Créer un client Console REST avec .NET core et C#

Dossier : « ...\Dev\esProjetsC#\WS_RestClient »

On va générer une application qui émet des requêtes HTTP vers un service REST sur GitHub.

On va lire des informations au format JSON et convertir ce paquet JSON en objets C#.

Enfin, on verra comment travailler avec les objets C#.

1. Création de l'application en mode console :

Ouvrez un invite de commandes et se placer dans le dossier des projets C#

- Tapez la commande `CLI .NET` (qui fait partie de la librarire `CLI .NET core`) suivante dans la fenêtre de console :

  ```
  dotnet new console --name WebApiClientRest
  ```

  Ceci va créer un fichier **program.cs** et aura pour effet :

  - de télécharger l'infrastructure .Net core,
  - et d'exécuter le gestionnaire de package NuGet pour restaurer les dépendances manquantes

  ```
  namespace WebApiClientRest
  {
      class Program
      {
          static void Main(string[] args)
          {
              Console.WriteLine("Hello World!");
          }
      }
  }
  ```

- Entrez dans le dossier créé et tapez la commande `CLI .NET` suivante dans la fenêtre de console : `dotnet run`

  Cette commande exécutera automatiquement la commande « `dotnet restore` » s'il manque des dépendances dans votre environnement. Il effectue également `dotnet build` si l'appli doit être régénérée.

2. Ajout de nouvelles dépendances :

L'un des objectifs de conception clés de .NET Core consiste à réduire la taille de l'installation .NET.

Si une application a besoin de bibliothèques supplémentaires pour certaines de ses fonctionnalités, vous ajoutez ces dépendances dans votre fichier projet (*.csproj) en C#.

Pour notre exemple, vous devez ajouter le package **`System.Runtime.Serialization.Json`** afin que votre application puisse traiter les réponses **JSON**.

Ajoutez ce package à votre projet en exécutant la commande CLI .NET suivante :

```
dotnet add package System.Text.Json
```

3. Ouverture du projet sous Visual Studio :

Lancer Visual Studio et ouvrez votre projet.

a) Effectuer des requêtes web :

Nous allons récupérer des données à partir du web, plus exactement à partir de l'API GitHub.

Nous allons lire des informations sur les projets couverts par la .NET Foundation.

Pour cela, nous allons utiliser le point de terminaison https://api.github.com/orgs/dotnet/repos. pour récupérer toutes les informations sur ces projets.

Nous allons donc utiliser une requête HTTP GET.

Avant de commencer, tester la dispo de ce web service Rest sur le navigateur qui utilise la même requête HTTP GET en tapant le lien précédent pour voir la forme des infos que nous allons récupérer.

Etape 1 : préparation de la méthode asynchrone

Pour effectuer des requêtes web, nous utiliserons la classe **HttpClient** qui fait partie de l'API .NET

Comme toutes les API .NET modernes, cette classe prend en charge uniquement les méthodes asynchrones.

Créez une méthode asynchrone dans la classe **program.cs** comme suit :

```
private static async Task ProcessRepositories()
{
}
```

Vous devez rajouter la ligne suivante aussi : `using System.Threading.Tasks;`

La méthode ProcessRepositories va afficher un avertissement car elle ne contient pas d'opérateur `await`.

Modifiez la signature de la méthode **Main** en rajoutant aussi l'appel à la méthode ProcessRepositories () :

```
static async Task Main(string[] args)
{
   await ProcessRepositories();
}
```

Avec le « await », la méthode `ProcessRepositories()` retournera une tâche, et le programme ne se terminera pas avant que cette tâche soit totalement terminée.

A ce niveau là, le programme ne fait rien, mais il le fait de façon asynchrone.

Exécutez le programme.

Etape 2 : Envoi de la requête et récupération d'une réponse (des données)

Maintenant, on va utiliser l'objet **HttpClient** , qui gère les demandes et réponses, pour récupérer les données à partir du web.

Instanciez cet objet de façon unique dans la classe **Program** comme suit :

```
private static readonly HttpClient client = new HttpClient();
```

Implémentez la méthode `ProcessRepositories()` :

On effectue une requête web pour lire la liste de tous les dépôts de l'organisation *DotNet Foundation*. (L'ID GitHub de la .NET Foundation est 'dotnet')

```
client.DefaultRequestHeaders.Accept.Clear();

// on configure le client http pour accepter les json du GitHub
client.DefaultRequestHeaders.Accept.Add(
   new MediaTypeWithQualityHeaderValue("application/vnd.github.v3+json")
);

// on ajoute un en-tête d'agent utilisateur à toutes les requêtes à partir de cet
objet client http
client.DefaultRequestHeaders.Add("User-Agent", ".NET Foundation Repository Reporter");

// On effectue une requête web et on récupère la réponse
// Cette méthode démarre une tâche qui effectue la requête web :
      - Envoie la demande,
      - Lit le flux de réponse
      - puis, extrait le contenu à partir du flux.
var stringTask = client.GetStringAsync("https://api.github.com/orgs/dotnet/repos");

// La chaîne message est disponible lorsque la tâche est terminée
var message = await stringTask;

Console.Write(message);
```

Il faudra ajouter les deux lignes suivantes en haut du fichier pour utiliser les librairies associées :

```
using System.Net.Http;
using System.Net.Http.Headers;
```

NB : Les deux lignes du code qui viennent après le Clear() sont des en-têtes et sont nécessaires pour récupérer des informations à partir de GitHub. Elles sont vérifiées par le code du serveur GitHub.

b) Traitement du résultat JSON

Dans ce qui suit, on va convertir la réponse récupérée JSON en objets C#.

On utilisera la classe `System.Text.Json.Serialization` qui sérialise les objets au format JSON et désérialise JSON en objets.

Etape 1 : On va créer le type de classe qu'on associera au sérialiseur JSON.

Pour cela, on crée une classe **Repository** , dans un fichier **repo.cs**, pour représenter l'objet JSON retourné à partir de l'API GitHub.

```
namespace WebAPIClientRest
{
    public class Repository
    {
        public string name { get; set; }
    }
}
```

Tous les champs dans le paquet JSON font partie de cette classe (ici on a utilisé que le champ name). Si des champs n'existent pas dans la classe, alors le sérialiseur JSON va les ignorer.

Etape 2 : On va procéder à la désérialisation du JSON en objets C#.

Ajoutez à la fin de la méthode `ProcessRepositories()` les lignes suivantes :

```
var streamTask = client.GetStreamAsync("https://api.github.com/orgs/dotnet/repos");
var message = await streamTask;
//récupération des noms des espaces de stockage du repo
var repositories = await JsonSerializer.DeserializeAsync<List<Repository>>(message);
```

Notez qu'on utilise la méthode `GetStreamAsync()` au lieu de `GetStringAsync()` vu dans le cas précédent.

Il faudra ajouter les deux lignes suivantes en haut du fichier pour utiliser les librairies associées :

```
using System.Collections.Generic;
using System.Text.Json;
```

Remplacez les deux lignes suivantes

```
var message = await stringTask;
Console.Write(message);
```

Par

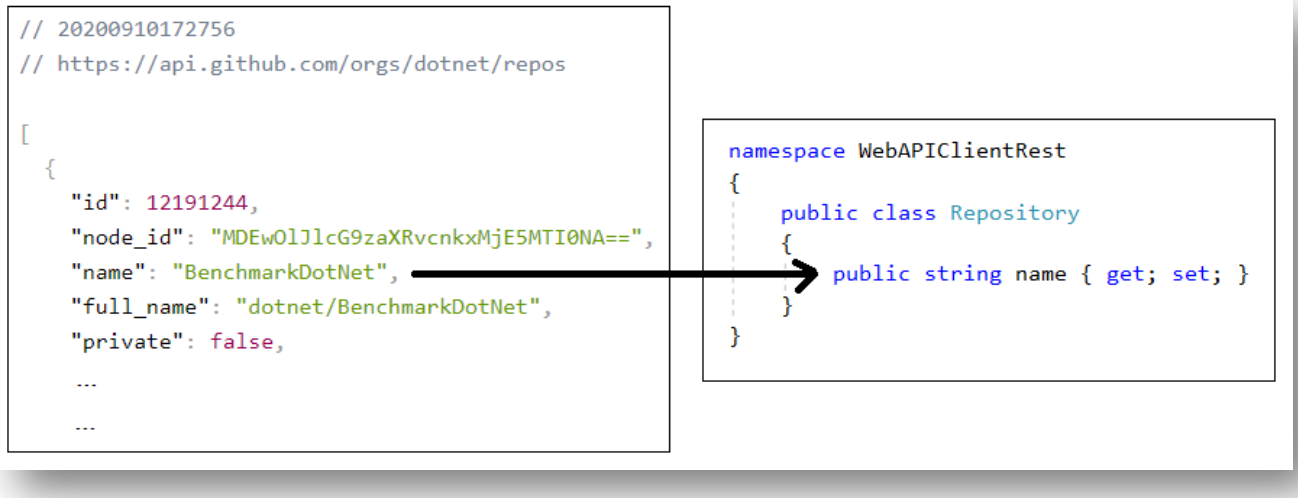
```
foreach (var repo in repositories)
   Console.WriteLine(repo.name);
```

Exécutez le code et observez le résultat.

c) Contrôle de la sérialisation

Etape 1 : Utilisons d'abord l'attribut [JsonPropertyName] pour choisir nous même nos variables.

Pour déclarer les variables devant contenir les données récupérées du Web Service REST, nous ne sommes pas obligés d'utiliser les mêmes noms que ceux fournis dans le résultat de retour du webService :



Pour cela, on va modifier la propriété « name » à l'aide de l'attribut [JsonPropertyName].

Modifiez la propriété « name » de la classe **Repository** comme suit :

```
[JsonPropertyName("name")]
public string Name { get; set; }
```

Pour pouvoir utilise cet attribute, il faudra ajouter le using suivant en haut du fichier :

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

Pour finir, puisqu'on a modifié l'attribut « name » en « Name », il faudra alors modifier le même attribut dans la classe **Program** comme suit :

```
Console.WriteLine(repo.Name);
```

Exécutez le code, on devrait avoir le même résultat que dans l'étape précédente [b) Etape 2].

Etape 2 : Transformons les données reçus du WebService en une liste d'objets C#.

La méthode `ProcessRepositories` peut renvoyer une collection des espaces de stockage toujours de manière asynchrone. On va donc renvoyer une tâche dont le résultat est une liste d'objets Repository ⇒ `List<Repository>`.

On va modifier la classe **Program** :

- On modifie le retour de la méthode comme suit :

```
private static async Task<List<Repository>> ProcessRepositories();
```

- On modifie la désérialisation des objets JSON en liste d'objets Repository :

```
//récupération des noms des espaces de stockage du repo
var streamTask = client.GetStreamAsync("https://api.github.com/orgs/dotnet/repos");
//Désérialisation du JSON reçu en des objets C#
var repositories = await JsonSerializer.DeserializeAsync<List<Repository>>(await streamTask);
return repositories;
```

On va modifier le **Main** comme suit :

```
var repositories = await ProcessRepositories();

foreach (var repo in repositories)
    Console.WriteLine(repo.Name);
```

Exécutez le code.

d) Récupérer plus d'informations des données retournées par le webService :

Nous allons essayer de récupérer une partie, mais pas tout, des données reçus et pour cela on aura besoin de déclarer plus de propriétés dans notre classe **Repository**.

On récupèrera par exemple des types primitifs, des types Uri et des types de DateTime.

Etape 1 : Modifier le code pour récupérer d'autres propriétés de type primitif et type Uri

Ajoutez le code suivant dans la classe **Repository** :

```
[JsonPropertyName("description")]
public string Description { get; set; }

[JsonPropertyName("html_url")]
public Uri GitHubHomeUrl { get; set; }

[JsonPropertyName("homepage")]
public Uri Homepage { get; set; }

[JsonPropertyName("watchers")]
public int Watchers { get; set; }
```

Attention !! si le paquet JSON contient des données qui ne sont pas convertibles dans le type cible, l'action de sérialisation lèvera une exception.

Mettez à jour la méthode Main comme suit :

```
Console.WriteLine(repo.Name);
Console.WriteLine(repo.Description);
Console.WriteLine(repo.GitHubHomeUrl);
Console.WriteLine(repo.Homepage);
Console.WriteLine(repo.Watchers);
```

```
Console.WriteLine();
```

Etape 1 : Modifier le code pour récupérer une propriété de type DateTime

On s'intéresse à la propriété « **pushed_at** » (qui correspond à la dernière opération Push effectuée) qui possède le format suivant : `2016-02-08T21:27:00Z`

Ce format est au format de temps universel (UTC)

NB : Si on préfère une date représentée dans notre fuseau horaire, on devra écrire notre propre méthode de conversion.

Ajoutez la propriété dans la classe **Repository** :

```
[JsonPropertyName("pushed_at")]
public DateTime LastPushUtc { get; set; }

public DateTime LastPush => LastPushUtc.ToLocalTime();
```

Mettez à jour la méthode Main comme suit :

```
...
...
Console.WriteLine(repo.LastPush);
Console.WriteLine();
```

Exécutez le code.