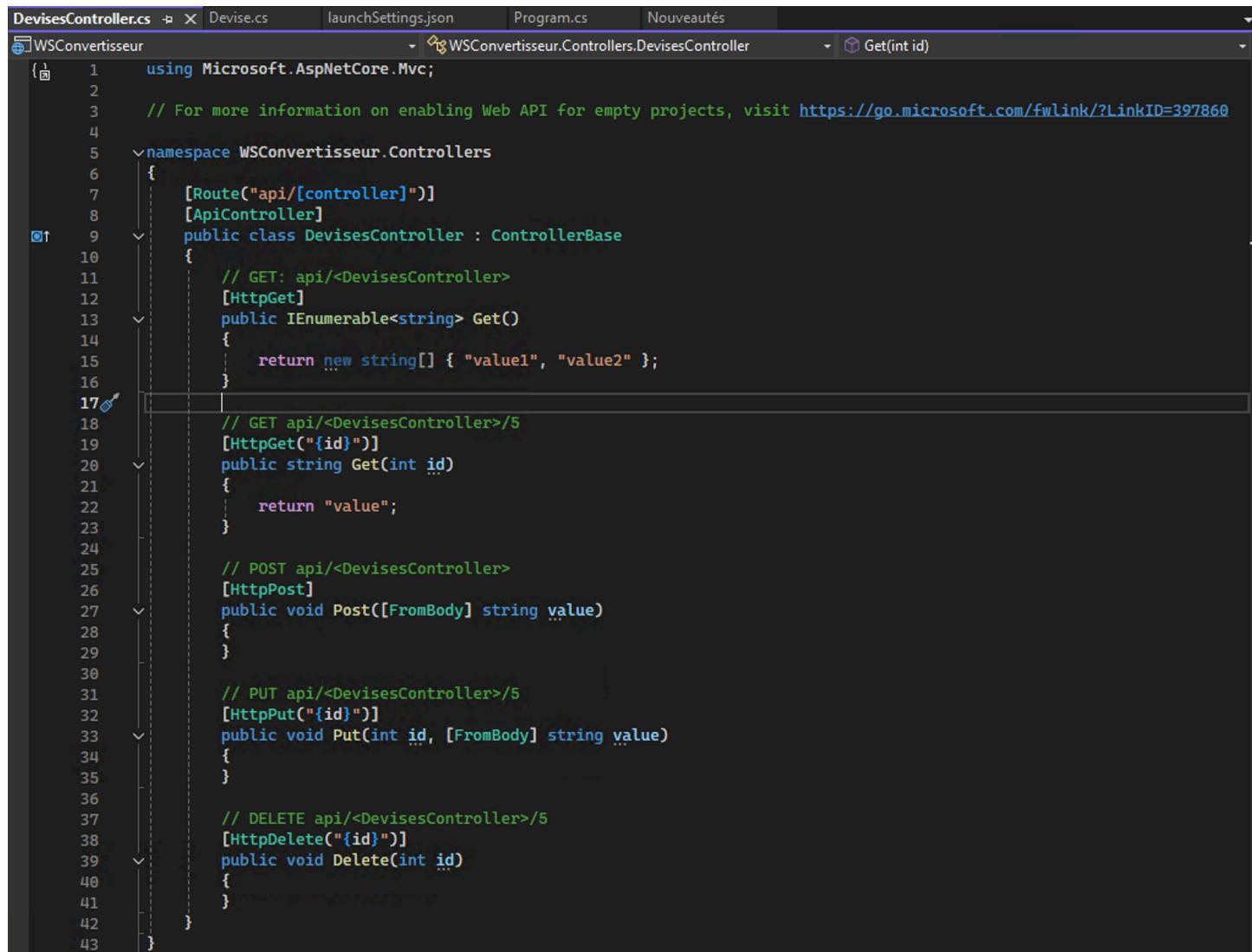


Vous pouvez renommer le nom du contrôleur en `DevisesController`.



En sélectionnant le modèle « Contrôleur d'API avec actions de lecture/écriture », Visual Studio génère le squelette de code correspondant aux méthodes GET (2 méthodes), PUT, POST et DELETE.

Cette technique de génération est appelée *scaffolding*.



The screenshot shows the Visual Studio code editor with the file 'DeviseController.cs' open. The code defines a controller named 'DevisesController' that inherits from 'ControllerBase'. It contains several methods: a GET method that returns an enumerable of strings, a GET method that takes an integer parameter 'id' and returns a single string 'value', a POST method that takes a string parameter 'value' and does nothing, a PUT method that takes an integer parameter 'id' and a string parameter 'value' and does nothing, and a DELETE method that takes an integer parameter 'id' and does nothing.

```
1  using Microsoft.AspNetCore.Mvc;
2
3  // For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860
4
5  namespace WSConvertisseur.Controllers
6  {
7      [Route("api/{controller}")]
8      [ApiController]
9      public class DevisesController : ControllerBase
10     {
11         // GET: api/<DevisesController>
12         [HttpGet]
13         public IEnumerable<string> Get()
14         {
15             return new string[] { "value1", "value2" };
16         }
17
18         // GET api/<DevisesController>/5
19         [HttpGet("{id}")]
20         public string Get(int id)
21         {
22             return "value";
23         }
24
25         // POST api/<DevisesController>
26         [HttpPost]
27         public void Post([FromBody] string value)
28         {
29         }
30
31         // PUT api/<DevisesController>/5
32         [HttpPut("{id}")]
33         public void Put(int id, [FromBody] string value)
34         {
35         }
36
37         // DELETE api/<DevisesController>/5
38         [HttpDelete("{id}")]
39         public void Delete(int id)
40         {
41         }
42     }
43 }
```

En entête de classe, on peut voir les annotations indiquant l'accès au contrôleur (*Route*) ainsi que le type de contrôleur (*ApiController*) :

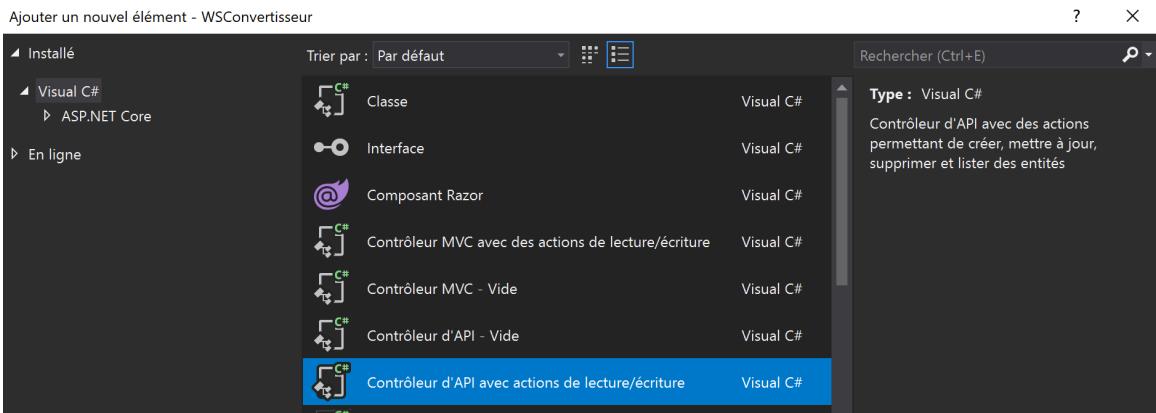
```
[Route("api/{controller}")]
[ApiController]
```

Explications :

- Le format de réponse est par défaut Json. On peut imposer ce format ou un autre en utilisant l'attribut `[Produces]` (par ex. `[Produces("application/json")]`). Par défaut, la Web API .NET Core prend en charge seulement Json : même si un autre format est spécifié dans le header `Accept` lors de l'appel, le résultat retourné est donc toujours au format JSON. Il est aussi possible d'utiliser d'autres formateurs, comme XML ou ATOM.
- `[ApiController]`. Cette annotation (attribut) n'est pas obligatoire mais permet d'activer les consignes strictes liées aux comportements spécifiques aux API : exigence du routage d'attribut, réponses HTTP 400 automatiques, utilisation des paramètres de source de liaison (`FromBody`, `FromRoute`, etc.), gestion des codes d'erreurs (404, etc.), etc.
- `[Route("api/[controller]")]` peut être remplacé par `[Route("api/Devises")]`.
- Un contrôleur d'API (i.e. contrôleur sans vue) hérite (normalement toujours, cf. remarque suivante) de la classe `ControllerBase` (`Microsoft.AspNetCore.Mvc.ControllerBase`) :  
<https://learn.microsoft.com/fr-fr/dotnet/api/microsoft.aspnetcore.mvc.controllerbase>

Remarques :

- On peut également ajouter un contrôleur d'API en utilisant Ajouter > Nouvel élément puis choisir « Contrôleur d'API avec actions de lecture/écriture » :



- **ATTENTION à ne pas choisir « Contrôleur MVC... » utilisé pour créer des applications Web ASP.NET avec vue razor. Dans ce cas, la classe utilisée sera Microsoft.AspNetCore.Mvc.Controller et non ControllerBase.**

Dans le contrôleur, ajouter un constructeur et y créer une liste (`List`) de devises (type `Devise`) contenant les données suivantes :

1	Dollar	1.08
2	Franc Suisse	1.07
3	Yen	120

### Méthode Get()

Renommer la 1<sup>ère</sup> méthode en `GetAll`.

En entête de méthode, on peut voir `[HttpGet]`. Cet attribut désigne une méthode qui répond à une requête HTTP GET. Le chemin d'URL (route) pour chaque méthode est construit ainsi :

- Prend le modèle de chaîne dans l'attribut `Route` du contrôleur (`[Route("api/Devises")]`).
- Si l'attribut `[HttpGet]` a un modèle de route (comme `[HttpGet("/produits")]`), il sera ajouté au chemin. Cet exemple n'utilise pas de modèle. Plus de détails ici :

<https://learn.microsoft.com/fr-fr/aspnet/core/mvc/controllers/routing?view=aspnetcore-8.0#attribute-routing-with-httpverb-attributes>

Modifier le code de la méthode (`public IEnumerable<string> GetAll()`) afin qu'elle retourne la liste des devises (type de retour de la méthode : `IEnumerable<Devise>`).

Remarques :

- *IList implémente IEnumerable. IEnumerable<Devise> permet de gérer des collections non génériques. Même si nous allons gérer une List de devises, il est préférable d'utiliser cette interface dans une API car elle est moins spécifique qu'IList et conviendra donc quelle que soit collection retournée.*
- *Le type de retour IQueryable<Devise> est également possible (meilleures performances si nombreuses données).*
- *Differences entre IEnumerable et IQueryable :*
  - <https://www.codeproject.com/Articles/832189/List-vs-IEnumerable-vs-IQueryable-vs-ICollection-v>
  - <http://stackoverflow.com/questions/4455428/difference-between-iqueryable-icollection-ilist-idictionary-interface>

La méthode Get récupérant toutes les devises est maintenant fonctionnelle et se consomme de cette façon : GET /api/Devises

Exécuter l'application. Vous pourrez utiliser swagger ou modifier la configuration de `launchSettings.json` pour appeler le contrôleur `Devises` par défaut (`"launchUrl": "api/devises"`).

```
[{"id":1,"nomDevise":"Dollar","taux":1.08}, {"id":2,"nomDevise":"Franc Suisse","taux":1.07}, {"id":3,"nomDevise":"Yen","taux":120}]
```

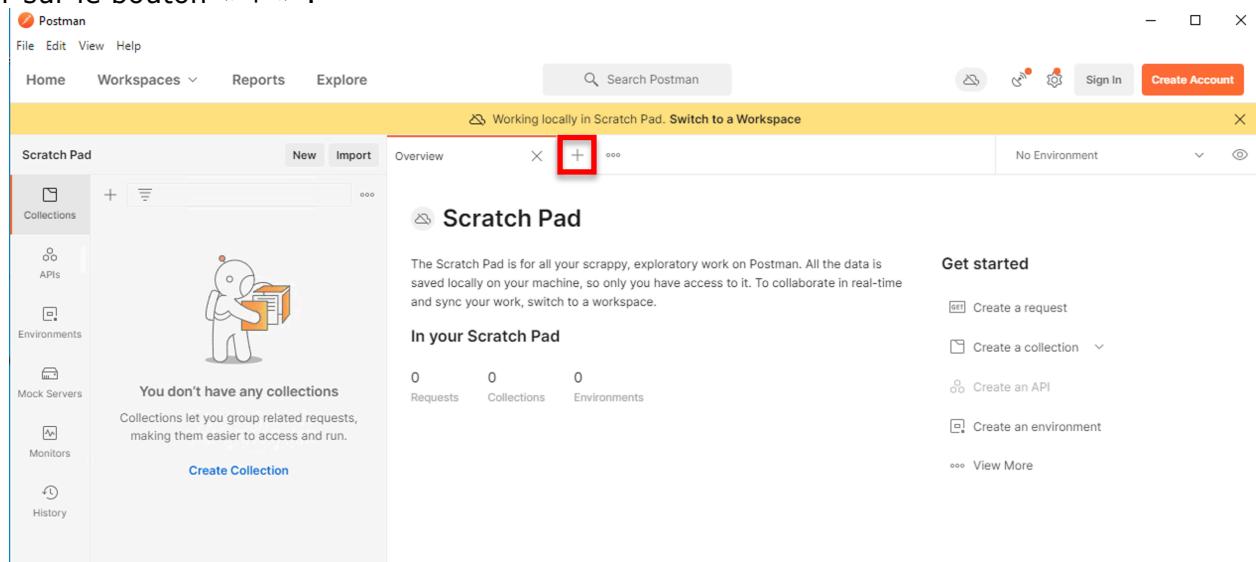
**Noter le port utilisé. Ici : 7223 pour le hosting environment en https et 5223 pour http.**

Ne pas fermer le navigateur.

Nous allons tester l'appel au WS (hébergé localement) dans l'outil « Postman ».

Lancer Postman disponible dans les applications Windows.

Cliquer sur le bouton « + » :



Saisir l'URL locale du WS, choisir la méthode « GET ». Il n'est pas nécessaire d'ajouter un header Accept car le format de retour est obligatoirement Json (même si vous spécifiez un autre format).

Cliquer sur « Send ».

**Postman pose parfois des problèmes pour tester des WS https. Pour résoudre ces problèmes, il suffit de cliquer sur « Disable SSL Verification » lors de l'exécution :**

SSL Error: Unable to verify the first certificate | **Disable SSL Verification**

**Sinon :**

- **Mettre en commentaire app.UseHttpsRedirection() puis tester l'appel sur le port 5xxx (en http).**
- **Ou utiliser swagger pour tester l'API.**

On obtient la liste de toutes les devises au format JSON.

The screenshot shows the Postman interface with a successful API call. The URL is `https://localhost:7223/api/devises`. The response status is **200 OK** with a response time of 1088 ms and a size of 277 B. The JSON response body is:

```

1 [
2   {
3     "id": 1,
4     "nomDevise": "Dollar",
5     "taux": 1.08
6   },
7   {
8     "id": 2,
9     "nomDevise": "Franc Suisse",
10    "taux": 1.07
11  },
12  {
13    "id": 3,
14    "nomDevise": "Yen",
15    "taux": 120
16  }
17 ]

```

Noter le statut (**200 OK**).

Vous remarquerez les [ ] dans le JSON, indiquant que l'on obtient une collection (tableau).

```
[  
  {  
    "Id": 1,  
    "nomDevise": "Dollar",  
    "taux": 1.08  
  },  
  {  
    "id": 2,  
    "nomDevise": "Franc Suisse",  
    "taux": 1.07  
  },  
  {  
    "Id": 3,  
    "nomDevise": "Yen",  
    "taux": 120  
  }]  
]
```

Mettre un point d'arrêt sur la première ligne de code du constructeur.

```

13  

14     public DeviseController()  

15     {  

16       devises = new List<Devise>();  

17       devises.Add(new Devise(1, "Dollar", 1.08));  

18       devises.Add(new Devise(2, "Franc Suisse", 1.07));  

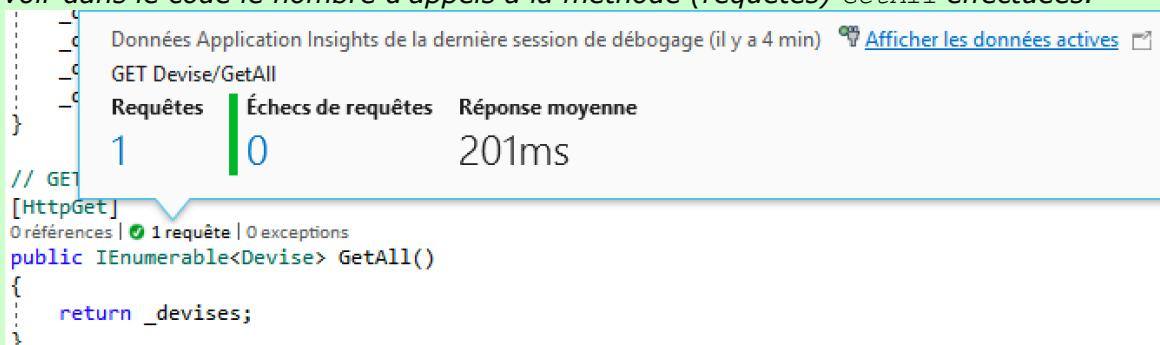
19       devises.Add(new Devise(3, "Yen", 120));  

20     }
21

```

Ré-exécuter l'appel GET dans Postman (vous pouvez le faire plusieurs fois). On remarque que le constructeur est exécuté à chaque appel. Autrement dit, un nouvel objet est instancié à chaque appel. Ainsi, la liste des devises est recréée à chaque fois.

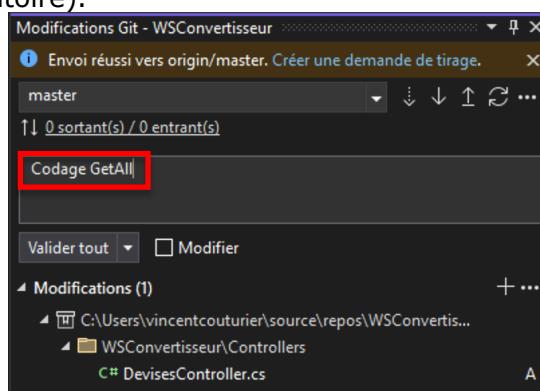
On peut voir dans le code le nombre d'appels à la méthode (requêtes) GetAll effectuées.



The screenshot shows the Application Insights interface with the 'Requêtes' tab selected. It displays 1 request for the 'GET Devise/GetAll' endpoint. Below the interface, the corresponding C# code is shown:

```
// GET [HttpGet]
public IEnumerable<Devise> GetAll()
{
    return _devises;
}
```

Committer (vous pouvez cette fois cliquer sur le bouton « Valider tout » afin de passer l'étape d'indexation, qui n'est pas obligatoire).



## Méthode Get (id)

Renommer la méthode en GetById.

```
[HttpGet("{id}", Name = "GetDevise")]
```

Name = "GetDevise" crée un itinéraire nommé. Les itinéraires nommés permettent à l'application de créer une liaison HTTP à l'aide du nom de l'itinéraire. Nous verrons ultérieurement à quoi sert un itinéraire nommé.

Dans cette méthode, il s'agit ici de parcourir la liste de devises et de renvoyer la bonne devise (type de retour de la méthode : Devise).

Pour parcourir la liste, plusieurs possibilités :

- Faire une boucle foreach (pas terrible !)
- Créer une instruction LINQ <https://learn.microsoft.com/fr-fr/dotnet/csharp/linq/get-started/write-linq-queries> :
  - o En pseudo SQL, on récupère une liste d'éléments correspondant à la clause where
 

```
IEnumerable<Devise> devise =
            from d in devises
            where d.Id == id
            select d;
            return devise.ToList()[0];
```

OU MIEUX (car le code précédent plante si la requête ne renvoie rien) :

```
Devise? devise =
  (from d in devises
  where d.Id == id
  select d).FirstOrDefault();
return devise;
```

- o En lambda expression :

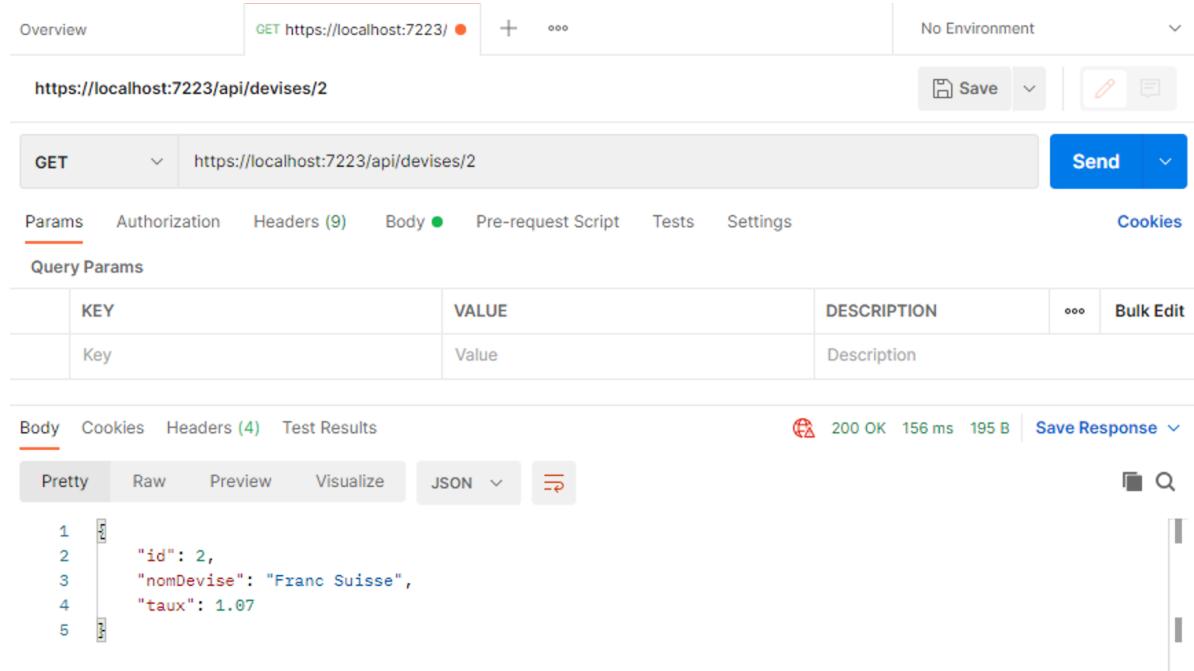
- Devise? devise = devises.FirstOrDefault((d) => d.Id == id);
- First/FirstOrDefault :
 <http://stackoverflow.com/questions/1024559/when-to-use-first-and-when-to-use-firstOrDefault-with-linq>

Ajouter le code nécessaire permettant de retourner la devise dont l'id est passé en paramètre.

Le mode par défaut pour récupérer le paramètre est [FromRoute] : public Devise GetById(int id) (↔ public Devise GetById([FromRoute] int id)). Cela signifie que l'ID va être récupéré à partir du chemin (Uri) d'appel du WS. [FromRoute] étant le mode par défaut, il n'est pas nécessaire de le préciser.

La méthode GetById récupérant une seule devise est maintenant fonctionnelle et se consomme de cette façon : GET /api/devise/id, par exemple : /api/devise/2

Résultat :

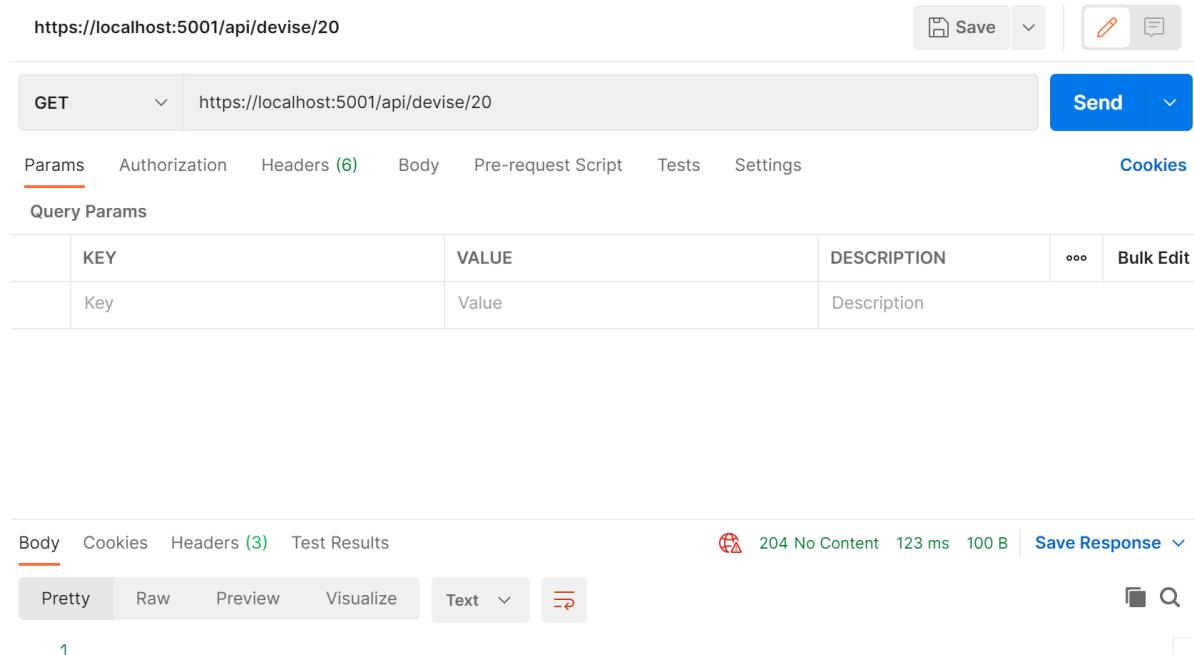


The screenshot shows a Postman request for `https://localhost:7223/api/devises/2`. The response body is:

```
1
2   "id": 2,
3   "nomDevise": "Franc Suisse",
4   "taux": 1.07
5
```

On remarque cette fois-ci qu'il ne s'agit pas d'une collection (pas de [ ]).

Tester avec un ID inexistant. On récupère une erreur 204 No Content, ce qui n'est pas un retour standard. Ce devrait être une erreur http 404 Not Found.



The screenshot shows a Postman request for `https://localhost:5001/api/devise/20`. The response body is:

```
1
```

Nous allons améliorer le code afin de gérer l'erreur 404 Not Found.

```
[HttpGet("{id}", Name = "GetDevise")]
public ActionResult<Devise> GetById(int id)
{
    Devise? devise = devises.FirstOrDefault(d => d.Id == id);
```

```

if (devise == null)
{
    return NotFound();
}
return devise;
}

```

- La classe `ActionResult` définit une commande qui crée de manière asynchrone un message http de réponse contenant une représentation (Json) de la devise rentrée ou un message d'erreur (`404 Not Found`). `ActionResult` gère également les différents codes d'état http, comme `BadRequestResult` (400), `NotFoundResult` (404) et `OkObjectResult` (200).
- Si aucun objet `Devise` n'a été trouvé, c'est la méthode `NotFound()` de la classe `Controller` qui est appelée. Elle générera une erreur 404 (`StatusCodes.Status404NotFound`).

`NotFound()`

Creates an [NotFoundResult](#) that produces a [Status404NotFound](#) response.

Tester le WS avec un ID existant puis avec un ID inconnu. Dans ce 2<sup>nd</sup> cas, une erreur 404 est maintenant rentrée.

The screenshot shows the Postman interface. A GET request is made to `https://localhost:7223/api/devises/20`. The response status is **404 Not Found**, with a time of 51 ms and a size of 325 B. The response body is a JSON object:

```

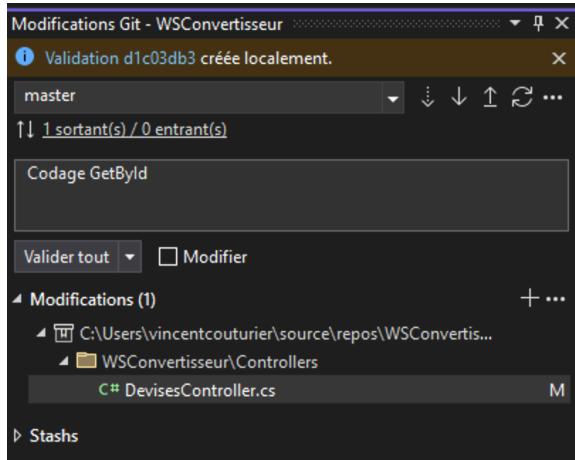
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",
  "title": "Not Found",
  "status": 404,
  "traceId": "00-8ae4291c44a7160eef410c9429074e41-b8d310857ec69495-00"
}

```

Dans la fenêtre « Modification Git », cliquer sur le fichier modifié :

The screenshot shows the Visual Studio Code interface with the DeviseController.cs file open in the Diff view. The changes are highlighted in red and green. The right-hand panel shows the Git modifications, where the file C# DeviseController.cs is selected.

Vous pouvez visualiser les modifications entre le commit précédent et la version actuelle du code. Commiter (<> Valider tout </>).



Vous pouvez envoyer vers le dépôt distant si vous le souhaitez.

**Vous commiterez au fur et à mesure. N'oubliez pas d'envoyer votre code vers le repos distant avant la fin de la séance pour pouvoir le continuer en séance suivante.**

Remarque : Pour récupérer votre code (cloner) à partir du dépôt, voir annexe en fin de document.

## Méthode Post

```
[HttpPost]
public ActionResult<Devise> Post([FromBody] Devise devise)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    devises.Add(devise);
    return CreatedAtRoute("GetDevise", new { id = devise.Id }, devise);
}
```

- Prend un objet `Devise` en paramètre et retourne un `ActionResult` contenant une représentation Json de la devise créée (choix de développement) ou une erreur 400 Bad Request en cas d'échec de la validation du modèle.
- Le code précédent est une méthode HTTP POST, comme indiqué par l'attribut `[HttpPost]`. L'attribut `[FromBody]` indique qu'il faut obtenir les informations de la devise à partir du corps (body) de la requête HTTP. Le JSON devra être intégré dans la partie `body` de l'interface de Swagger ou de Postman.  
Autres attributs possibles en fonction des méthodes : `FromQuery`, `FromRoute`, `FromHeader`, `FromForm`  
<https://learn.microsoft.com/fr-fr/aspnet/core/mvc/models/model-binding>  
<https://www.dotnetcurry.com/aspnet/1390/aspnet-core-web-api-attributes>
- `ModelState.IsValid` indique si une (ou plusieurs) erreur de modèle (classe métier `Devise`) ont été ajoutées à `ModelState`. Des erreurs peuvent être ajoutées dans le cas de problèmes d'encodage ou de conversion (par exemple, du texte, alors qu'un `int` ou `float` est attendu).  
Exemple, une erreur sera levée lorsque vous passez le fichier JSON suivant en `body` (voir écran Postman plus bas) :

```
{
    "id": 4,
    "nomdevise": "Rouble",
    "taux": AAA
}
```

```

if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
_devises.Add(devise);
return CreatedAtAction("GetDevise", new { id = devise.Id });
}

```

L'erreur d'encodage est ensuite retournée au programme appelant :

```

if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
_devises.Add(devise);
return CreatedAtAction("GetDevise", new { id = devise.Id });
}

```

Affichage : une erreur **400 Bad Request** est générée par la méthode `BadRequest()` de la classe `Controller` avec le `ModelState` en paramètre.

POST https://localhost:7223/api/devises

Body (JSON)

```

1 {
2     "id": 4,
3     "nomDevise": "Rouble",
4     "taux": AAA
5 }

```

400 Bad Request

```

1 {
2     "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
3     "title": "One or more validation errors occurred.",
4     "status": 400,
5     "traceId": "00-19625ef7a55a309229a508cc37db938f-0b3c2ef2dc4ce824-00",
6     "errors": {
7         "devise": [
8             "The devise field is required."
9         ],
10        "$.taux": [
11            "'A' is an invalid start of a value. Path: $.taux | LineNumber: 3 | BytePositionInLine: 10."
12        ]
13    }
14 }

```

De même, si vous passez le fichier JSON Suivant, une erreur **400 Bad Request** sera levée.

```
{
    "id": 4,
    "nomdevise": "Rouble",
```

The screenshot shows a Postman request to `https://localhost:7223/api/devises`. The request method is `POST` and the URL is `https://localhost:7223/api/devises`. The `Body` tab is selected, showing a JSON payload:

```

1
2     "id": 4,
3     "nomDevise": "Rouble",
4     "taux": "AAA"
5

```

The response status is `400 Bad Request` with a `Content-Type` of `application/json`. The response body is:

```

1
2     "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
3     "title": "One or more validation errors occurred.",
4     "status": 400,
5     "errors": {
6         "$": [
7             "The JSON value could not be converted to System.String. Path: $ | LineNumber: 0 | BytePositionInLine: 1."
8         ],
9         "value": [
10            "The value field is required."
11        ]
12    },

```

- La méthode `CreatedAtRoute("GetDevise", new { id = devise.id }, devise)` de Controller génère l'URL (chemin) suivante `/api/devises/IdInséré` permettant d'afficher la devise insérée.  
`CreatedAtRoute(String, Object, Object)`

Creates a [CreatedAtRouteResult](#) object that produces a [Status201Created](#) response.

```
C#
[Microsoft.AspNetCore.Mvc.NonAction]
public virtual Microsoft.AspNetCore.Mvc.CreatedAtRouteResult CreatedAtRoute (string
routeName, object routeValues, object value);
```

#### Parameters

`routeName` `String`

The name of the route to use for generating the URL.

`routeValues` `Object`

The route data to use for generating the URL.

`value` `Object`

The content value to format in the entity body.

#### Returns

`CreatedAtRouteResult`

The created [CreatedAtRouteResult](#) for the response.

Elle :

- Retourne une réponse 201. HTTP 201 est la réponse standard d'une méthode HTTP POST qui crée une ressource sur le serveur.
- Ajoute un en-tête `Location` à la réponse. L'en-tête `Location` spécifie l'URI de l'élément d'action qui vient d'être créé.
- Utilise l'itinéraire nommé « `GetDevise` » pour créer l'URL. Il est défini dans la méthode  `GetById : [HttpGet("{id}"), Name = "GetDevise"]`.

Remarque : à la place, il est aussi possible d'appeler directement l'action (la méthode) de la façon suivante : `CreatedAtAction(" GetById", new { id = devise.id }, devise)`

### CreatedAtAction(String, Object, Object)

Creates a [CreatedAtActionResult](#) object that produces a [Status201Created](#) response.

```
C#  
[Microsoft.AspNetCore.Mvc.NonAction]  
public virtual Microsoft.AspNetCore.Mvc.CreatedAtActionResult CreatedAtAction (string  
actionName, object routeValues, object value);
```

#### Parameters

**actionName** `String`

The name of the action to use for generating the URL.

**routeValues** `Object`

The route data to use for generating the URL.

**value** `Object`

The content value to format in the entity body.

#### Returns

`CreatedAtActionResult`

The created [CreatedAtActionResult](#) for the response.

La méthode Post permettant d'insérer une devise est maintenant fonctionnelle et se consomme de cette façon : `POST /api/Devises` en lui passant une représentation Json.

Test d'insertion d'une devise valide :

The screenshot shows the Postman interface. The URL is `https://localhost:7223/api/devises`. The method is set to `POST`. In the `Body` tab, the `JSON` tab is selected, and the following JSON is entered:

```
1  {  
2      "id": 4,  
3      "nomDevise": "Rouble",  
4      "taux": 1510.5  
5  }
```

The response at the bottom shows a `201 Created` status with a `227 ms` response time and `244 B` size. The JSON response body is identical to the one sent:

```
1  {  
2      "id": 4,  
3      "nomDevise": "Rouble",  
4      "taux": 1510.5  
5  }
```

On remarque le statut « 201 Created » et que le nouvel enregistrement est retourné.

### Exécuter ensuite la méthode GET permettant de récupérer toutes les devises.

On remarque que la devise 4 n'est pas dans le JSON renvoyé et donc absente de la liste des devises. Pourtant, elle a bien été ajoutée à la liste (vous pouvez le vérifier en mettant un point d'arrêt ou un espion dans la méthode `Post`). En effet, comme nous l'avons vu précédemment la liste de devises est recréée à chaque appel au WS. Le WS réalisé est donc sans état. Une base de données est donc nécessaire afin de sauvegarder les modifications de devises (ajout, Maj, suppression). Celle-ci est à gérer dans la couche `Model` via un CRUD créé manuellement ou via l'ORM Entity Framework Core (nous le ferons dans un prochain TP).

Test d'insertion de données manquantes : par défaut, on peut créer une devise sans nom (ou sans ID ou sans taux).

The screenshot shows the Postman interface with a POST request to `http://localhost:5000/api/devise`. The Body tab is selected, showing the following JSON payload:

```
1 ~ {  
2   "id": 4,  
3   "taux": 1510.5  
4 }  
5
```

The response status is 201 Created, time 776ms, size 238 B.

Modifier la classe Devise ainsi :

```
private string? nomDevise;  
  
[Required]  
public string? NomDevise  
{  
    get { return nomDevise; }  
    set { nomDevise = value; }  
}
```

Rajouter le namespace `using System.ComponentModel.DataAnnotations;`  
Réexécuter le POST précédent. Vous obtiendrez l'erreur suivante :

The screenshot shows a POST request to `https://localhost:7223/api/devises`. The Body tab is selected, showing the same JSON payload as before:

```
1 ~ {  
2   "id": 4,  
3   "taux": 1.5  
4 }
```

The response status is 400 Bad Request, time 71 ms, size 417 B.

The error response body is:

```
1 ~ {  
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",  
3   "title": "One or more validation errors occurred.",  
4   "status": 400,  
5   "errors": {  
6     "NomDevise": [  
7       "The NomDevise field is required."  
8     ]  
9   },  
10  "traceId": "00-4c8f81057bf4a8828b05e0c648866fcb-ac57edf8cc33b4cf-00"  
11 }
```

On peut rajouter de nombreuses annotations au modèle, spécifiant notamment des contraintes, comme [Phone], [Url], [EmailAddress], [RegularExpression], etc. (nous y reviendrons dans un prochain TP).

### Méthode Delete

Coder la méthode `Delete` ressemblant à la méthode  `GetById(int Id)` :

- Même type de retour que les méthodes précédentes.
- Utiliser une expression LINQ permettant de récupérer dans la liste la devise dont l'Id est passé en paramètre.
- Si la devise est inexistante, renvoyer une erreur *Not Found*.
- Utiliser la méthode `Remove` de la liste pour supprimer la devise.
- Retour : la devise supprimée (on peut aussi renvoyer un `NoContent()`)

La méthode `Delete` se consomme de cette façon : `DELETE /api/devises/id`.

Tester en mettant un point d'arrêt ou un espion pour vérifier que la devise est bien supprimée de la liste.

### Méthode Put

```
[HttpPut("{id}")]
public ActionResult Put(int id, [FromBody] Devise devise)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    if (id != devise.Id)
    {
        return BadRequest();
    }
    int index = devises.FindIndex((d) => d.Id == id);
    if (index < 0)
    {
        return NotFound();
    }
    devises[index] = devise;
    return NoContent();
}
```

- Ici, nous avons fait le choix de ne pas retourner l'objet `Devise` modifié (`return devise`), mais juste un statut de réponse `http 204 No content`, sans contenu JSON.

D'après la spécification HTTP, une requête PUT nécessite que le client envoie toute l'entité mise à jour, et pas seulement les différences. Pour prendre en charge les mises à jour partielles, il faut utiliser HTTP PATCH (nous le ferons dans un prochain TP) :

<https://www.infoworld.com/article/2254018/how-to-perform-partial-updates-to-rest-web-api-resources.html>  
<https://www.youtube.com/watch?v=oVzD74AJL4E>.

La méthode PUT permettant de mettre à jour une devise se consomme de cette façon : `PUT /api/devises/id` en lui passant une représentation Json toujours en body.

Tests de modifications non valides :

https://localhost:5001/api/devise/2

PUT https://localhost:5001/api/devise/2

Params Authorization Headers (8) Body **JSON** Pre-request Script Tests Settings Cookies Beautify

```

1 {
2   "id": 2,
3   "NomDevise": "Franc Suisse",
4   "taux": "AAAA"
5 }
```

Body Cookies Headers (4) Test Results

400 Bad Request 272 ms 491 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
3   "title": "One or more validation errors occurred.",
4   "status": 400,
5   "traceId": "00-07382c1972050c46a6329a7a08b4055f-a97ed5a031c40740-00",
6   "errors": {
7     "$.taux": [
8       "The JSON value could not be converted to System.Double. Path: $.taux | LineNumber: 3 |
9       BytePositionInLine: 16."
10    ]
11 }
```

https://localhost:5001/api/devise/1

PUT https://localhost:5001/api/devise/1

Params Authorization Headers (8) Body **JSON** Pre-request Script Tests Settings Cookies Beautify

```

1 {
2   "id": 2,
3   "NomDevise": "Franc Suisse",
4   "taux": 1.8
5 }
```

Body Cookies Headers (4) Test Results

400 Bad Request 157 ms 328 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
3   "title": "Bad Request",
4   "status": 400,
5   "traceId": "00-0172c417d3000743a6002b473dc723d2-0a2a6fa854d4d447-00"
6 }
```

The screenshot shows a POST request to <https://localhost:5001/api/devise/50>. The Body tab is selected, showing a JSON payload:

```

1 {
2   "id": 50
3   "NomDevise": "Franc Suisse",
4   "taux": 1.8
5 }

```

The response status is 404 Not Found, with a response body indicating a not found error:

```

1 {
2   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",
3   "title": "Not Found",
4   "status": 404,
5   "traceId": "00-3134a2df76e9b349914b31be8162e995-87410f51eab62647-00"
6 }

```

**Modification valide :**

Tester en mettant un point d'arrêt pour vérifier que la devise est bien mise à jour dans la liste.

The screenshot shows a POST request to <https://localhost:5001/api/devise/2>. The Body tab is selected, showing a JSON payload:

```

1 {
2   "id": 2,
3   "NomDevise": "Franc Suisse",
4   "taux": 1.8
5 }

```

The response status is 204 No Content.

### 1.6. Documentation de l'API

Une API (OpenAPI) doit toujours fournir une documentation afin que les développeurs sachent comment la consommer.

La documentation doit contenir la liste des actions disponibles (URL, méthode), leurs paramètres et leur réponse (code de statut http, corps).

Nous allons réaliser la documentation à partir du code réalisé et de commentaires XML. Celle-ci sera visualisable dans Swagger.

## 1. Ajouter des commentaires XML (documentation) en entête de **chaque action** du contrôleur.

### Commentaires XML :

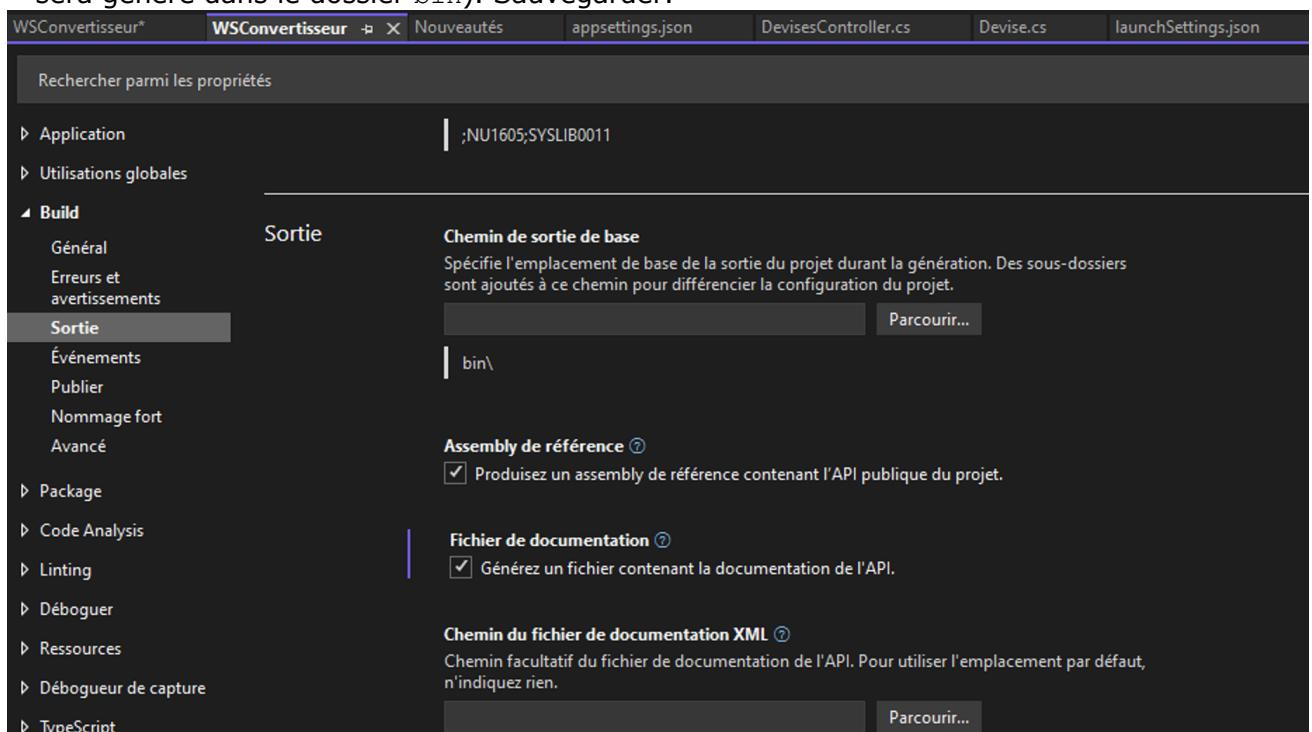
- o <summary></summary> : Description de l'action.
- o <remarks></remarks> : Information supplémentaire, exemple.
- o <returns><returns> : Description de la valeur de retour
- o <param name="id"></param> : Description d'un paramètre d'entrée
- o <response code="201"></response> : Description de la valeur de retour quand status code = 201.

### Exemple :

```
///<summary>
/// Get a single currency.
///</summary>
///<returns>Http response</returns>
///<param name="id">The id of the currency</param>
///<response code="200">When the currency id is found</response>
///<response code="404">When the currency id is not found</response>
// GET api/<DevisesController>/5
[HttpGet("{id}", Name = "GetDevise")]
public ActionResult<Devise> GetByid(int id)
```

Vous pouvez également commenter la classe *Model*.

## 2. Dans les propriétés du projet (bouton droit de la souris sur le nom du projet puis *Propriétés*), cocher « Fichier de documentation XML » dans l'onglet « Build ». Laisser le chemin par défaut (le fichier sera généré dans le dossier `bin`). Sauvegarder.



Toute méthode non commentée sera maintenant soulignée en vert :

```
// POST api/<DevisesController>
[HttpPost]
public ActionResult<Devise> Post([FromBody] Devise devise)
{
    if (!ModelState.IsValid)
    {
```

ActionResult<Devise> DevisesController.Post(Devise devise)

CS1591: Commentaire XML manquant pour le type ou le membre visible publiquement 'DevisesController.Post(Devise)'

Des avertissements seront également générés :

Liste d'erreurs					Rechercher dans		
Solution complète	Code	Description		Projet	Fichier	Li...	État de
CS1591	Commentaire XML manquant pour le type ou le membre visible publiquement 'DevisController'			WSConvertisseur	DevisController.cs	14	Actif
CS1591	Commentaire XML manquant pour le type ou le membre visible publiquement 'DevisController.DevisController()'			WSConvertisseur	DevisController.cs	18	Actif
CS1591	Commentaire XML manquant pour le type ou le membre visible publiquement 'DevisController.GetAll()'			WSConvertisseur	DevisController.cs	28	Actif
CS1591	Commentaire XML manquant pour le type ou le membre visible publiquement 'DevisController.Post(Devis)'			WSConvertisseur	DevisController.cs	56	Actif
CS1591	Commentaire XML manquant pour le type ou le membre visible publiquement 'DevisController.Put(int Devis)'			WSConvertisseur	DevisController.cs	68	Actif

Une fois le projet généré, la documentation se trouve dans le dossier bin du projet :

Nom	Modifié le	Type	Taille
appsettings.Development.json	09/09/2024 15:17	JSON File	1 Ko
appsettings.json	09/09/2024 15:17	JSON File	1 Ko
Microsoft.OpenApi.dll	29/08/2020 22:14	Extension de l'app...	170 Ko
Swashbuckle.AspNetCore.Swagger.dll	19/07/2022 17:21	Extension de l'app...	15 Ko
Swashbuckle.AspNetCore.SwaggerGen.dll	19/07/2022 17:21	Extension de l'app...	95 Ko
Swashbuckle.AspNetCore.SwaggerUI.dll	19/07/2022 17:21	Extension de l'app...	3 337 Ko
WSConvertisseur.deps.json	09/09/2024 17:42	JSON File	5 Ko
WSConvertisseur.dll	09/09/2024 17:45	Extension de l'app...	13 Ko
WSConvertisseur.exe	09/09/2024 17:45	Application	140 Ko
WSConvertisseur.pdb	09/09/2024 17:45	Program Debug D...	24 Ko
WSConvertisseur.runtimeconfig.json	09/09/2024 15:52	JSON File	1 Ko
WSConvertisseur.xml	09/09/2024 17:45	Microsoft Edge H...	1 Ko

3. Indiquer à Swagger où récupérer la documentation XML générée (fichier Program.cs) :

```
builder.Services.AddSwaggerGen()
    doc =>
    {
        //Configure Swagger to use the XML documentation file
        var xmlFile = Path.ChangeExtension(typeof(Program).Assembly.Location, ".xml");
        doc.IncludeXmlComments(xmlFile);
    };
};
```

4. Ajouter des attributs [ProducesResponseType] au niveau des actions indiquant les types de retour. [ProducesResponseType] génère des détails descriptifs de la réponse pour les pages d'aide de l'API. Il indique les types connus et les codes d'état HTTP que l'action doit retourner.

Exemples :

- o Pour GetAll : [ProducesResponseType(200) ]
- o Pour GetById : [ProducesResponseType(200) ] et [ProducesResponseType(404) ]
- o Pour Post : [ProducesResponseType(201) ] et [ProducesResponseType(400) ]
- o Etc.

Exemple :

```
/// <summary>
/// Get a single currency.
/// </summary>
/// <returns>Http response</returns>
/// <param name="id">The id of the currency</param>
/// <response code="200">When the currency id is found</response>
/// <response code="404">When the currency id is not found</response>
// GET api/<DevisesController>/5
```

```
[HttpGet("{id}", Name = "GetDevise")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<Devise> GetByld(int id)
```

5. Exécuter l'API. Pour avoir accès à la documentation Swagger : <http://<host>:<port>/swagger/>

## WSConvertisseur 1.0 OAS3

<https://localhost:7223/swagger/v1/swagger.json>

### Devises

<b>GET</b>	/api/Devises	^
<b>POST</b>	/api/Devises	▼
<b>GET</b>	/api/Devises/{id} Get a single currency.	▼
<b>PUT</b>	/api/Devises/{id}	▼
<b>DELETE</b>	/api/Devises/{id}	▼

### WeatherForecast

<b>GET</b>	/WeatherForecast	^
------------	------------------	---

### Schemas

```
Devise ▶ {
    id          integer($int32)
    nomDevise*  string
    taux        number($double)
}
```

GET :

The screenshot shows the Swagger UI interface for the /api/Devises API. It includes:

- GET /api/Devises**: A general GET operation.
- POST /api/Devises**: A POST operation.
- GET /api/Devises/{id}**: A specific GET operation to get a single currency by its ID. It includes a parameter table for 'id' (required, integer, path) and a 'Try it out' button.
- Responses** section:
  - 200**: Description: "When the currency id is found". Media type: text/plain. Example value: A JSON object with fields 'id', 'nomDevise', and 'taux'.
  - 404**: Description: "When the currency id is not found". Media type: text/plain. Example value: A JSON object with fields 'type', 'title', 'status', and 'detail'.

## 2. Réalisation des tests unitaires

Une API doit toujours être testée : vous devez au minimum réaliser des tests unitaires.

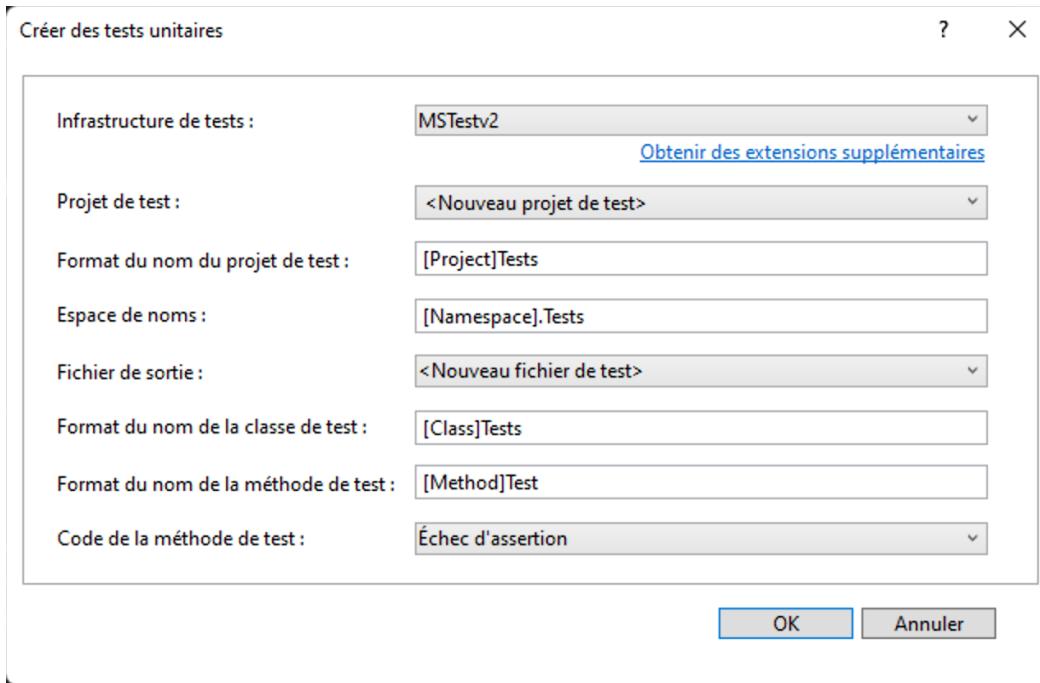
Dans notre cas, il n'est pas utile de tester la couche *Model* car elle ne contient qu'une classe métier simple. Remarquez, que dans le cas d'une couche *Model* générée via un ORM tel que Entity Framework Core, on ne crée pas non plus de test.

Nous allons donc uniquement tester le contrôleur `DevisesController`.

Cliquez avec le bouton droit de la souris sur la méthode `GetAll` de l'API puis *Créer des tests unitaires*.

```
// GET: api/<DevisesController>
[HttpGet]
public IEnumerable<Devise> GetAll()
{
    return devises;
}

/// <summary>
/// Get a single currency.
/// </summary>
/// <returns>Http response</returns>
/// <param name="id">The id of the currency</param>
/// <response code="200">When the currency id is found</response>
/// <response code="404">When the currency id is not found</response>
// GET api/<DevisesController>
[HttpGet("{id}", Name = "GetDe")
[ProducesResponseType(200)]
[ProducesResponseType(404)]
```



## Tests de la méthode GetById

Il faut (au moins) créer un test par type de retour (erreur 404, Devise).

- 1<sup>er</sup> test qui retourne un objet Devise :

```
[TestMethod]
public void GetById_ExistingIdPassed_ReturnsRightItem()
{
    // Arrange
    DevisesController controller = new DevisesController();

    // Act
    var result = controller.GetById(1);

    // Assert
    Assert.IsInstanceOfType(result, typeofActionResult<Devise>), "Pas un ActionResult"); // Test du type de retour
    Assert.IsNotNull(result.Result, "Erreur est pas null"); //Test de l'erreur
    Assert.IsInstanceOfType(result.Value, typeof(Devise), "Pas une Devise"); // Test du type du contenu (valeur) du retour
    Assert.AreEqual(new Devise(1, "Dollar", 1.08), (Devise?)result.Value, "Devises pas identiques"); //Test de la devise
    récupérée
}
```

On remarque l'utilisation du type `var` qui est un type implicite. Il est courant dans un test d'API (mais aussi dans tout test bien écrit) de ne pas connaître le type de retour suite à l'exécution de la méthode (action du contrôleur) à tester (Act) et ensuite de tester ce type de retour via une assertion (1<sup>er</sup> assert). `var` permet de déclarer une variable locale implicitement typée mais qui est fortement typée comme si vous aviez déclaré le type vous-même, mais c'est en réalité le compilateur qui détermine le type.

Plus de détails ici :

<https://learn.microsoft.com/fr-fr/dotnet/csharp/language-reference/statements/declarations>

Générer la méthode `Equals` dans la classe `Devise`.

Exécuter le test (vous pouvez aussi activer le *Live Unit testing*).

*Mais comment avons-nous fait pour connaître les assertions à écrire ?*

- Mettre un point d'arrêt sur la 1<sup>ère</sup> assertion (ou utiliser un espion sur `result`) :
 

```
Assert.IsInstanceOfType(result, typeofActionResult<Devise>), "Pas un ActionResult");
```
- Menu Test > Déboguer tous les tests
- Résultat :

```

// Act
var result = controller.GetById(1);
// Assert
Assert.Is(result, null, "Pas un ActionResult");
Assert.Is(result.Result, null, "Pas une Devise"); // Test du type
Assert.AreEqual(result.Result.Value.Id, 1, "Devises pas identiques");
Assert.AreEqual(result.Result.Value.NomDevise, "Dollar", "NomDevise");
Assert.AreEqual(result.Result.Value.Taux, 1.08, "Taux");
Assert.AreEqual(result.Result.Value.id, 1, "id");
Assert.AreEqual(result.Result.Value.nomDevise, "Dollar", "nomDevise");
Assert.AreEqual(result.Result.Value.taux, 1.08, "taux");

[TestMethod]
public void GetById_UnknownGuidPassed_ReturnsNotFoundResult()

```

On voit que :

- Le type de result est ActionResult<Devise> (Microsoft.AspNetCore.Mvc.ActionResult), d'où le test :  
`Assert.Is(result, null, "Pas un ActionResult");`  
 Comme indiqué plus haut, on voit bien que la variable est fortement typée. Sauf que c'est le compilateur qui a déterminé ce type.
- Que cette variable contient 2 properties :
  - Result qui contiendra l'erreur si une erreur est retournée (et donc qui ici est null), d'où le test :  
`Assert.IsNull(result.Result, "Erreur est pas null"); //Test de l'erreur`
  - Value qui contient la devise retournée, d'où les 2 tests :  
`Assert.Is(result.Result.Value, null, "Pas une Devise");`  
`Assert.AreEqual(result.Result.Value.Id, 1, "Devises pas identiques");`

#### • 2<sup>nd</sup> test pour l'erreur 404 NotFound :

Procéder de la même façon. Créer 3 assertions :

- Test du type de retour de result qui doit toujours être ActionResult<Devise>
- Test du type de retour de result.Result qui doit cette fois être du type NotFoundResult en raison de l'erreur levée.
- Test de result.Value qui doit cette fois être null.

Remarque : normalement ces indications ne devraient pas vous être utiles si vous mettez un point d'arrêt et regardez le contenu de la variable result.

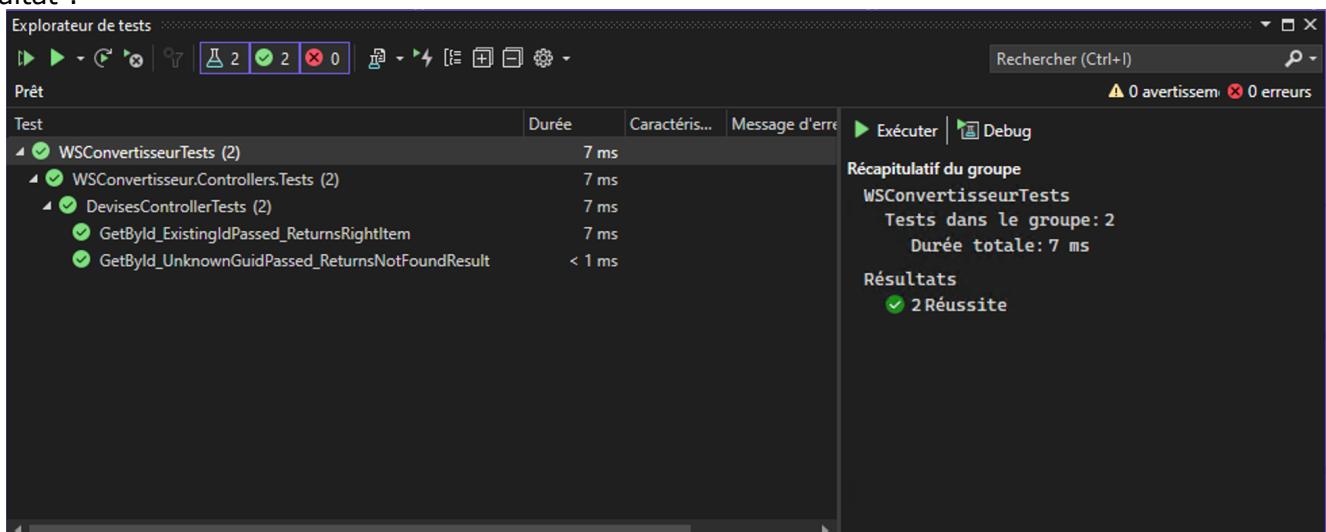
On peut éventuellement rajouter, même si ce n'est pas vraiment utile, le code suivant dans la section Assert qui permet de bien déterminer le code d'erreur testé :

```
var errorResult = result.Result as NotFoundResult;
Assert.AreEqual(errorResult.StatusCode, StatusCodes.Status404NotFound, "Pas 404");
```

Lignes que l'on peut simplifier de la façon suivante :

```
Assert.AreEqual(((NotFoundResult)result.Result).StatusCode, StatusCodes.Status404NotFound, "Pas 404");
```

Résultat :



## Autres tests

GetAll

Vincent COUTURIER

Un seul test, qui teste la liste de devises retournée, composé de 2 assertions :

- Teste le type de retour : `IEnumerable<Devise>`
- Teste le contenu de `result` (les 3 devises). Vous devrez transformer l'`IEnumerable` en `List` (`result.ToList()`) et utiliser la classe `CollectionAssert` (pour tester les 3 devises retournées) : <https://msdn.microsoft.com/fr-fr/library/microsoft.visualstudio.testtools.unittesting.collectionassert.aspx>

Post

Un test qui réussit :

- Ajouter un point d'arrêt et bien regarder ce qui est retourné lors de l'appel de l'action `Post` du contrôleur. Cette fois la valeur de la devise créée et retournée est stockée dans `result.Result`
- Ajouter l'assertion permettant de tester le type de `result`
- Ajouter l'assertion permettant de tester le type de `result.Result` (type : `CreatedAtRouteResult`)
- Caster ensuite `result.Result` pour pouvoir y accéder et tester son contenu (`StatusCode` et `Value`) : `CreatedAtRouteResult routeResult = (CreatedAtRouteResult)result.Result;`
- Ajouter l'assertion permettant de tester la propriété `StatusCode` de `routeResult` (valeur : `StatusCodes.Status201Created`)
- Ajouter l'assertion permettant de tester la devise retournée et stockée dans la propriété `Value` de `routeResult`

Une fois que ce code de test fonctionne, le dupliquer (ne pas oublier la annotation `[TestMethod]`). Modifier le nom du test. La devise passée n'aura pas de nom (par ex. `Devise devise = new Devise(4, null, 4.0)`) afin de pouvoir tester l'erreur `BadRequest` résultant d'un modèle de devise non valide (le nom de la devise est requis en raison de l'annotation `[Required]`). Lancer l'exécution du test.

Vous verrez que le test passe, alors qu'une erreur `BadRequest` aurait dû être levée. En effet, les contraintes sur le modèle ne sont vérifiées qu'à l'exécution de l'API (quand on fait un test unitaire, l'application testée n'est jamais exécutée) car la validation de l'état du modèle n'est déclenchée que pendant l'exécution. Un test unitaire ne peut donc pas tester un modèle non valide. Nous verrons ultérieurement comment réaliser ce test de modèle non valide.

Explorateur de tests				
Série de tests achevée : 5 tests (5 réussi(s), 0 non réussi(s), 0 ignoré(s)) exécutés en 75 ms				
Test	Durée	Caractéris...	Message d'erreur	
WSConvertisseurTests (5)	11 ms			
WSConvertisseur.Controllers.Tests (5)	11 ms			
DevisesControllerTests (5)	11 ms			
GetAll_ReturnsRightsItems	8 ms			
GetById_ExistingIdPassed_ReturnsRightItem	1 ms			
GetById_UnknownGuidPassed_ReturnsNotFoundResult	< 1 ms			
Post_InvalidObjectPassed_ReturnsBadRequest	2 ms			
Post_ValidObjectPassed_ReturnsObject	< 1 ms			

Mettre en commentaire ce dernier test qui n'est pas fonctionnel.

Put

Réaliser 3 tests unitaires permettant de tester les 3 retours suivants :

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState); // PAS TESTABLE
}
if (id != devise.Id)
{
    return BadRequest();
}
int index = devises.FindIndex((d) => d.Id == id);
if (index < 0)
{
    return NotFound();
}
devises[index] = devise;
return NoContent();
```

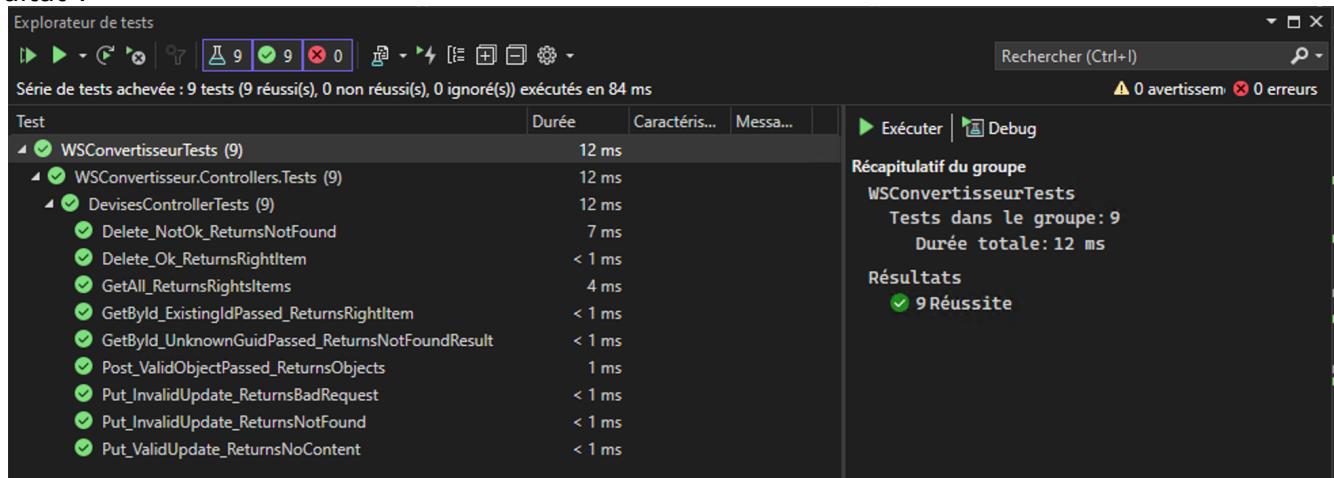
Comme précédemment, il ne sera pas possible de tester le retour `BadRequest` dans le cas d'un modèle non valide (première condition).

*Indication : utiliser le point d'arrêt, comme précédemment.*

Delete

A vous de réfléchir...

Résultat :



## Refactoring du code

Comme la partie « Arrange » est dupliquée (instanciation du contrôleur), refactoriser le code en mettant la ligne dupliquée dans une méthode [TestInitialize]

*Rappel : on peut ajouter des méthodes [TestInitialize] et [TestCleanup].*

```
[TestInitialize]
public void InitialisationDesTests()
{
    // Rajouter les initialisations exécutées avant chaque test
}

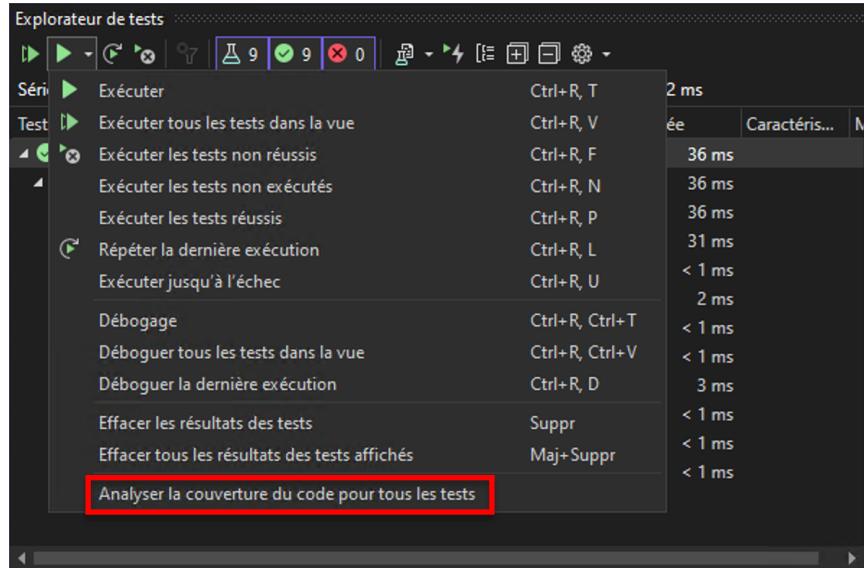
[TestMethod]
public void MonTest()
{
    // test à faire
}

[TestCleanup]
public void NettoyageDesTests()
{
    // Nettoyer les variables, ... après chaque test
}
```

*Rappel : ces méthodes sont exécutées lors de chaque test.*

## Couverture du code

Voilà la couverture que l'on obtient sur le contrôleur Devise, une fois tous les tests codés :

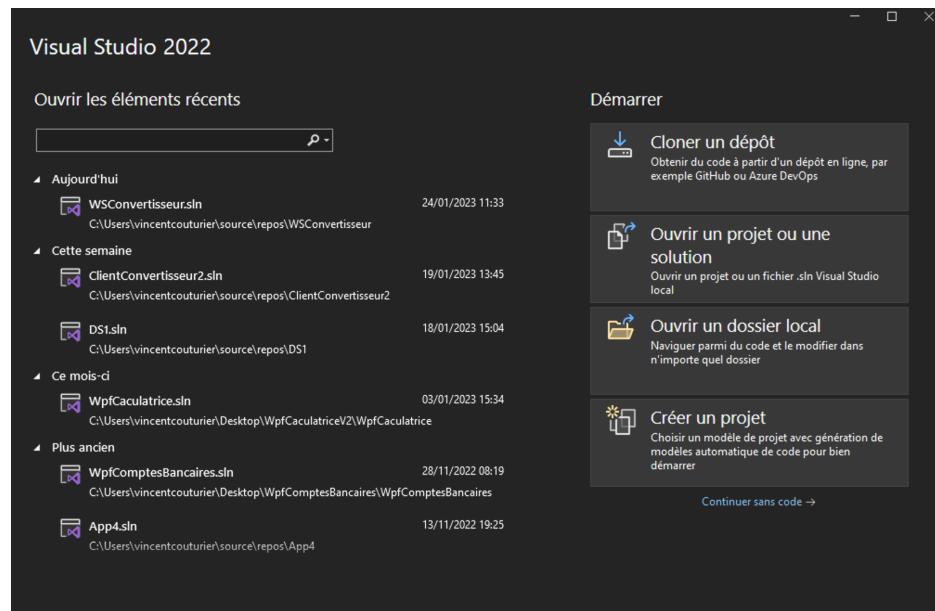


Hiérarchie	Couverts (blocs)	Non couverts (blocs)	Couvertes (Lignes)	Partiellement couv.
vcout_D101-S01_2024-09-09.18_05_11.coverage	177	44	121	5
wsconvertisseur.dll	91	44	59	5
{ } Classes globales	0	28	0	0
{ } WSConvertisseur.Models	29	9	14	5
{ } WSConvertisseur.Controllers	62	7	45	0
DevisesController	56	7	42	0
DevisesController()	9	0	7	0
GetAll()	2	0	3	0
GetById(int)	9	0	7	0
Post(WSConvertisseur.Models.Devise)	10	4	5	0
Put(int, WSConvertisseur.Models.Devise)	16	3	12	0
Delete(int)	10	0	8	0
DevisesController.<>c_DisplayClass3_0	2	0	1	0
DevisesController.<>c_DisplayClass5_0	2	0	1	0
DevisesController.<>c_DisplayClass6_0	2	0	1	0
wsconvertisseurtests.dll	86	0	62	0

Les actions *Get* (méthodes *GetAll* et  *GetById*) et *Delete* sont pleinement testées. Ce qui n'est pas le cas des actions *Put* et *Post*.

### 3. ANNEXE : cloner un dépôt

Pour récupérer votre code situé sur un dépôt GitHub, lancer Visual Studio puis choisir « Cloner un dépôt ».



Sélectionner le dépôt et définir le « Local path » **sur le bureau** :

