



NuGet et REST C#

date	révision
novembre 2018	Création (v.13)



SOMMAIRE

1	Outil de paquetage Microsoft.....	4
1.1	Présentation de NuGet.....	4
1.2	Installation de NuGet.....	4
1.3	Utilisation NuGet.....	5
1.3.1	Nuget derrière un proxy.....	5
2	Paquetage Newtonsoft.Json.....	6
2.1	Sérialisation et dé-sérialisation.....	6
2.2	Installation.....	7
3	Application simple REST.....	9
3.1	Création de l'interface graphique :.....	10
3.2	Code de lecture JSON.....	11
3.3	Explications.....	12
3.3.1	Importation des bibliothèques (librairies).....	12
3.3.2	Communication simulant les requêtes d'un navigateur.....	12
3.3.3	En résumé.....	13
4	Annexes.....	14
4.1	Sources.....	14
4.2	Proxy.....	14



ARCHITECTURE REST EN C#

L'activité de ce module est l'usage des données disponibles en REST dans une application lourde en C#.

À l'issu de cette activité, l'étudiant devra :

- Connaître l'outil de paquetage de Microsoft
- Comprendre l'architecture et les communications REST
- Mettre en œuvre un projet utilisant C# et Json.NET

<p>Durée</p> 	<p>Difficulté</p> 	 <p>Taxonomie Bloom</p>
<p>2 séances</p>	<p>① ② ③ ④</p>	<p>Compréhension – Application</p>



1 OUTIL DE PAQUETAGE MICROSOFT

L'utilisation générale des "packages" ou paquetages dans la plupart des langages répond aux attentes des développeurs qui souhaitent créer et partager puis exploiter des codes utiles.

Chez Microsoft, les DLL (Dynamic Link Library) sont régulièrement utilisées pour étendre les capacités des outils développés dans l'environnement Windows.

Exemple : la librairie LAME encodeur MP3 est souvent repris pour permettre aux applications multimédias de sauvegarder les fichiers audios dans le format MP3 : <http://lame.sourceforge.net/>

1.1 PRÉSENTATION DE NUGET

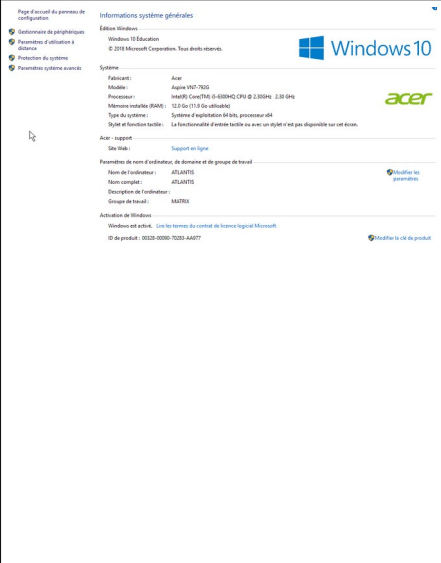
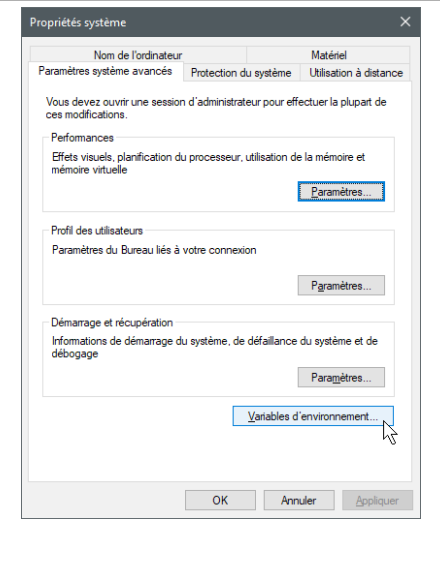
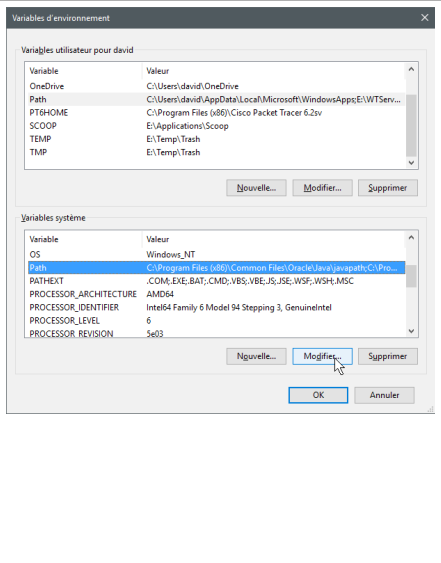
Dans le cas de .NET, le mécanisme de partage géré par Microsoft est NuGet (prononcer "new get").

Pour simplifier, un paquetage NuGet est un fichier ZIP avec sa propre extension, **.nupkg**, qui contient le code compilé au format DLL. D'autres fichiers sont joints, comme le manifeste qui décrit les informations d'utilisation et les versions du paquetage.

1.2 INSTALLATION DE NUGET

Il faut se rendre sur le site <https://www.nuget.org/downloads> pour y télécharger le programme.

Il s'agit d'un exécutable sans installation : il faut donc le placer dans un répertoire facilement accessible, et modifier le chemin de l'environnement (variable PATH de l'OS Microsoft).

		
<p>Panneau d'accueil de configuration : [Win]+[Pause]</p>	<p>Paramètres système avancés</p>	<p>Variables d'environnement (modifier le PATH dans "Variables système")</p>



1.3 UTILISATION NUGET

Si vous avez correctement configuré votre variable PATH en ajoutant le chemin d'accès au répertoire contenant NuGet.exe, vous devez maintenant pouvoir y accéder depuis une invite de commande :

Invite de commande DOS (cmd)	Invite de commande PowerShell

Accessible depuis n'importe où...

1.3.1 Nuget derrière un proxy

Dans le cas particulier d'un proxy entre Internet et le poste de travail, il est possible de définir les paramètres pour pouvoir l'utiliser :

```
nuget.exe config -set http_proxy=http://my.proxy.address:port
nuget.exe config -set https_proxy=http://my.proxy.address:port
```

Il est aussi possible de trouver les paramètres dans le fichier *NuGet.config* dans le répertoire où se trouve le programme nuget.exe.

Désormais, vous connaissez l'équivalent des outils RPM ou APT pour Windows : cependant, pour le moment ce mode d'installation reste marginal et nous allons voir que pour un projet, une installation locale est un peu plus facile (ce qui ne favorise pas l'emploi de Nuget).

2 PAQUETAGE NEWTONSOFT.JSON

Le paquetage Newtonsoft.Json s'installe normalement, à partir de l'outil NuGet.

Il s'agit d'un paquetage dédié à la gestion des données au format JSON, actuellement en version 11.0.2 !

L'intérêt de ce paquetage est notamment de pouvoir sérialiser et dé-sérialiser des données JSON pour créer un objet classique (gérable dans C#).

<https://www.newtonsoft.com/json>

<pre>Product product = new Product(); product.Name = "Apple"; product.Expiry = new DateTime(2008, 12, 28); product.Sizes = new string[] { "Small" }; string json = JsonConvert.SerializeObject(product); // { // "Name": "Apple", // "Expiry": "2008-12-28T00:00:00", // "Sizes": [// "Small" //] // }</pre> <p style="text-align: center;">Serialize JSON ● ○ ○</p>	<pre>string json = @"{ 'Name': 'Bad Boys', 'ReleaseDate': '1995-4-7T00:00:00', 'Genres': ['Action', 'Comedy'] }"; Movie m = JsonConvert.DeserializeObject<Movie>(json); string name = m.Name; // Bad Boys</pre> <p style="text-align: center;">Deserialize JSON ○ ● ○</p>
---	--

La documentation est disponible sur <https://www.newtonsoft.com/json/help/html/Introduction.htm>

2.1 SÉRIALISATION ET DÉ-SÉRIALISATION

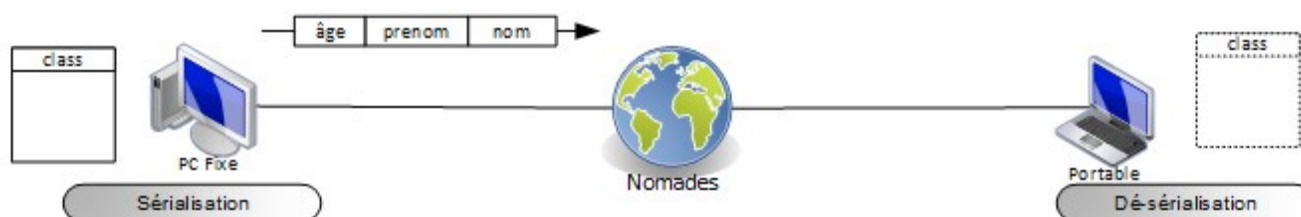
Il s'agit d'un concept important dans la transmission d'information. En effet, les données – stockées sous forme d'objets ou dans des tableaux – ne peuvent pas être transmises sous cette forme dans les flux de communication.

On appelle sérialisation, l'action de transformer un ensemble de données en un flux binaire (la plupart du temps, des octets).

On appelle dé-sérialisation, l'action de transformer un flux binaire en données cohérentes, compatibles avec les tableaux et objets d'une application.

Il convient donc d'établir un protocole de communication : dans quel ordre les objets sont envoyés, ce qu'ils contiennent, etc.

<https://www.supinfo.com/articles/single/6727-serialisation-deserialisation>

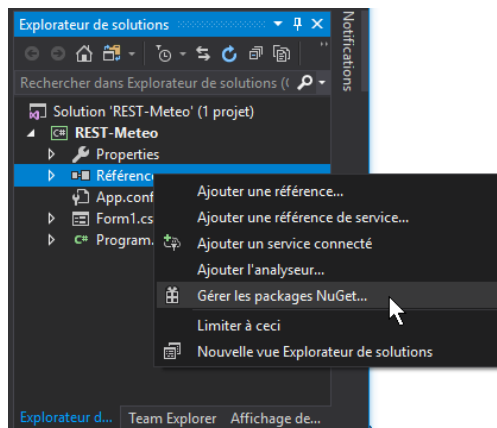




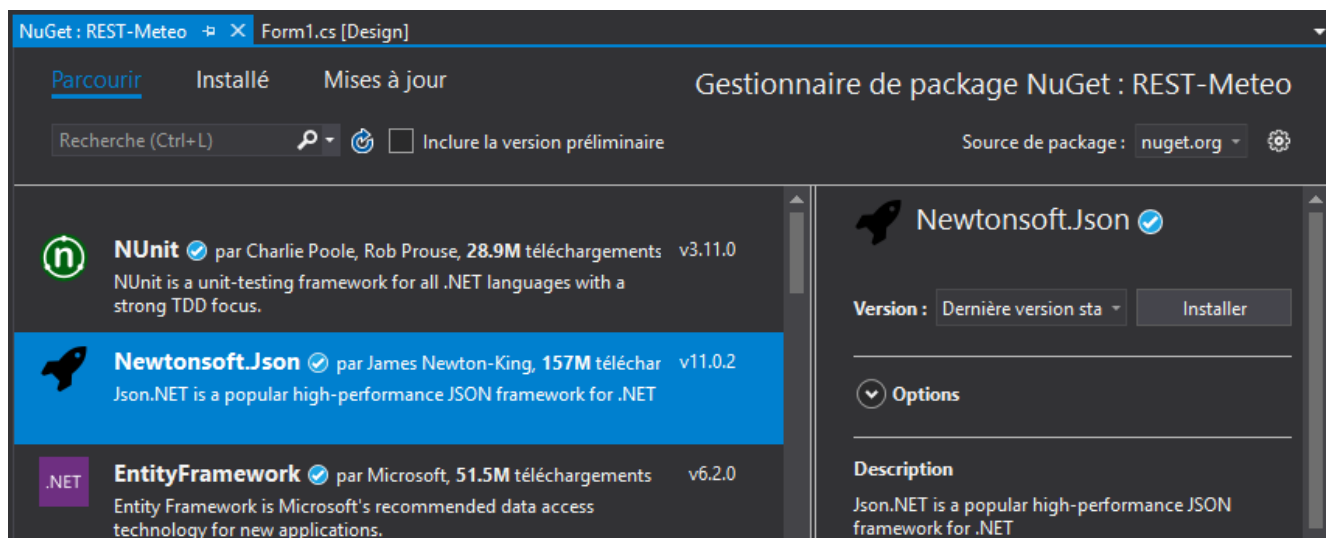
2.2 INSTALLATION

Il n'est pas obligatoire d'utiliser NuGet si vous disposez de Microsoft Visual Studio. Dans ce cas, le paquetage peut être intégré localement, dans une application :

1. Ouvrez Visual Studio 2017
2. Créez un nouveau projet de type Winform (.NET Framework)
3. Dans l'explorateur de solutions > clic droit sur "Références" > Gérer les packages NuGet...

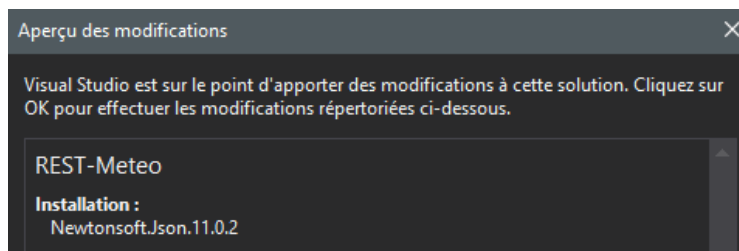


La fenêtre qui s'ouvre contient 3 onglets : Parcourir, Installé et Mises à jour. Sélectionnez [Parcourir]



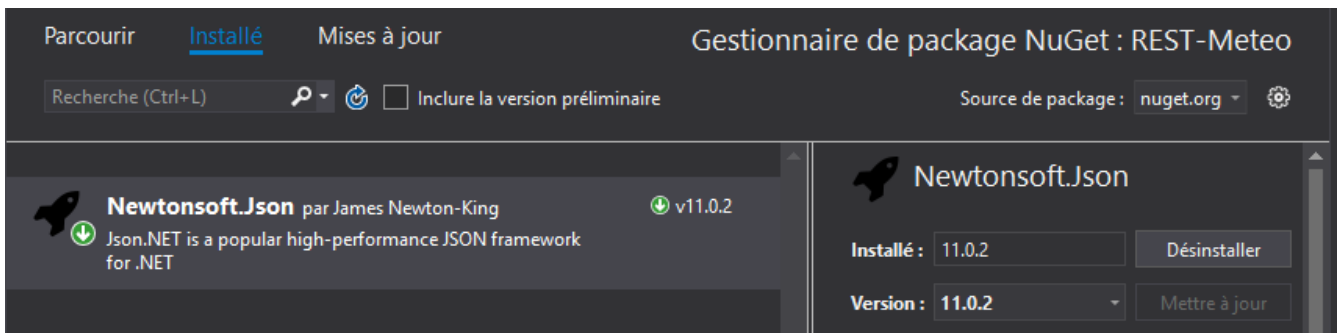
Puis sélectionnez Newtonsoft.Json et cliquez sur le bouton [Installer].

VS2017 avertit de l'installation du paquetage pour le projet en cours.





Vous devriez avoir l'affichage suivant en cliquant sur l'Onglet [Installé].



Vous pouvez fermer la fenêtre (NuGet : *nom-de-votre-application*) et désormais créer le code.



3 APPLICATION SIMPLE REST

L'objectif est de créer une application capable d'afficher les données d'un site ayant une API REST dans une application créée en C#.

URL REST

	userid	id	title	body
▶	1	1	sunt aut facere re...	quia et suscipitsu...
	1	2	qui est esse	est rerum tempor...
	1	3	ea molestias qua...	et iusto sed quo i...
	1	4	eum et est occae...	ullam et saepe rei...
	1	5	nesciunt quas odio	repudiandae veni...
	1	6	dolorem eum mag...	ut aspernatur cor...
	1	7	magnam facilis a...	dolore placeat qu...
	1	8	dolorem dolore es...	dignissimos aperi...
	1	9	nesciunt iure omn...	consectetur animi...
	1	10	optio molestias id ...	quo et expedita ...
	2	11	et ea vero quia la...	delectus reiciendi...
	2	12	in quibusdam tem...	itaque id aut mag...
	2	13	dolorum ut in volu...	aut dicta possimu...
	2	14	voluptatem eligen...	fuga et accusam...
	2	15	eveniet quod tem...	reprehenderit quo...
	2	16	sint suscipit persp...	suscipit nam nisi ...

```
[ { "userid": 1, "id": 1, "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit", "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto" }, { "userid": 1, "id": 2, "title": "qui est esse", "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla" }, { "userid": 1, "id": 3, "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut", "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut" }, { "userid": 1, "id": 4, "title": "eum et est occaecati", "body": "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit" }, { "userid": 1, "id": 5, "title": "nesciunt quas odio", "body": "repudiandae veniam quaerat sunt sed\nnalias aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolor neque" }, { "userid": 1, "id": 6, "title": "dolorem eum magni eos aperiam quia", "body": "ut aspernatur corporis" } ]
```

Le site utilisé (contenant des données fictives sous la forme d'un tableau JSON) est : <https://jsonplaceholder.typicode.com/posts>

3.1 CRÉATION DE L'INTERFACE GRAPHIQUE :

The screenshot shows a window titled "Show REST API result" with a text input field containing the URL "https://jsonplaceholder.typicode.com/posts" and an "Envoyer" button. Below the input is a table with columns: userId, id, title, and body. The table contains 16 rows of data. Below the table is a text area displaying the JSON response for the first row.

userId	id	title	body
1	1	sunt aut facere re...	quia et suscipit...
1	2	qui est esse	est rerum tempor...
1	3	ea molestias qua...	et iusto sed quo i...
1	4	eum et est occae...	ullam et saepe rei...
1	5	nesciunt quas odio	repudiandae veni...
1	6	dolorem eum mag...	ut aspernatur cor...
1	7	magnam facilis a...	dolore placeat qu...
1	8	dolorem dolore es...	dignissimos aperi...
1	9	nesciunt iure omn...	consectetur animi...
1	10	optio molestias id ...	quo et expedita ...
2	11	et ea vero quia la...	delectus reiciendi...
2	12	in quibusdam tem...	itaque id aut mag...
2	13	dolorum ut in volu...	aut dicta possimu...
2	14	voluptatem eligen...	fuga et accusam...
2	15	eveniet quod tem...	reprehenderit quo...
2	16	sint suscipit perso...	suscipit nam nisi...

```
[ { "userId": 1, "id": 1, "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit", "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto" }, { "userId": 1, "id": 2, "title": "qui est esse", "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditis voluptate poro vel nihil molestiae ut reiciendis\ninqui aperiam non debitis possimus qui neque nisi nulla" }, { "userId": 1, "id": 3, "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut", "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae poro eius odio et labore et velit aut" }, { "userId": 1, "id": 4, "title": "eum et est occaecati", "body": "ullam et saepe reiciendis voluptatem adipisci\nvnisit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit" }, { "userId": 1, "id": 5, "title": "nesciunt quas odio", "body": "repudiandae veniam quaerat sunt sed\nnihil aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus quis\nvnisit aut tenetur dolor neque" }, { "userId": 1, "id": 6, "title": "dolorem eum magni eos aperiam quia", "body": "ut aspernatur corporis" } ]
```

Notez que la forme DataGridView sera initialement vide (pas la peine de saisir des données).

Pour gagner du temps, il est utile de placer l'URL <https://jsonplaceholder.typicode.com/posts> dans la propriété 'Text' de la forme txtURL.

Enfin, en double-cliquant sur le bouton [Envoyer] (nom = LireJSON), vous pourrez ajouter le code utile pour lire du JSON.



3.2 CODE DE LECTURE JSON

Dans le fichier Form1.cs (lors du double-clic sur le bouton, c'est ce fichier qui a dû s'ouvrir), il faut ajouter les parties suivantes :

```

/* -----
 * Author : David ROUMANET
 * Date   : 05/11/2018
 * Descr. : Exemple d'utilisation d'une API REST en C# en utilisant Newtonsoft
 * -----
 * v0.1 : Le code ne fonctionne pas. Ajout debug sans succès
 * v0.2 : Changement de méthode (utilisation using + Newtonsoft.Json.Linq. Récup variable texte
 * v0.3 : Tentative de récupération de tableau pour DataGridView...
 * -----
 */
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net;
using System.Windows.Forms;
using System.Diagnostics;

```

puis dans la méthode LireJSON_Click() :

```

private void LireJSON_Click(object sender, EventArgs e) {
    try {
        // la classe HttpWebRequest permet de faire des requêtes comme un navigateur (hérite de
        // WebRequest)
        var webRequest = (HttpWebRequest)WebRequest.Create(txtURL.Text);
        webRequest.Method = "GET";
        webRequest.Accept = "application/json";

        // état de la réponse...
        var webResponse = (HttpWebResponse)webRequest.GetResponse();
        string contenu = string.Empty;

        // using permet de libérer le flux à la fin du traitement (mieux qu'un try-catch-finally)
        using (var flux = webResponse.GetResponseStream()) {
            using (var sr = new StreamReader(flux)) {
                contenu = sr.ReadToEnd();
            }
        }
        Debug.WriteLine("Résultat : {0} et longueur {1}", webResponse.StatusCode,
            webResponse.ContentLength);
        Debug.WriteLine("Contenu " + contenu);
        var arr = JSONArray.Parse(contenu);
        txtBox.Text = contenu;
        // var arr = JsonConvert.DeserializeObject<Newtonsoft.Json.Linq.JSONArray>(contenu);
        dataGridView1.DataSource = arr;
    }
    catch (Exception ex) {
        MessageBox.Show(ex.Message);
    }
}

```



3.3 EXPLICATIONS

3.3.1 Importation des bibliothèques (librairies)

Utiliser les classes et méthodes de NewtonsoftJSON :

```
using Newtonsoft.Json;  
using Newtonsoft.Json.Linq;
```

Utiliser les bibliothèques réseaux de Microsoft pour httpWebRequest par exemple :

```
using System.Net;
```

Utiliser les librairies de lecture de flux (stream) :

```
using System.IO;
```

Utiliser les affichages dans la console pour le débogage (à la compilation finale, cette librairie ne sera pas utilisée) :

```
using System.Diagnostics;
```

3.3.2 Communication simulant les requêtes d'un navigateur

Le code suivant va utiliser les primitives classiques des navigateurs (GET, POST, DELETE, UPDATE) pour communiquer via le protocole HTTP

```
var webRequest = (HttpWebRequest)WebRequest.Create(txtURL.Text);  
webRequest.Method = "GET";  
webRequest.Accept = "application/json";  
  
// état de la réponse...  
var webResponse = (HttpWebResponse)webRequest.GetResponse();  
string contenu = string.Empty;
```

Le format de la première ligne crée une variable de type HttpWebRequest mais effectue la conversion implicite du résultat de WebRequest.Create() qui reçoit comme paramètre le contenu du champ du textbox 'txtURL'.

On peut spécifier dans la variable quelle méthode HTTP utiliser (ici GET) et d'autres paramètres. Cette variable est bien un objet de la classe WebRequest !

Une fois la requête construite, on crée une variable webResponse pour obtenir le résultat de la requête. C# lance la requête.



3.3.3 En résumé

Vous avez maintenant la capacité de manipuler des informations en provenance d'API REST en C#

Cela vous permet de créer des projets mixant les accès directs à une base de données et des accès à des données sur Internet.

C'est utile pour créer une application fonctionnant depuis un client mobile (Android par exemple), un client lourd (C# ou Java par exemple) et une application web (en PHP ou en Angular2)

Vous devriez suivre ce cours : <https://anceret-matthieu.developpez.com/tutoriels/web-api-realisation-bdd/>



4 ANNEXES

4.1 SOURCES

NuGet

<https://docs.microsoft.com/fr-fr/nuget/quickstart/install-and-use-a-package-in-visual-studio>

REST

<https://www.codeproject.com/Tips/712109/How-to-Get-REST-Data-and-Display-it-in-a-DataGridV>

<https://docs.microsoft.com/fr-fr/dotnet/framework/network-programming/how-to-request-data-using-the-webrequest-class>

<https://www.maxpou.fr/rest-crud>

using

<http://www.dotnetdojo.com/instruction-indispensable-en-csharp-using-idisposable/>

4.2 PROXY

Pour pouvoir utiliser un proxy avec C#, voici la méthode recommandée par Microsoft :

```
WebProxy proxyObject = new WebProxy("http://172.16.0.1:3128/");  
GlobalProxySelection.Select = proxyObject;
```

Une autre méthode plus complète est exposé ici :

<http://www.pausedotnet.fr/configurer-proxy-csharp-webrequest/>

Cependant, par défaut, une application .NET est censée utiliser la configuration du proxy utilisée par le système (et Internet Explorer).