

REST Api

Exercice d'introduction

Introduction

Cet exercice va vous permettre de créer une API REST à l'aide de la plateforme .NET qui exploite une base de données. Vous allez ensuite interroger cette API à l'aide de plusieurs clients http. Nous critiquerons ensuite cette dernière API durant la séance.

Plusieurs questions vous sont posées durant l'exercice. Consignez vos réponses.

Mise en place de l'environnement

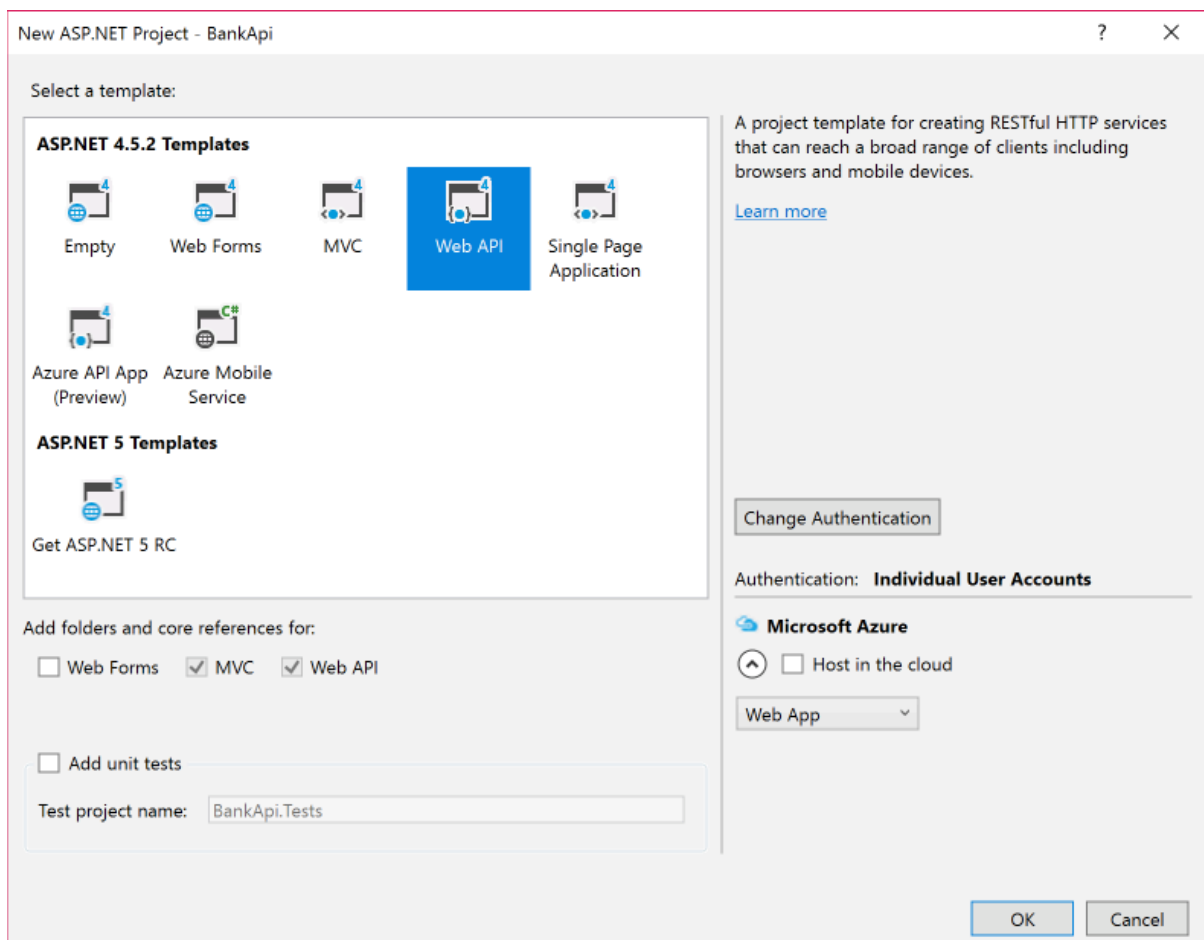
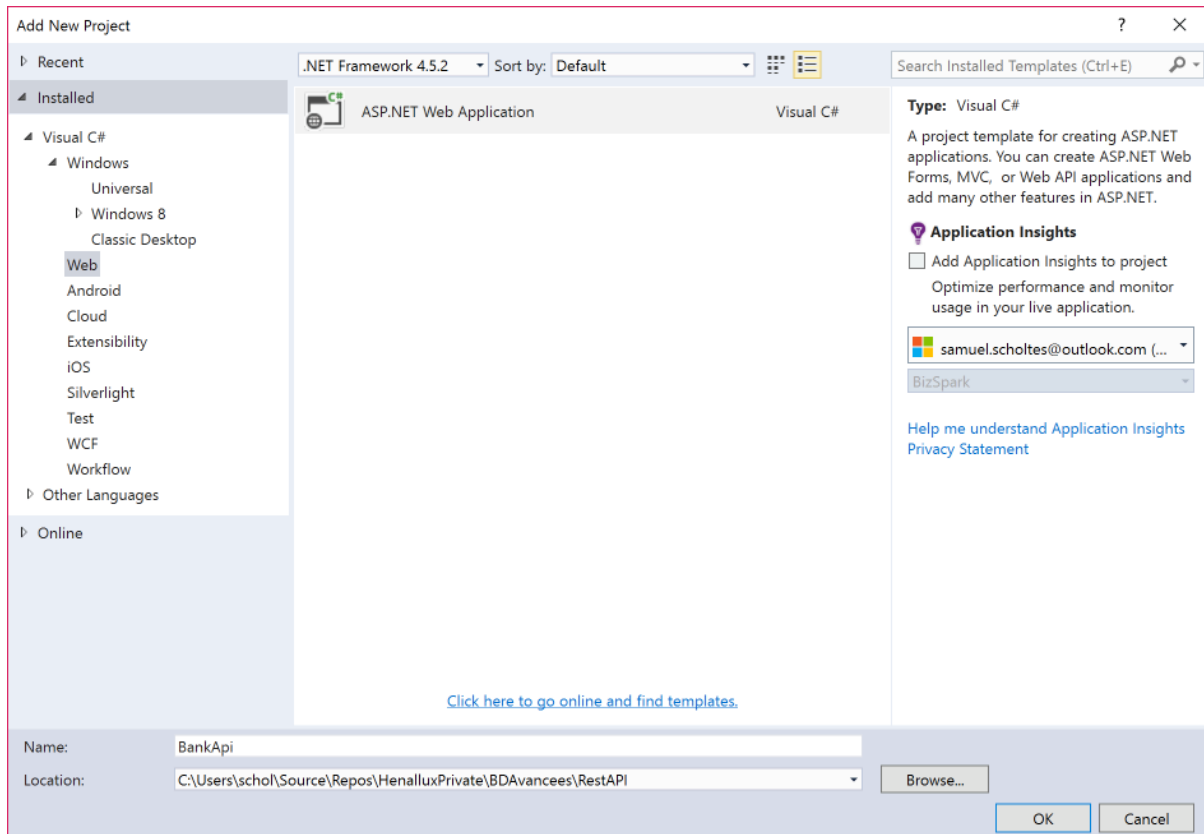
Pour cet exercice, vous ne partirez pas de rien. La couche d'accès aux données a été créée afin de faciliter votre tâche. Elle est conçue en .NET (Entity framework code first) et est disponible dans une solution Visual Studio sur le repository GitHub habituel (<https://github.com/samiroquai/Henallux.git>) dans la solution RestApiDemo. Commencez par en récupérer les sources et copiez-les dans votre repository local, afin de pouvoir sauvegarder votre travail (commit et push) dans ce dernier à la fin de la séance.

Le modèle exploité est une évolution de celui utilisé pour le laboratoire sur les éditions concurrentes.

Compilez et lancez les tests associés. La base de données devrait être créée. Dans la base de données créée, insérez plusieurs clients et plusieurs comptes en banque liés à ces derniers.

Création d'un projet Web API

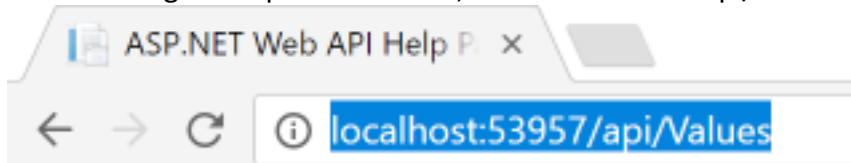
Ajoutez un nouveau projet à votre solution, ce projet sera du template « Web » => « Web Application » => Web Api. Voir screenshots ci-dessous. Définissez ce nouveau projet comme étant celui de démarrage.



Le projet créé contient une structure particulière. Prêtez une attention particulière au dossier « Controllers ». Ce dossier doit exister dans votre solution car ASP.NET Web API utilise par défaut des conventions pour router les requêtes http (voir plus loin) et ces conventions seront cassées si le dossier Controllers n'est pas présent.

Lancez votre application. Visual Studio va lancer un serveur Web de développement qui va écouter les requêtes sur un port correspondant à celui repris dans l'URL visible dans la barre d'adresse du navigateur. Ce numéro de port est auto-attribué. **Pouvez-vous retrouver le serveur web de développement ? Comment se manifeste-t-il ? Quel est son nom ?**

Dans le navigateur qui s'est ouvert, suffixez l'url de « Api/Values ».



Vous devriez obtenir une réponse vous signifiant que votre requête n'est pas autorisée.

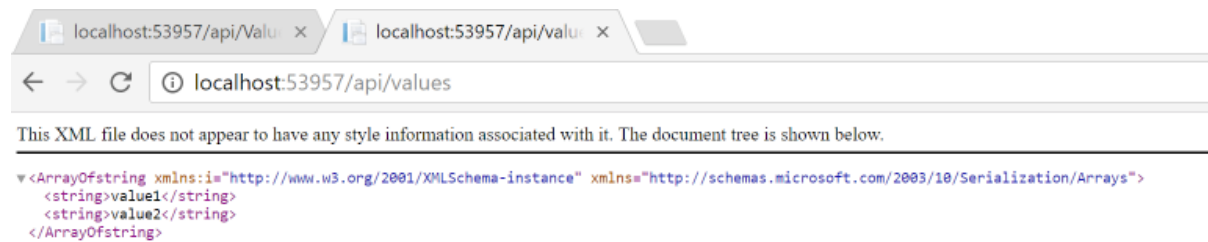
Examinons ce qui vient de se produire.

1. Vous avez à l'aide de votre navigateur envoyé une requête http avec la méthode GET vers l'url <http://localhost:53957/api/Values>.
2. Le serveur web de développement a reçu cette requête et a dû déterminer comment la traiter :
 - a. Il sait que la mention de /api/ indique que l'appel va devoir être traité par une Web API (il ne s'agit pas d'une page web à retourner) car la configuration de votre application l'a exprimé (voir classe WebApiConfig de votre projet).
 - b. Il sait également que la seconde partie reprend le nom du controller qui devra traiter la requête (voir classe WebApiConfig de votre projet).
 - c. Il va donc rechercher dans le namespace Controllers de votre solution le controller correspondant au nom indiqué dans l'URL (ce sont les conventions dont il a été question plus tôt)
 - d. Il sait qu'il s'agit d'une méthode GET, donc il recherche l'action GET sur le controller invoqué.
3. Il appelle l'action correspondante et retourne le résultat dans une réponse http.

L'étape 2 de cette séquence s'appelle le routage et fonctionne par conventions. Points d'attention : nom du controller, méthode http utilisée.

Maintenant, pourquoi avez-vous obtenu une erreur d'autorisation ? Regardez votre controller. Il est décoré avec l'attribut Authorize. Consultez l'article suivant afin de comprendre son fonctionnement : <https://www.asp.net/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api> (voir section Using the Authorize Attribute).

Commentez cet attribut pour l'instant, puis relancez le debug et réexécutez la requête http. Vous devriez avoir un résultat similaire au suivant.



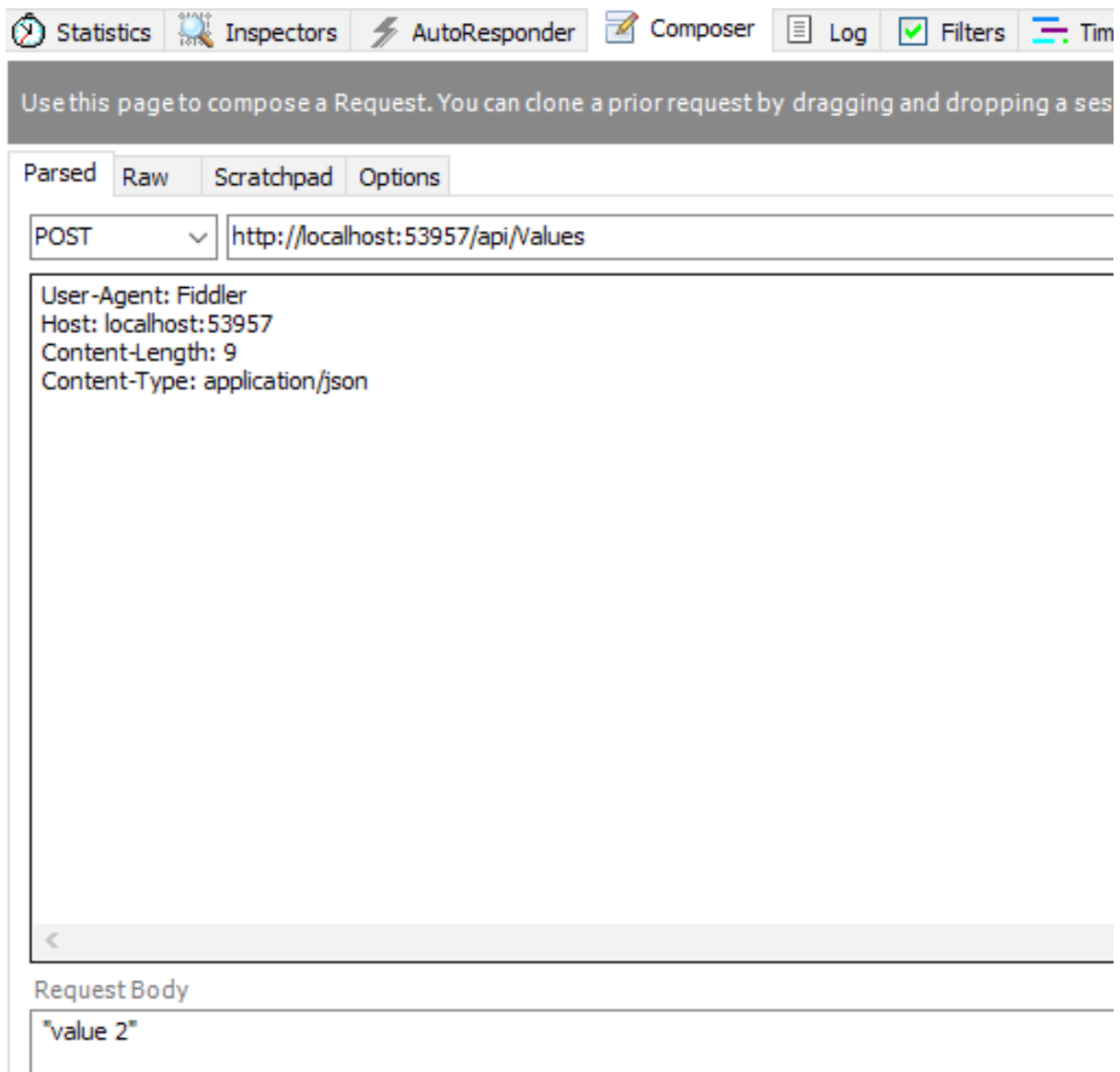
Il s'agit des valeurs Hardcodées dans le controller Values de votre projet.



Vous venez de faire un GET : une opération de lecture de données en REST. A quoi d'après-vous sert la seconde action GET reprise dans le controller ? Essayez de l'appeler.

Elle retourne "value", permet de définir une valeur ?
Essayons maintenant d'utiliser l'API REST pour insérer une nouvelle valeur dans la collection. REST décrit que pour réaliser des créations de données, il faut utiliser la méthode http POST. Examinez le controller Values. Il contient déjà cette méthode POST, mais elle ne contient pas de comportement. Modifiez votre controller afin que son implémentation soit la suivante.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Net;
5  using System.Net.Http;
6  using System.Web.Http;
7
8  namespace BankApi.Controllers
9  {
10     // [Authorize]
11     public class ValuesController : ApiController
12     {
13
14         List<string> values = new List<string> { "value1", "value2" };
15         // GET api/values
16         public IEnumerable<string> Get()
17         {
18             return values;
19         }
20
21         // GET api/values/5
22         public string Get(int id)
23         {
24             return values[id];
25         }
26
27         // POST api/values
28         public void Post([FromBody] string value)
29         {
30             values.Add(value);
31         }
32
33         // PUT api/values/5
34         public void Put(int id, [FromBody] string value)
35         {
36         }
37
38         // DELETE api/values/5
39         public void Delete(int id)
40         {
41         }
42     }
43 }
```

Nous allons ensuite réaliser un appel en POST sur le contrôleur afin d'ajouter un élément à la liste « values ». Réaliser un POST à l'aide de votre navigateur va vous sembler ardu. Nous allons utiliser un autre client http : Fiddler (<http://www.telerik.com/fiddler>). Fiddler est un debugger http gratuit. Il en existe d'autres (ex : PostMan qui a l'avantage de tourner sous plusieurs plateformes). Il est installé sur les stations de l'IESN. Lancez Fiddler, puis composez une nouvelle requête http à l'aide du composeur, telle que décrite ci-dessous.



La réponse devrait être un statut 204. Refaites l'expérience en ayant placé au préalable un breakpoint dans votre action Post sur  ller Values. Examinez la valeur reçue et la manière dont évolue la liste « values ».

Réalisez ensuite un appel en GET au controller, afin de récupérer la nouvelle liste de valeurs. Celle-ci ne devrait pas avoir évolué. Ca devrait être comme si votre insertion n'avait pas fonctionné. **Pour quelle raison ?** (octroyez-vous 5 minutes de réflexion avant de passer à la suite).

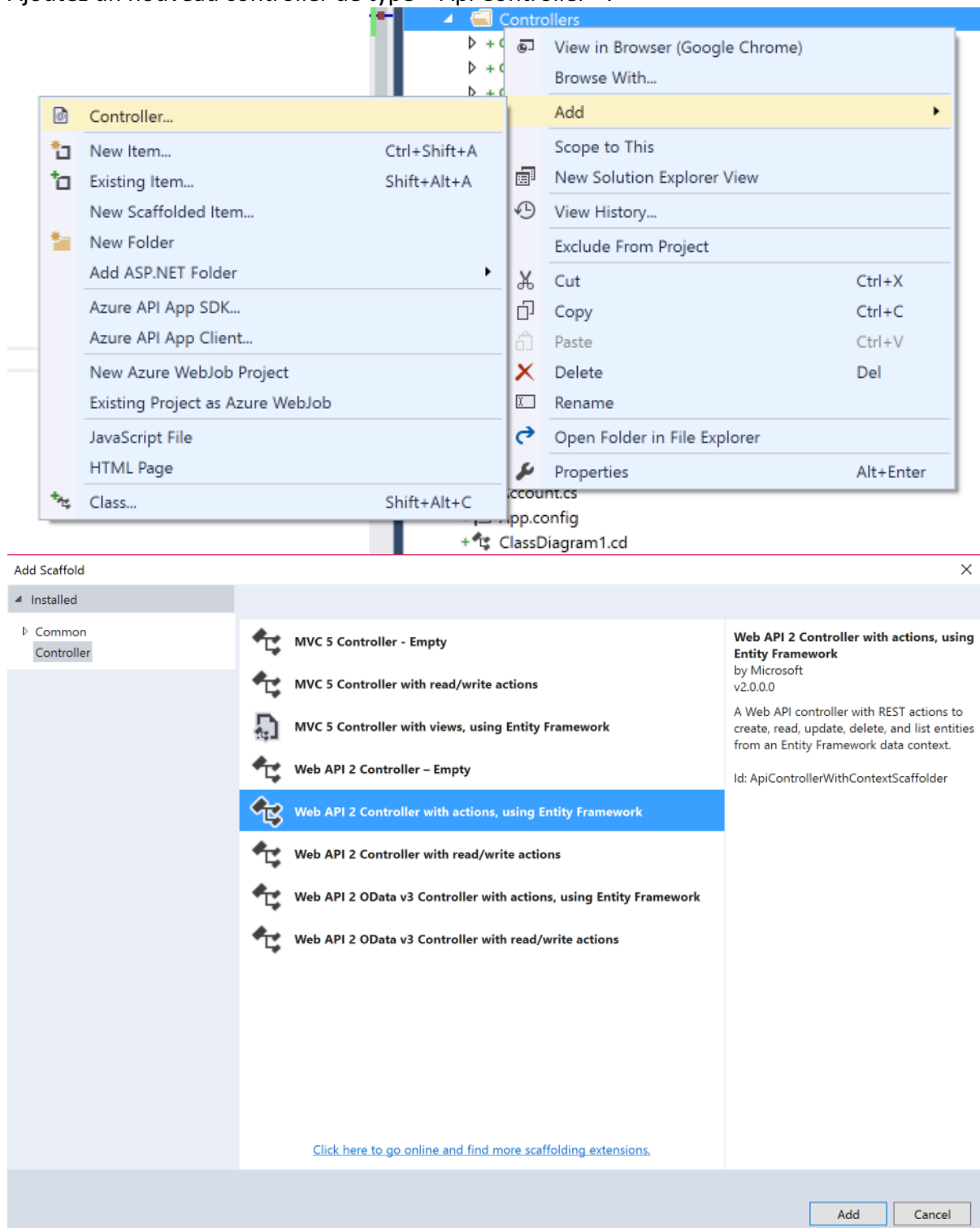
Vous avez réalisé un appel en écriture (ajout de données, POST) et un en lecture (récupération de données, GET), Si vous souhaitez mettre à jour les données, utilisez PUT ou PATCH. Si vous souhaitez en supprimer, utilisez DELETE.

Intégration à une base de données

Retourner des valeurs hardcodées n'est pas très représentatif de la réalité. Nous allons retourner des données de notre base de données à l'aide de l'API en tirant parti de la couche d'accès aux données déjà existante.

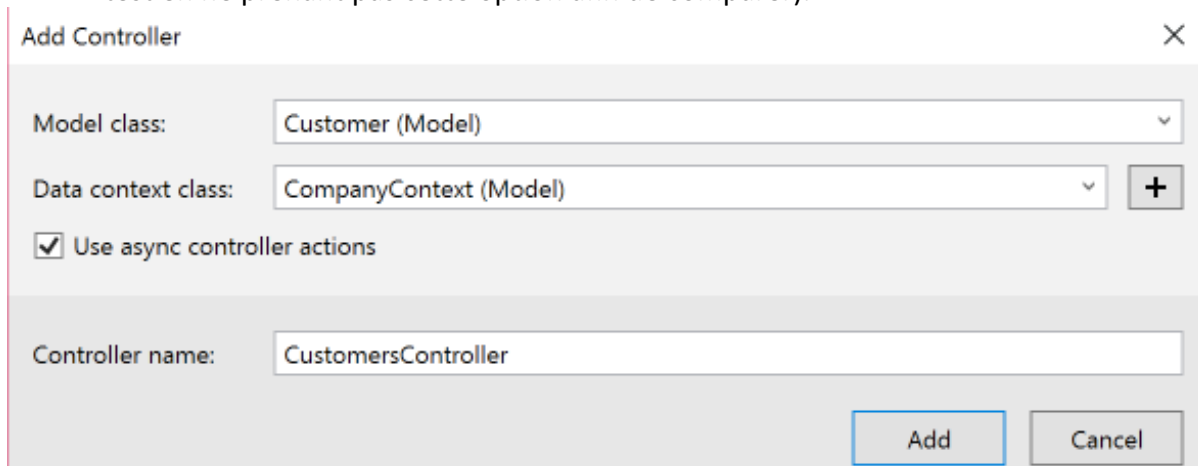
Ajoutez une référence au projet Model depuis votre Web API. Ensuite, recompilez votre solution.

Ajoutez un nouveau controller de type « Api Controller ».



Dans la boîte de dialogue suivante, vous allez devoir sélectionner plusieurs éléments :

- Model class : la classe de votre modèle qui sera retournée par le controller. Nous allons créer un controller qui va retourner des ressources de type « Customer ».
- Data context class : la classe qui va gérer les accès aux données (votre DbContext).
- Utilisez des actions de controller asynchrones (vous pourrez par la suite refaire un test en ne prenant pas cette option afin de comparer).



Relancez l'application Web, puis à l'aide de votre client http préféré, réalisez un appel à l'action Get de votre nouveau controller Customers (pas illustré ici, à vous de créer la requête. Pensez au routage). Le résultat devrait être similaire à celui-ci.

```
<ArrayOfCustomer xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/Model">
  <Customer>
    <Accounts>
      <AddressLine1>Rue des cerisiers, 16</AddressLine1>
      <AddressLine2 i:nil="true"/>
      <City>Arlon</City>
      <Country>Belgique</Country>
      <EMail>info@me.com</EMail>
      <Id>1</Id>
      <Name>Albert Dupont</Name>
      <PostCode>6700</PostCode>
      <Remark>Hey</Remark>
      <RowVersion>AAAAAAAAAB9c</RowVersion>
    </Customer>
  </Customer>
  <Customer>
    <Accounts>
      <AddressLine1>Rue Sonetty, 98</AddressLine1>
      <AddressLine2 i:nil="true"/>
      <City>Arlon</City>
      <Country>Belgique</Country>
      <EMail>info@me.com</EMail>
      <Id>2</Id>
      <Name>John Doe</Name>
      <PostCode>6700</PostCode>
      <Remark>A de grosses difficultés financières</Remark>
      <RowVersion>AAAAAAAAAB9Q</RowVersion>
    </Customer>
  </Customer>
</ArrayOfCustomer>
```

Essayez ensuite de récupérer les informations d'un client particulier (il ne doit y avoir qu'un seul client dans la réponse renvoyée).

Remarquez : les comptes en banque des clients ne sont pas inclus à la réponse. Pour quelle raison ? Aidez-vous de l'article suivant pour répondre à cette question :

<https://msdn.microsoft.com/en-us/data/jj574232.aspx> et modifiez votre

CustomersController (méthode Get) afin d'inclure les comptes en banque à la réponse http retournée au client. Cette réponse devrait alors ressembler à celle-ci.


```
<ArrayOfCustomer xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/Model">
  <Customer>
    <Accounts>
      <Account>
        <Balance>10</Balance>
        <IBAN>ABCD</IBAN>
        <Id>1</Id>
        <RowVersion>AAAAAAAAAB9I</RowVersion>
      </Account>
      <Account>
        <Balance>15</Balance>
        <IBAN>EFGH</IBAN>
        <Id>2</Id>
        <RowVersion>AAAAAAAAAB9M</RowVersion>
      </Account>
    </Accounts>
    <AddressLine1>Rue des cerisiers, 16</AddressLine1>
    <AddressLine2 i:nil="true"/>
    <City>Arlon</City>
    <Country>Belgique</Country>
    <EMail>info@me.com</EMail>
    <Id>1</Id>
    <Name>Albert Dupont</Name>
    <PostCode>6700</PostCode>
    <Remark>Hey</Remark>
    <RowVersion>AAAAAAAAAB9c</RowVersion>
  </Customer>
  <Customer>
    <Accounts>
      <Account>
        <Balance>18</Balance>
        <IBAN>IJKL</IBAN>
        <Id>3</Id>
        <RowVersion>AAAAAAAAAB9U</RowVersion>
      </Account>
      <Account>
        <Balance>9</Balance>
        <IBAN>MNOP</IBAN>
        <Id>4</Id>
        <RowVersion>AAAAAAAAAB9Y</RowVersion>
      </Account>
    </Accounts>
    <AddressLine1>Rue Sonetty, 98</AddressLine1>
    <AddressLine2 i:nil="true"/>
    <City>Arlon</City>
    <Country>Belgique</Country>
    <EMail>info@me.com</EMail>
    <Id>2</Id>
    <Name>John Doe</Name>
    <PostCode>6700</PostCode>
    <Remark>A de grosses difficultés financières</Remark>
    <RowVersion>AAAAAAAAAB9Q</RowVersion>
  </Customer>
</ArrayOfCustomer>
```

Essayez alors de créer un nouveau client. Rappel, vous devez réaliser une requête http en POST sur le contrôleur, en lui communiquant les informations relatives au nouveau client. Réalisez cette étape à l'aide de Fiddler.