



eBook Gratuit

APPRENEZ

Entity Framework Core

eBook gratuit non affilié créé à partir des
contributeurs de Stack Overflow.

#entity-

framework-

core

Table des matières

À propos.....	1
Chapitre 1: Démarrer avec Entity Framework Core.....	2
Remarques.....	2
Exemples.....	2
Ajout de packages au projet.....	2
Base de données d'abord dans Entity Framework Core avec une bibliothèque de classes et SQL.....	2
Étape 1 - Installation de .NET Core.....	2
Étape 2 - Créer les projets.....	3
Étape 3 - Installation des packages EF.....	5
----- OU.....	6
Étape 4 - Création du modèle de base de données.....	7
 finalement.....	9
 Passer une chaîne de connexion.....	10
Modèle, interrogation et sauvegarde de données.....	11
Modèle.....	11
Interroger.....	11
La sauvegarde des données.....	12
Suppression de données.....	12
Mise à jour des données.....	12
Chapitre 2: EF Core vs EF6.x.....	14
Remarques.....	14
Exemples.....	14
Comparaison côte à côte.....	14
Chapitre 3: Mettre à jour une relation plusieurs à plusieurs.....	18
Introduction.....	18
Exemples.....	18
MVC POST Editer exemple.....	18
Crédits.....	20

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [entity-framework-core](#)

It is an unofficial and free Entity Framework Core ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Entity Framework Core.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec Entity Framework Core

Remarques

Entity Framework (EF) Core est une version légère et extensible de la technologie d'accès aux données populaire d'Entity Framework.

EF Core est un mappeur objet-relationnel (O / RM) qui permet aux développeurs .NET de travailler avec une base de données à l'aide d'objets .NET. Cela élimine le besoin de la plupart du code d'accès aux données que les développeurs doivent généralement écrire.

Exemples

Ajout de packages au projet

Pour ajouter EntityFrameworkCore à votre projet, mettez à jour le fichier `project.json` (ajoutez de nouvelles lignes dans les sections des `dependencies` et des `tools`):

```
"dependencies": {  
  ...  
  "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",  
  "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",  
  "Microsoft.EntityFrameworkCore.Design": {  
    "version": "1.0.0",  
    "type": "build"  
  },  
},  
"tools": {  
  ...  
  "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"  
}
```

N'oubliez pas de lancer `dotnet restore` pour télécharger ces paquets sur Internet.

Si vous utilisez un SGBDR autre que Microsoft SQLServer - remplacez

`Microsoft.EntityFrameworkCore.SqlServer` par la version correcte (

`Microsoft.EntityFrameworkCore.Sqlite`, `Npgsql.EntityFrameworkCore.PostgreSQL` ou autre - consultez la documentation de votre SGBDR pour le package recommandé).

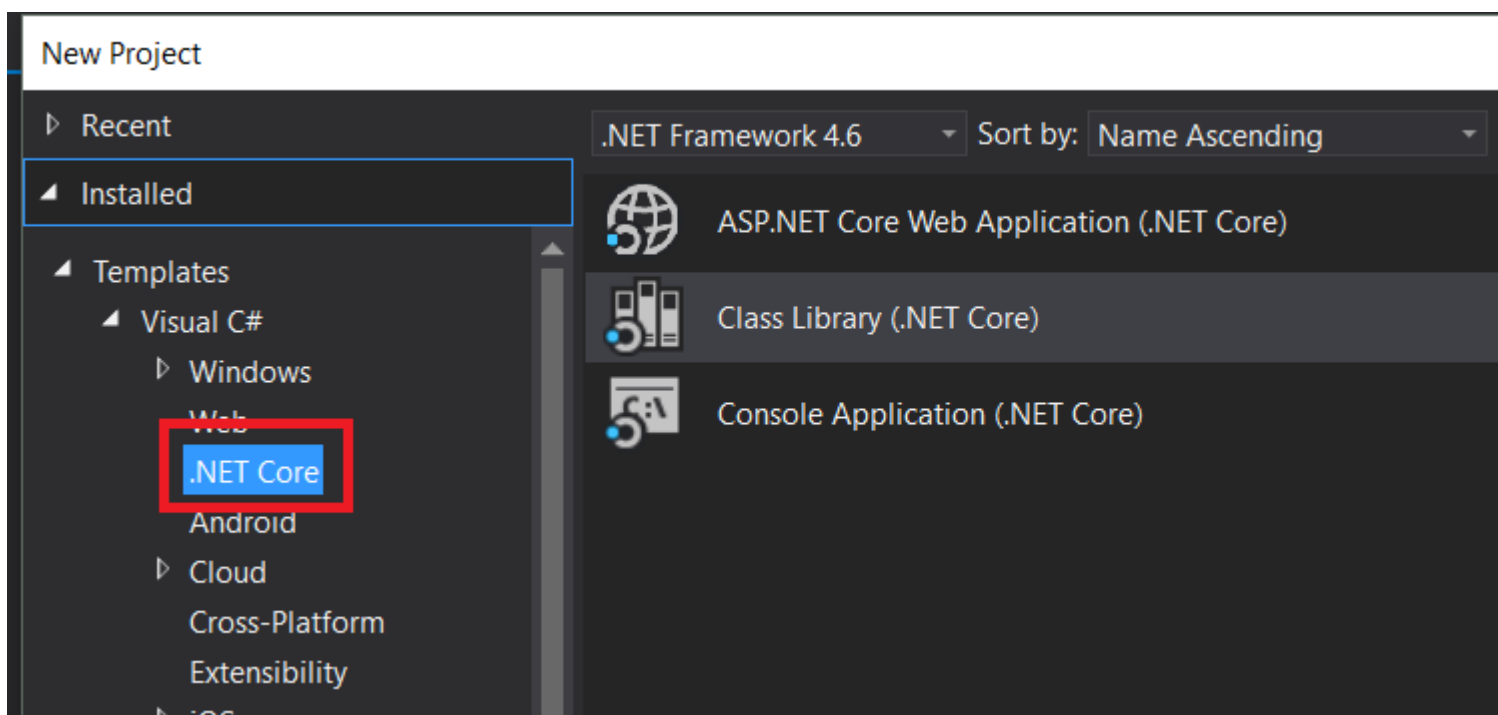
Base de données d'abord dans Entity Framework Core avec une bibliothèque de classes et SQL Server

Bon, il m'a fallu environ une journée pour le comprendre, alors voici les étapes que j'ai suivies pour que ma base de données fonctionne d'abord dans un `Class Project (.NET Core)`, avec une application Web .NET Core.

Étape 1 - Installation de .NET Core

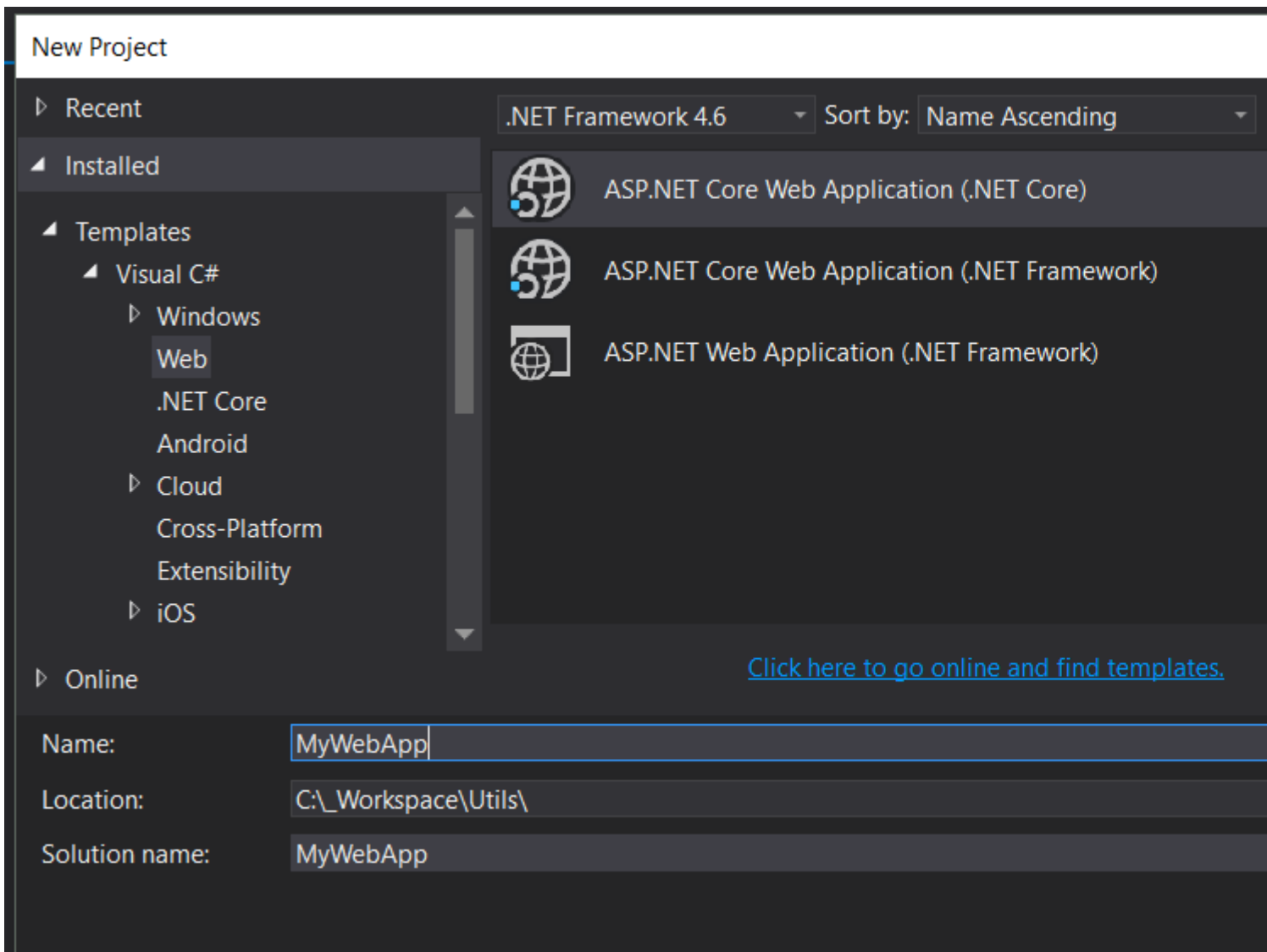
Assurez - vous que vous utilisez Core .NET ne DNX (Hint: You should be able to see the .NET Core option when creating a New Project) **en** (Hint: You should be able to see the .NET Core option when creating a New Project) **de** (Hint: You should be able to see the .NET Core option when creating a New Project) **de la** (Hint: You should be able to see the .NET Core option when creating a New Project) **d'** (Hint: You should be able to see the .NET Core option when creating a New Project) - Si pas télécharger à partir [ici](#)

Si vous rencontrez des problèmes lors de l'installation de .NET Core (l'erreur est quelque chose comme Visual Studio 2015 Update 3 n'est pas installé correctement) - Vous pouvez exécuter l'installation à l'aide de la commande: [`DotNetCore.1.0.0-VS2015Tools.Preview2.exe SKIP_VSU_CHECK=1`] Ce qui empêchera l'installation d'effectuer le [problème](#) Visual Studio Check [Github](#)

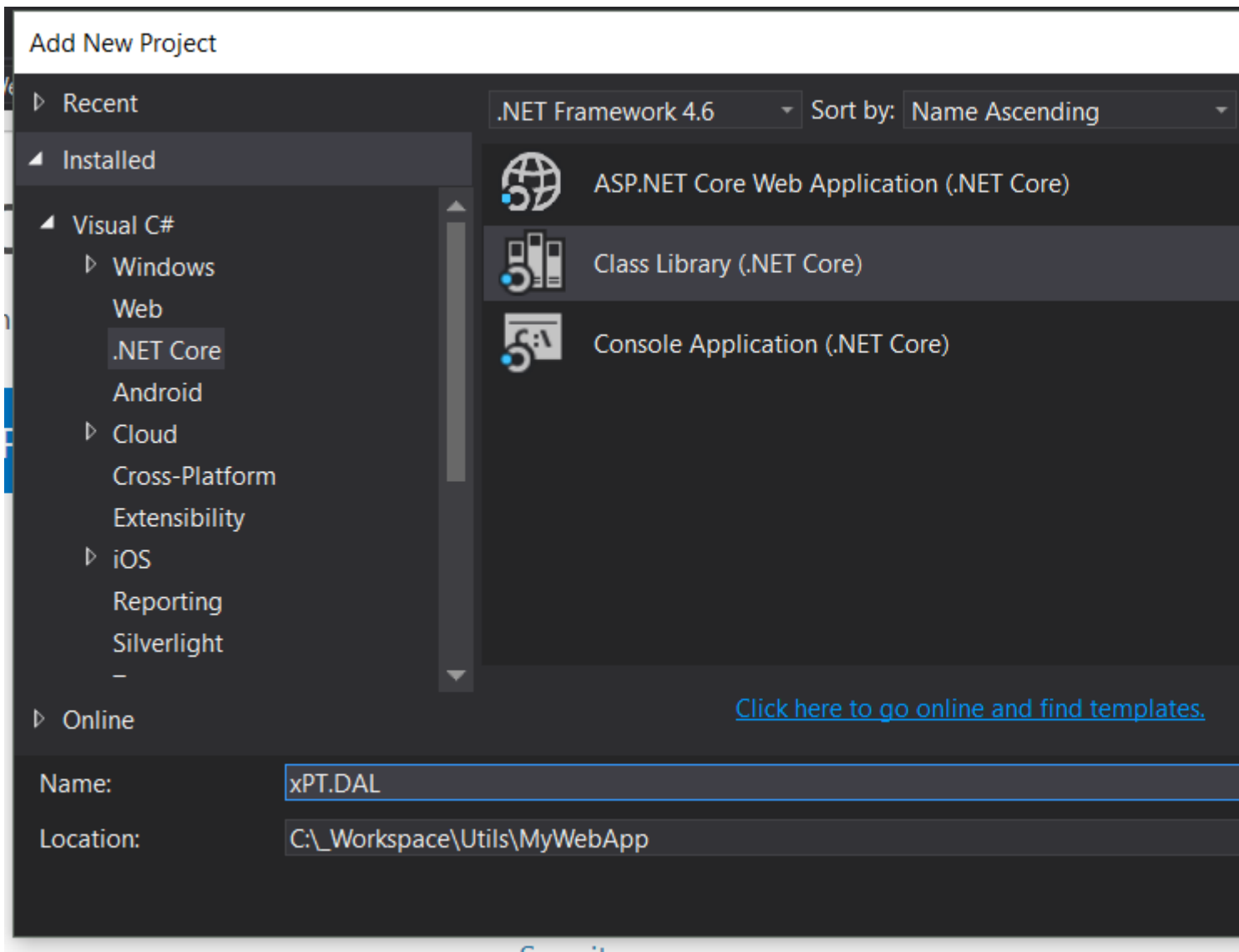


Étape 2 - Créer les projets

Créez une nouvelle application Web ASP.NET Core -> Sélectionnez ensuite l'application Web dans l'écran suivant.



Ajouter un projet de `Class Library (.NET Core)`



Étape 3 - Installation des packages EF

Ouvrez votre fichier `project.json` de la bibliothèque de classes et collez les éléments suivants, puis enregistrez le fichier:

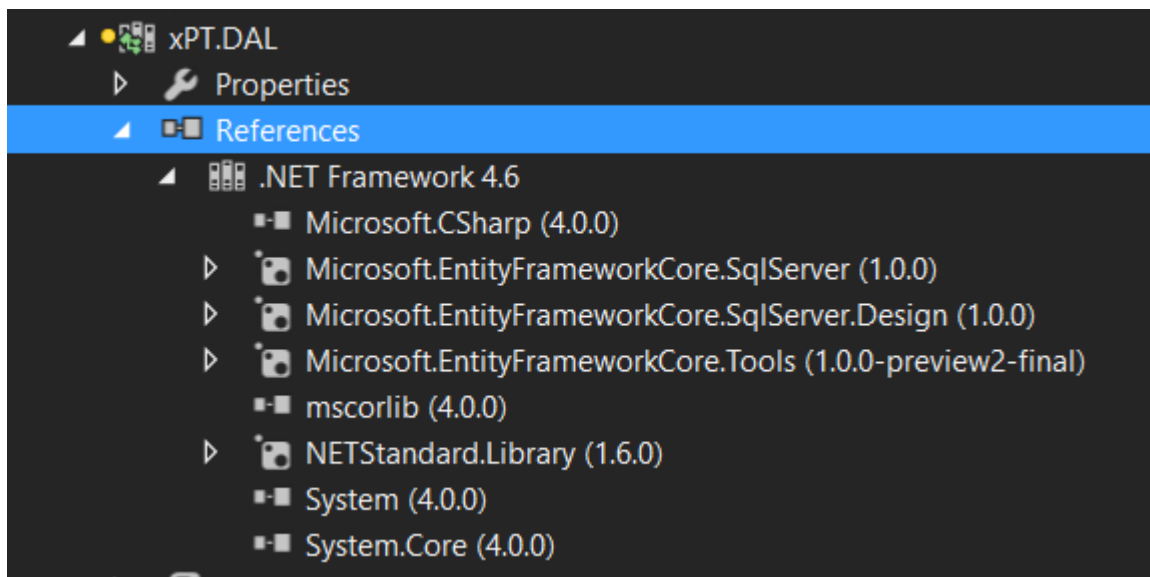
```
{
  "version": "1.0.0-*",
  "dependencies": {
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "NETStandard.Library": "1.6.0"
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  },
  "frameworks": {
    "net46": {
    }
  }
}
```

```

"netcoreapp1.0": {
  "dependencies": {
    "Microsoft.NETCore.App": {
      "type": "platform",
      "version": "1.0.0-*"
    }
  }
}
}
}
}

```

Cela devrait restaurer les paquets sous `References`



OU

Vous pouvez les installer à l'aide du gestionnaire de packages Nuget en exécutant les commandes suivantes dans la console du gestionnaire de packages.

```

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools -Pre

Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design

```

Remarque: installez un package à la fois - si vous obtenez une erreur après l'installation

```
Microsoft.EntityFrameworkCore.Tools
```

Puis changez le contenu de votre section `frameworks` `project.json` à ceci:

```

"frameworks": {
  "net46": {
  },
  "netcoreapp1.0": {
    "dependencies": {
      "Microsoft.NETCore.App": {

```



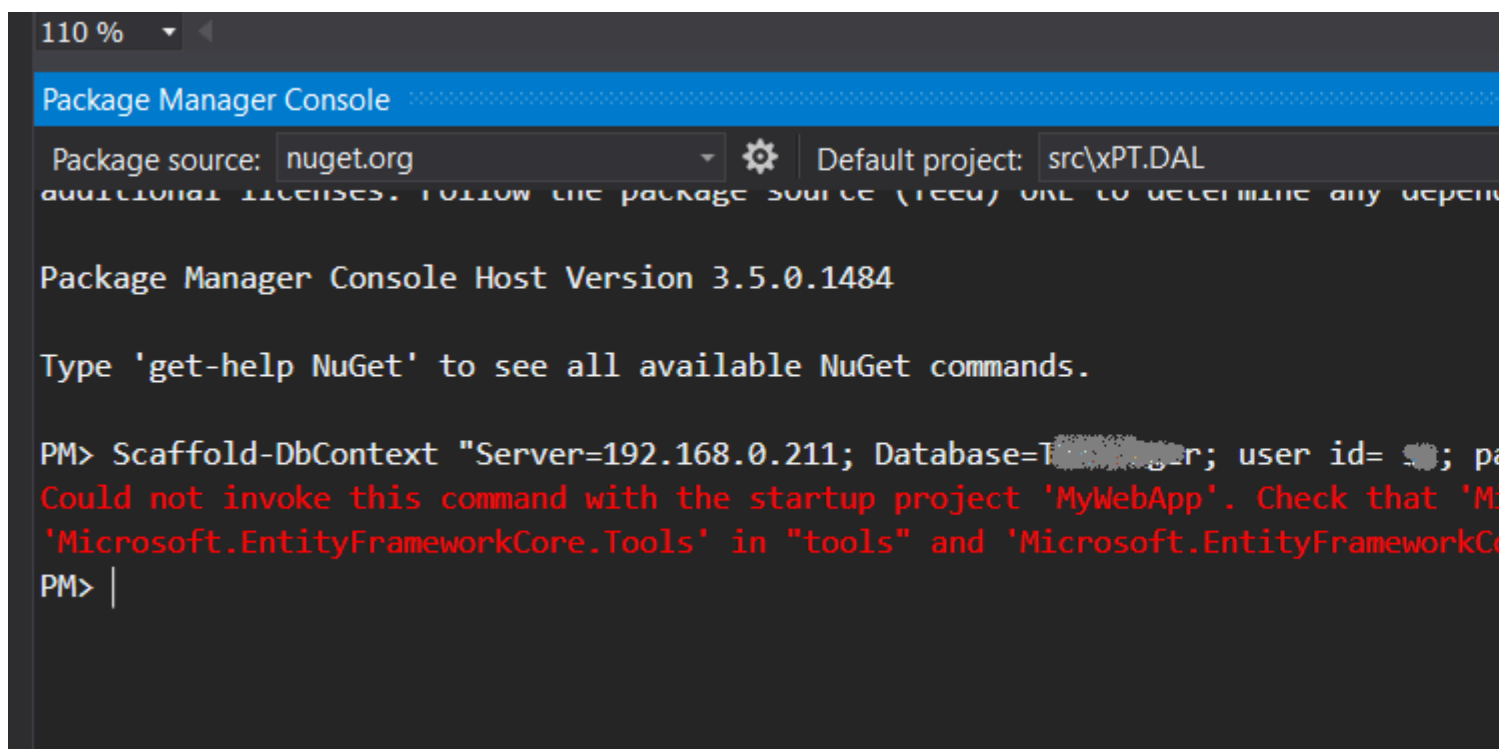
```
        "type": "platform",
        "version": "1.0.0-*"
    }
}
}
```

Étape 4 - Création du modèle de base de données

Maintenant, pour générer la base de données, exécutez la commande suivante dans la `Package Manager Console` (n'oubliez pas de modifier la chaîne de connexion à votre base de données)

```
Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"
Microsoft.EntityFrameworkCore.SqlServer
```

Cela vous donnera l'erreur sur le projet de démarrage:



The screenshot shows the Package Manager Console interface. At the top, it displays '110 %' and 'Package Manager Console'. Below that, it shows 'Package source: nuget.org' and 'Default project: src\xPT.DAL'. The main text in the console reads: 'Package Manager Console Host Version 3.5.0.1484', 'Type \'get-help NuGet\' to see all available NuGet commands.', and 'PM> Scaffold-DbContext "Server=192.168.0.211; Database=T...; user id= ...; pa...'. Below this, a red error message is displayed: 'Could not invoke this command with the startup project \'MyWebApp\''. Check that \'M...\' \'Microsoft.EntityFrameworkCore.Tools\' in "tools" and \'Microsoft.EntityFrameworkCore...'. The prompt 'PM>' is visible at the bottom left.

Pour cela, vous devez ajouter les mêmes références que vous avez ajoutées à Class Library à l'application Web .NET.

Alors ouvrez votre `project.json` pour l'application Web,

Sous `dependencies`, ajoutez:

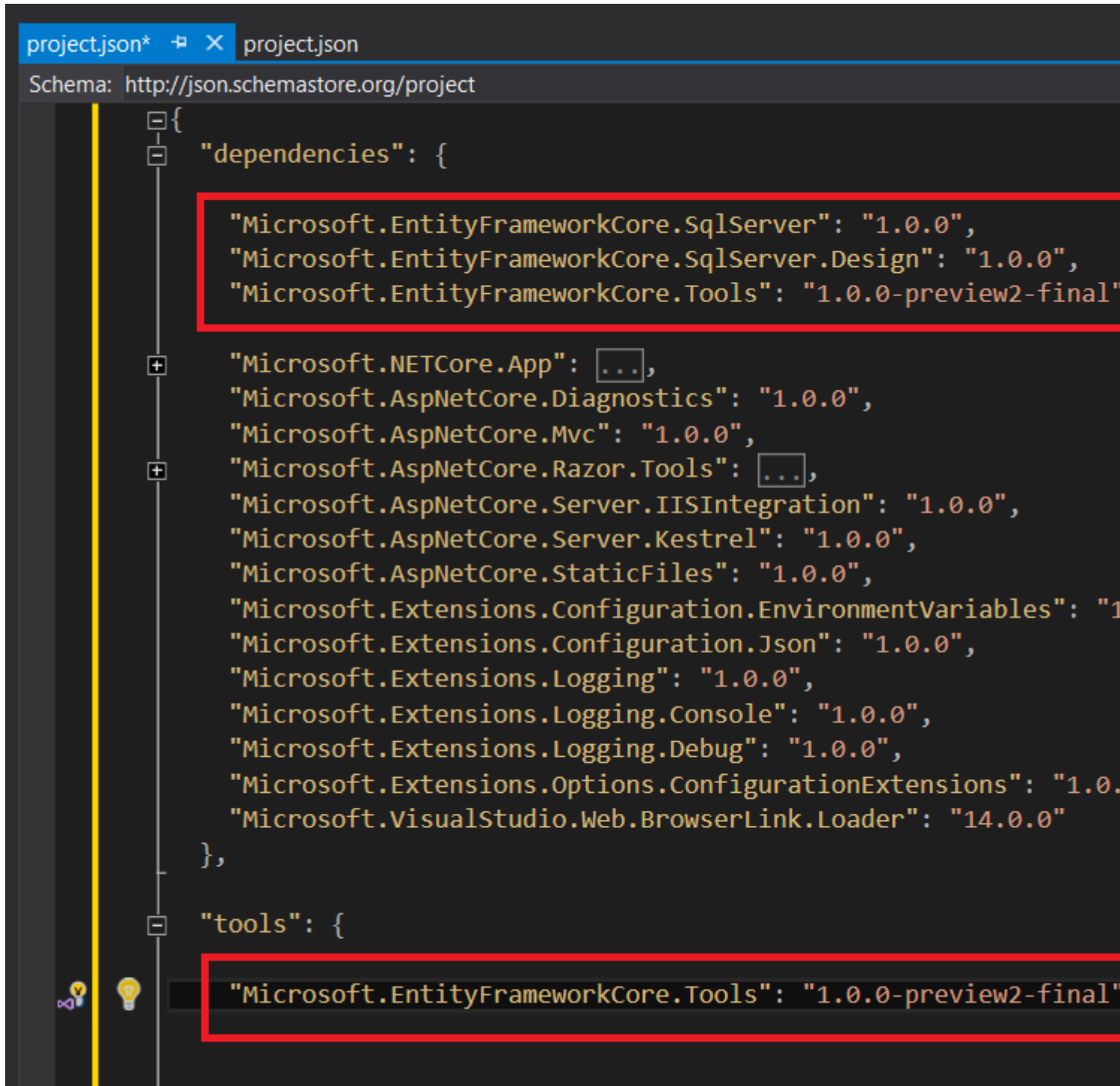
```
"Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
"Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

et sous `tools` ajouter:

```
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

Après avoir apporté les modifications Enregistrez le fichier.

Voici à quoi ressemble mon projet.json



```
project.json* X project.json
Schema: http://json.schemastore.org/project
{
  "dependencies": {
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "Microsoft.NETCore.App": "...",
    "Microsoft.AspNetCore.Diagnostics": "1.0.0",
    "Microsoft.AspNetCore.Mvc": "1.0.0",
    "Microsoft.AspNetCore.Razor.Tools": "...",
    "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
    "Microsoft.AspNetCore.StaticFiles": "1.0.0",
    "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.0.0",
    "Microsoft.Extensions.Configuration.Json": "1.0.0",
    "Microsoft.Extensions.Logging": "1.0.0",
    "Microsoft.Extensions.Logging.Console": "1.0.0",
    "Microsoft.Extensions.Logging.Debug": "1.0.0",
    "Microsoft.Extensions.Options.ConfigurationExtensions": "1.0.0",
    "Microsoft.VisualStudio.Web.BrowserLink.Loader": "14.0.0"
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  }
}
```

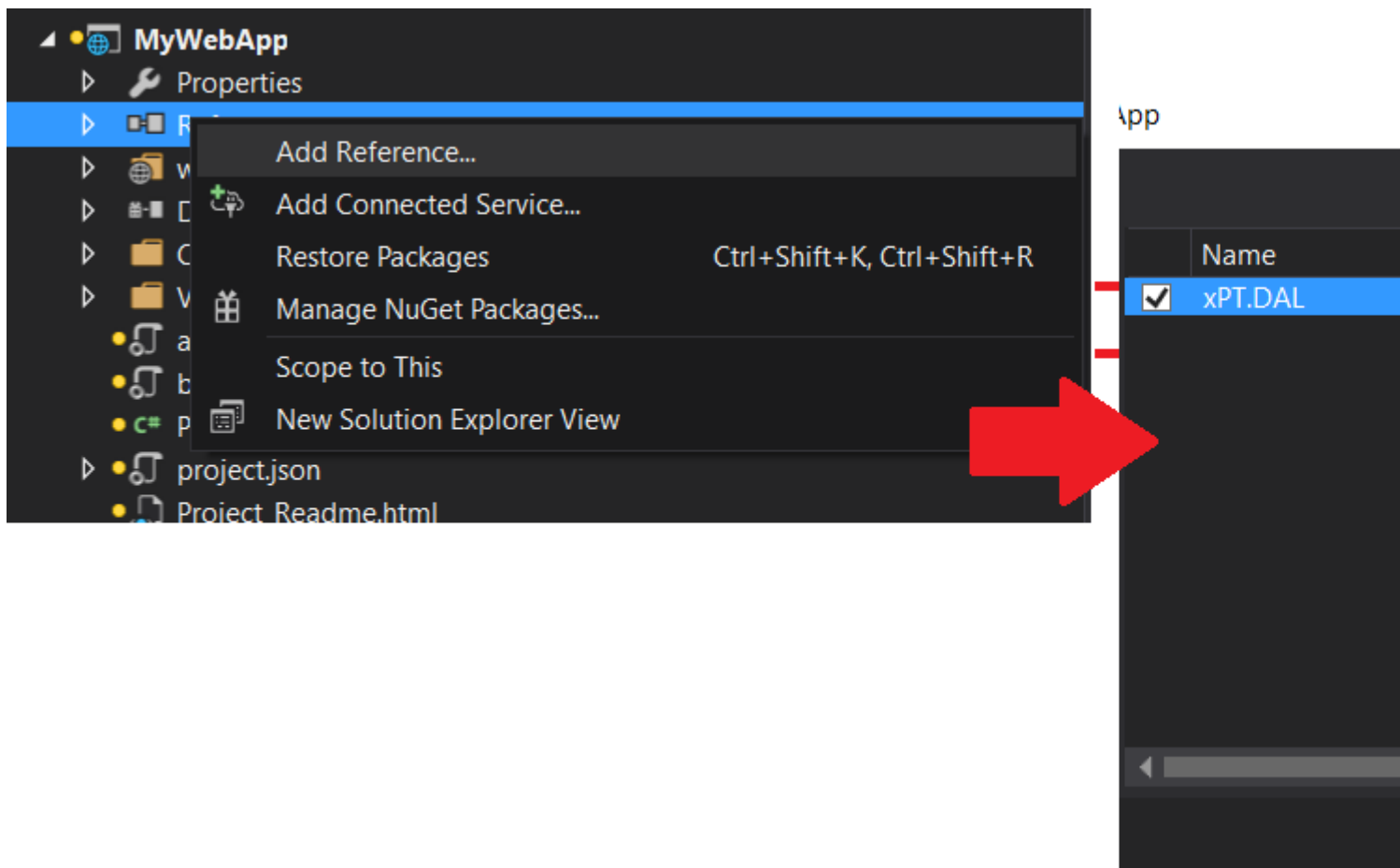
Ensuite, exécutez à nouveau la commande dans Package Manager Console sur la bibliothèque de classes:

Si vous n'avez pas encore ajouté la référence de votre bibliothèque de classes à l'application

Web, vous obtiendrez cette erreur:

```
PM> Scaffold-DbContext "Server=192.168.0.211; Database=
System.AggregateException: One or more errors occurred. (Could not find assembly
Microsoft.EntityFrameworkCore.Design.OperationException: Could not find assembly
Microsoft.EntityFrameworkCore.Design.Internal.OperationExecutor..ctor(CommonOptio
    at Microsoft.EntityFrameworkCore.Tools.Cli.DbContextScaffoldCommand.<Execute/
--- End of inner exception stack trace ---
    at System.Threading.Tasks.Task.ThrowIfExceptional(Boolean includeTaskCanceled
    at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotifica
```

pour résoudre cette référence, ajoutez votre bibliothèque de classes à votre application Web:

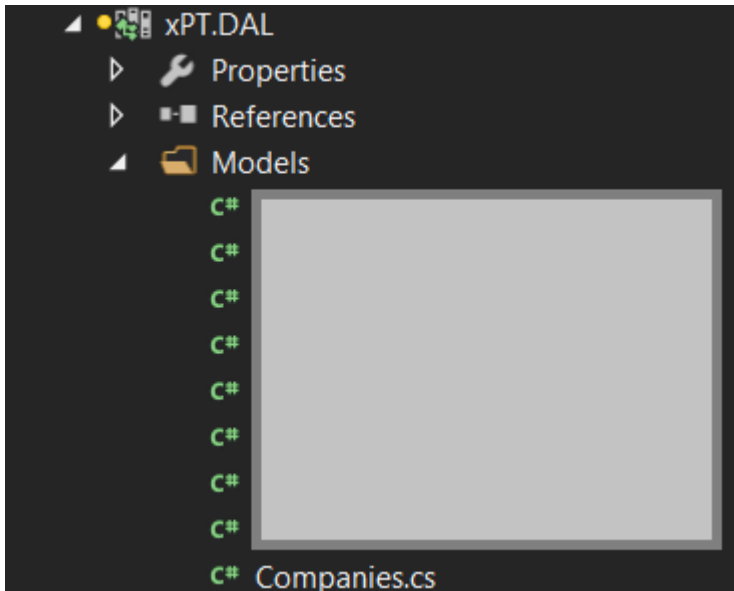


finalement

Exécutez à nouveau la commande - dans la Package Manager Console :

```
Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Cela devrait créer le dossier Entités sous Modèles, dans la bibliothèque de classes



Passer une chaîne de connexion

Dans mon cas, nous avons une application multi-locataire dans laquelle chaque client a sa propre base de données, par exemple Client_1, Client_2, Client_3. La chaîne de connexion devait donc être dynamique.

Nous avons donc ajouté une propriété de chaîne de connexion à un constructeur et l'

`OnConfiguring` transmise au contexte dans la méthode `OnConfiguring`

```
public partial class ClientContext
{
    private readonly string _connectionString;

    public ClientContext(string connectionString) : base()
    {
        _connectionString = connectionString;
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connectionString);
    }
}
```

et l'a utilisé comme ceci:

```
public void TestConnection()
{
    var clientId = 1;

    var connectionString = string.Format("Server=192.168.0.211; Database=Client_{0}; user id= USER; password = PWD;", clientId);

    using (var clientContext = new ClientContext(connectionString))
    {
        var assets = clientContext.Users.Where(s => s.UserId == 1);
    }
}
```

```
}  
}
```

Modèle, interrogation et sauvegarde de données

Modèle

Avec EF Core, l'accès aux données est effectué à l'aide d'un modèle. Un modèle est composé de classes d'entités et d'un contexte dérivé qui représente une session avec la base de données, ce qui vous permet d'interroger et de sauvegarder des données.

Vous pouvez générer un modèle à partir d'une base de données existante, coder manuellement un modèle pour qu'il corresponde à votre base de données ou utiliser EF Migrations pour créer une base de données à partir de votre modèle (et le faire évoluer au fil du temps).

```
using Microsoft.EntityFrameworkCore;  
using System.Collections.Generic;  
  
namespace Intro  
{  
    public class BloggingContext : DbContext  
    {  
        public DbSet<Blog> Blogs { get; set; }  
        public DbSet<Post> Posts { get; set; }  
  
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  
        {  
optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True");  
        }  
    }  
  
    public class Blog  
    {  
        public int BlogId { get; set; }  
        public string Url { get; set; }  
  
        public List<Post> Posts { get; set; }  
    }  
  
    public class Post  
    {  
        public int PostId { get; set; }  
        public string Title { get; set; }  
        public string Content { get; set; }  
  
        public int BlogId { get; set; }  
        public Blog Blog { get; set; }  
    }  
}
```

Interroger

Les instances de vos classes d'entités sont extraites de la base de données à l'aide de Language Integrated Query (LINQ).

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

La sauvegarde des données

Les données sont créées, supprimées et modifiées dans la base de données à l'aide d'instances de vos classes d'entités.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

Suppression de données

Les instances de vos classes d'entités sont extraites de la base de données à l'aide de Language Integrated Query (LINQ).

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Attach(blog);
    db.Blogs.Remove(blog);
    db.SaveChanges();
}
```

Mise à jour des données

Les données sont mises à jour dans la base de données à l'aide d'instances de vos classes d'entités.

```
using (var db = new BloggingContext())
```

```
{
    var blog = new Blog { Url = "http://sample.com" };
    var entity = db.Blogs.Find(blog);
    entity.Url = "http://sample2.com";
    db.SaveChanges();
}
```

Lire Démarrer avec Entity Framework Core en ligne: <https://riptutorial.com/fr/entity-framework-core/topic/3796/demarrer-avec-entity-framework-core>

Chapitre 2: EF Core vs EF6.x

Remarques

Pour les dernières mises à jour, veuillez vous reporter à: [Comparaison des fonctionnalités](#)

Exemples

Comparaison côte à côte

Le tableau suivant compare les fonctionnalités disponibles (1) dans EF Core et EF6.x.

Il est destiné à fournir une comparaison de haut niveau et ne répertorie pas toutes les fonctionnalités, ni ne tente de donner des détails sur les différences possibles entre les mêmes fonctionnalités.

Créer un modèle	EF6.x	EF Core 1.0.0
Modélisation de base (classes, propriétés, etc.)	Oui	Oui
Conventions	Oui	Oui
Conventions personnalisées	Oui	Partiel
Annotations de données	Oui	Oui
API Fluent	Oui	Oui
Héritage: Table per hierarchy (TPH)	Oui	Oui
Héritage: Table par type (TPT)	Oui	
Héritage: Table par classe concrète (TPC)	Oui	
Propriétés de l'état d'ombre		Oui
Clés alternatives		Oui
Plusieurs-à-plusieurs: Avec entité de jointure	Oui	Oui
Plusieurs à plusieurs: sans entité	Oui	
Génération de clés: base de données	Oui	Oui
Génération de clés: Client		Oui
Types complexes / valeur	Oui	

Créer un modèle		EF6.x	EF Core 1.0.0
Données spatiales		Oui	
Visualisation graphique du modèle		Oui	
Editeur graphique de glisser / déposer		Oui	
Format du modèle: code		Oui	Oui
Format du modèle: EDMX (XML)		Oui	
Reverse engineering d'un modèle à partir d'une base de données: ligne de commande			Oui
Inversion du modèle à partir de la base de données: assistant VS		Oui	
Modèle de mise à jour incrémentielle à partir de la base de données		Oui	
Demande de données	EF6.x	EF Core 1.0.0	
LINQ: requêtes simples	Stable	Stable	
LINQ: requêtes modérées	Stable	Stabilisation	
LINQ: requêtes complexes	Stable	En cours	
LINQ: requêtes utilisant des propriétés de navigation	Stable	En cours	
Génération «jolie» SQL	Pauvre	Oui	
Evaluation mixte client / serveur		Oui	
Chargement des données liées: Désireux	Oui	Oui	
Chargement des données associées: Lazy	Oui		
Chargement des données associées: Explicite	Oui		
Requêtes SQL brutes: types de modèles	Oui	Oui	
Requêtes SQL brutes: types non mappés	Oui		
Requêtes SQL brutes: composition avec LINQ		Oui	
La sauvegarde des données	EF6.x	EF Core 1.0.0	
Sauvegarder les modifications		Oui	Oui
Suivi des modifications: instantané		Oui	Oui

La sauvegarde des données	EF6.x	EF Core 1.0.0
Suivi des modifications: notification	Oui	Oui
Accès à l'état suivi	Oui	Partiel
Concurrence optimiste	Oui	Oui
Transactions	Oui	Oui
Mise en lots des relevés		Oui
Procédure stockée	Oui	
Prise en charge des graphiques détachés (N-Tier): API de bas niveau	Pauvre	Oui
Prise en charge des graphiques détachés (niveau N): de bout en bout		Pauvre
Autres caractéristiques	EF6.x	EF Core 1.0.0
Migrations	Oui	Oui
API de création / suppression de base de données	Oui	Oui
Données de semences	Oui	
Résilience de connexion	Oui	
Crochets de cycle de vie (événements, interception de commandes, ...)	Oui	
Fournisseurs de bases de données	EF6.x	EF Core 1.0.0
serveur SQL	Oui	Oui
MySQL	Oui	Payé seulement, non payé à venir (2)
PostgreSQL	Oui	Oui
Oracle	Oui	Payé seulement, non payé à venir (2)
SQLite	Oui	Oui
SQL Compact	Oui	Oui
DB2	Oui	Oui

Fournisseurs de bases de données	EF6.x	EF Core 1.0.0
InMemory (pour tester)		Oui
Azure Table Storage		Prototype
Redis		Prototype

Modèles d'application	EF6.x	EF Core 1.0.0
WinForms	Oui	Oui
WPF	Oui	Oui
Console	Oui	Oui
ASP.NET	Oui	Oui
ASP.NET Core		Oui
Xamarin		Bientôt (3)
UWP		Oui

Notes de bas de page:

(1): à partir de 2016/10/18

(2): les fournisseurs payants sont disponibles, les fournisseurs non rémunérés sont en cours de traitement. Les équipes travaillant sur les fournisseurs non rémunérés n'ont pas partagé les détails publics de la chronologie, etc.

(3): EF Core est conçu pour fonctionner sur Xamarin lorsque la prise en charge de .NET Standard est activée dans Xamarin.

Lire EF Core vs EF6.x en ligne: <https://riptutorial.com/fr/entity-framework-core/topic/7513/ef-core-vs-ef6-x>

Chapitre 3: Mettre à jour une relation plusieurs à plusieurs

Introduction

Comment mettre à jour une relation Many to Many dans EF Core:

Exemples

MVC POST Editer exemple

Disons que nous avons une classe de produits avec plusieurs couleurs qui peuvent être sur de nombreux produits.

```
public class Product
{
    public int ProductId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}

public class ColorProduct
{
    public int ProductId { get; set; }
    public int ColorId { get; set; }

    public virtual Color Color { get; set; }
    public virtual Product Product { get; set; }
}

public class Color
{
    public int ColorId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}
```

Utiliser cette extension pour faciliter les choses:

```
public static class Extensions
{
    public static void TryUpdateManyToMany<T, TKey>(this DbContext db, IEnumerable<T>
currentItems, IEnumerable<T> newItems, Func<T, TKey> getKey) where T : class
    {
        db.Set<T>().RemoveRange(currentItems.Except(newItems, getKey));
        db.Set<T>().AddRange(newItems.Except(currentItems, getKey));
    }

    public static IEnumerable<T> Except<T, TKey>(this IEnumerable<T> items, IEnumerable<T>
other, Func<T, TKey> getKeyFunc)
    {
        return items
            .GroupJoin(other, getKeyFunc, getKeyFunc, (item, tempItems) => new { item,
```

```

tempItems })
    .SelectMany(t => t.tempItems.DefaultIfEmpty(), (t, temp) => new { t, temp })
    .Where(t => ReferenceEquals(null, t.temp) || t.temp.Equals(default(T)))
    .Select(t => t.t.item);
}
}

```

La mise à jour des couleurs d'un produit ressemblerait à ceci (une méthode MVC Edit POST)

```

[HttpPost]
public IActionResult Edit(ProductVm vm)
{
    if (ModelState.IsValid)
    {
        var model = db.Products
            .Include(x => x.ColorProducts)
            .FirstOrDefault(x => x.ProductId == vm.Product.ProductId);

        db.TryUpdateManyToMany(model.ColorProducts, vm.ColorsSelected
            .Select(x => new ColorProduct
            {
                ColorId = x,
                ProductId = vm.Product.ProductId
            }), x => x.ColorId);

        db.SaveChanges();

        return RedirectToAction("Index");
    }
    return View(vm);
}

public class ProductVm
{
    public Product Product { get; set; }

    public IEnumerable<int> ColorsSelected { get; set; }
}

```

Le code a été simplifié autant que possible, pas de propriétés supplémentaires sur les classes.

Lire [Mettre à jour une relation plusieurs à plusieurs en ligne](https://riptutorial.com/fr/entity-framework-core/topic/9527/mettre-a-jour-une-relation-plusieurs-a-plusieurs): <https://riptutorial.com/fr/entity-framework-core/topic/9527/mettre-a-jour-une-relation-plusieurs-a-plusieurs>

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Entity Framework Core	Community , Dawood Awan , Dmitry , hasan , natemcmaster , NovaDev , tmg , uTeisT
2	EF Core vs EF6.x	Frédéric , Ruud Lenders , uTeisT
3	Mettre à jour une relation plusieurs à plusieurs	Paw Ormstrup Madsen