Rencontre 11

Images

Bases de données et programmation Web

Sommaire

- Gestion des images

Gestion des images

❖ Gestion des images

- ♦ L'objectif sera de stocker des images et de pouvoir les afficher dans les pages Web au besoin.
  - ○ Il y aura peu de similarités avec la gestion des images du cours Prog Web services.
  - ○ Nous allons tenter d'isoler la gestion des fichiers dans SQL Server et d'utiliser des méthodes différentes pour apprendre de nouvelles choses.

Gestion des images

Dilemme : où stocker les images ?

- Choix 1 : Dans la **base de données**. Le type utilisé serait varbinary(max) et les images seraient stockées en tant que BLOB. (Binary Large OBject, donc des gros amas de bytes)

- Choix 2 : Dans le **système de fichiers** du serveur. Il n'y aurait donc aucun fichier image dans la BD, mais plutôt une référence qui permettrait de retrouver l'image dans le système de fichiers.

- Choix 3 : Dans un *cloud storage* qui gère les images pour nous. (Azure Blob Storage, Amazon S3, etc.) Permet de simplifier le stockage des fichiers lourds. L'application Web n'a pas besoin de rouler sur le même serveur que les images.
  - Nous écarterons cette option vu qu'elle est coûteuse et surtout pertinente à grande échelle.

Gestion des images

❖ Choix 1 : Stocker les images dans la BD

♦ Avantages possibles 📈

- Pas besoin de nouvelles stratégies de backup comme les images sont avec les autres données. (Les backups de la BD sont beaucoup plus **lourds** par contre)
- Plus facile d'assurer l'intégrité des données en lien avec les images.
- Facile de **limiter l'accès** aux images grâce à des permissions SQL.

♦ Désavantages possibles 📉

- Les images peuvent prendre jusqu'à **deux fois plus d'espace** dans une BD que dans le file system à cause de leur conversion en varbinary.
- Réduction de la performance. Généralement, récupérer une image dans la BD est plus lent que récupérer une image dans le file system... et dans tous les cas, la base de données a plus de données à gérer que si les images étaient stockées ailleurs.
- Une BD met en cache les données fréquemment accédées. Si des images prennent beaucoup de place dans le cache, ça fait moins d'espace pour d'autres données plus petites et cela rend le **cache moins performant**.

❖ Choix 2 : Stocker les images dans le File System

♦ Avantages possibles 📈

- Meilleure performance : Le File System est généralement mieux adapté pour charger des fichiers lourds et stocker une grande quantité de fichiers sans problèmes.
- Simplicité : Évite certains défis de conversion. Avec la plupart des applications côté serveur, récupérer des fichiers dans le File System est très simple et n'apporte pas de défi particulier.

♦ Désavantages possibles 📉

- Nécessite une autre **stratégie de backup**.
- Nécessite une **stratégie d'accès** pour éviter que n'importe qui fouille dans les dossiers.
- La BD a moins de contrôle sur **l'intégrité des données** qui sont à l'extérieur de la BD, bien entendu. Par exemple, facile d'oublier de supprimer une image lorsqu'on DELETE la rangée associée dans la BD.

Gestion des images

- Choix1: Stocker les images dans la BD

VS

- Choix2: Stocker les images dans le File System

- Les deux options sont couramment utilisées en entreprise. Cela dit, nous opterons pour le File System car cela semble être une solution **moins coûteuse**, **plus performante** et généralement **plus appropriée** pour des applications où le **nombre d'images** est grand.

Gestion des images

- Dans les diapos qui suivent ...
  - Configurer FILESTREAM avec SQL Server
  - Préparer une table qui stockera des images
  - Action + Vue Razor + ViewModel pour upload d'une image dans l'appli
  - Action + Vue Razor + ViewModel pour afficher des images dans l'appli

Gestion des images

- Configurer FILESTREAM avec SQL Server
  - SQL Server nous permet de stocker des fichiers en varbinary(max) ... mais il nous propose aussi une alternative pour nous accompagner dans le stockage des images dans le File System !
    - Cette alternative est le FILESTREAM.
  - Rappel : Avec l'usage d'Evolve, on exécute les **migrations** à l'aide de **commandes** plutôt qu'exécuter les scripts dans SSMS.
    - **Exception** : On devait exécuter CREATE DATABASE à la main, car cette instruction ne peut pas être utilisée dans une **transaction** avec Evolve.
    - La configuration du FILESTREAM devra aussi être faite **à la main**, sans Evolve. Ça veut dire qu'à chaque fois que vous changez de poste de travail, il faudra configurer FILESTREAM à nouveau en créant la BD.
      - Il est donc suggéré de se garder à portée de main un **script SQL** qui crée la BD et qui configure FILESTREAM.

Gestion des images

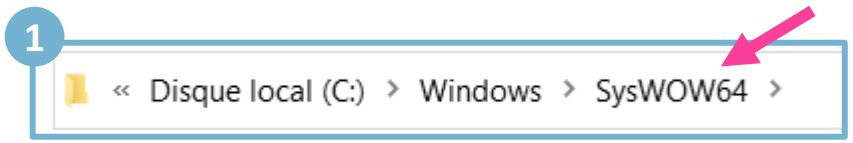
Messages

```
Msg 5591, Niveau 16, État 3, Ligne 5
FILESTREAM feature is disabled.
Msg 1921, Niveau 16, État 4, Ligne 8
Invalid filegroup 'FG_Images' specified.
```

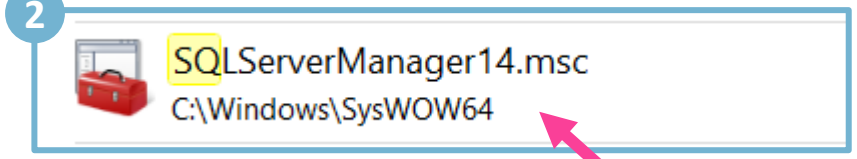
❖ Configurer FILESTREAM avec SQL Server

- ♦ Si vous avez une erreur qui mentionne que FILESTREAM n'est pas activé sur le serveur SQL, voici la marche à suivre pour l'activer :

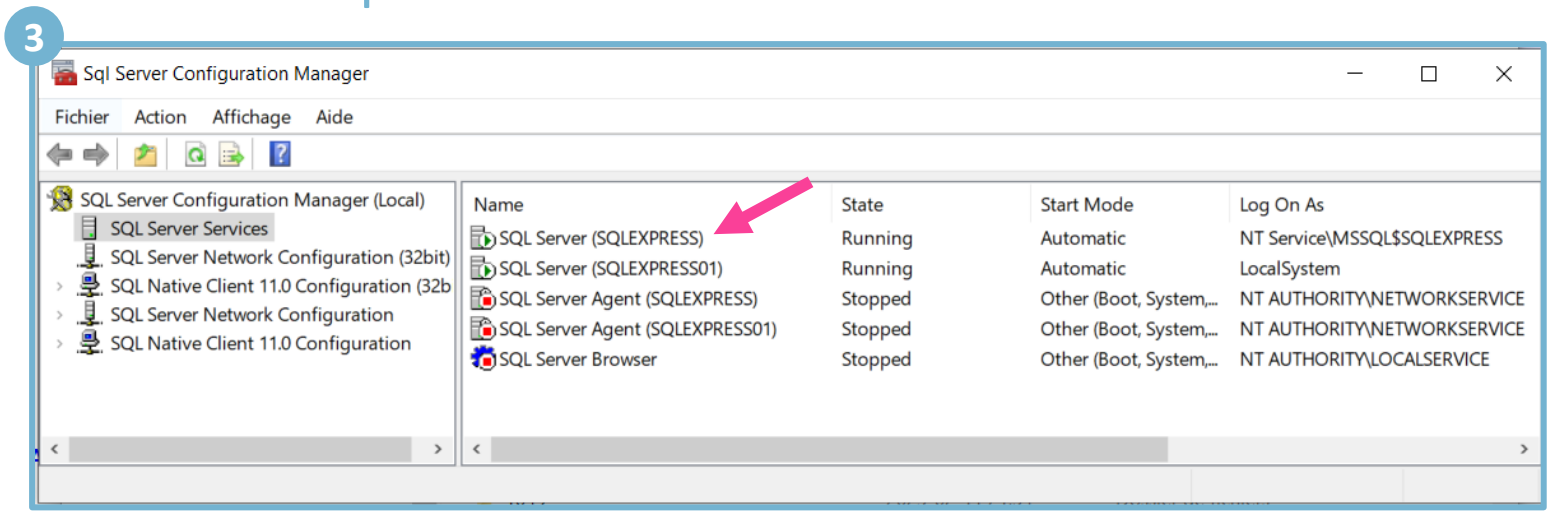
1



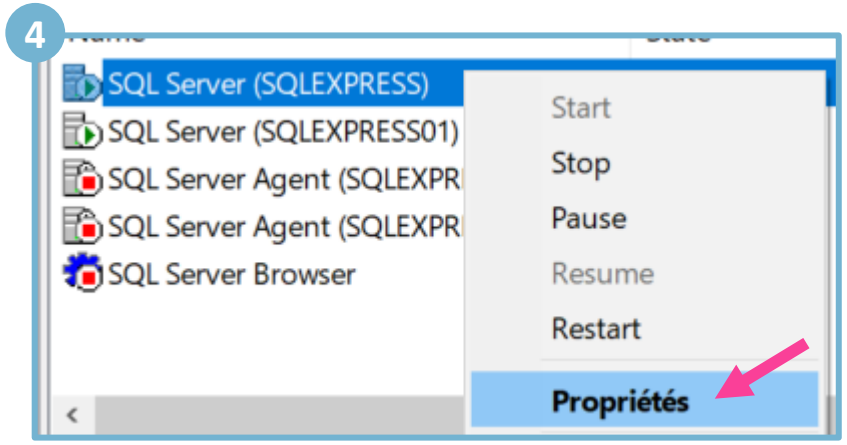
2



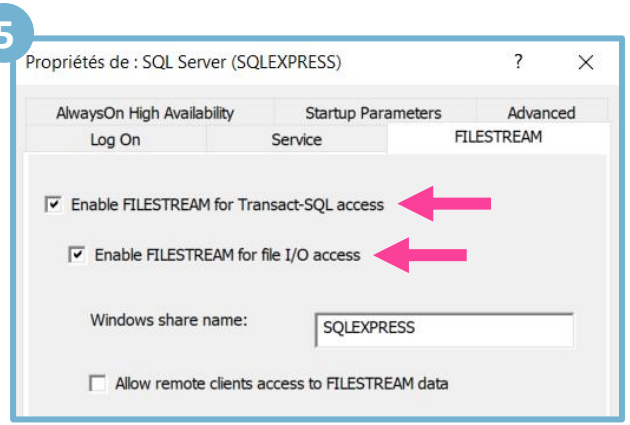
3



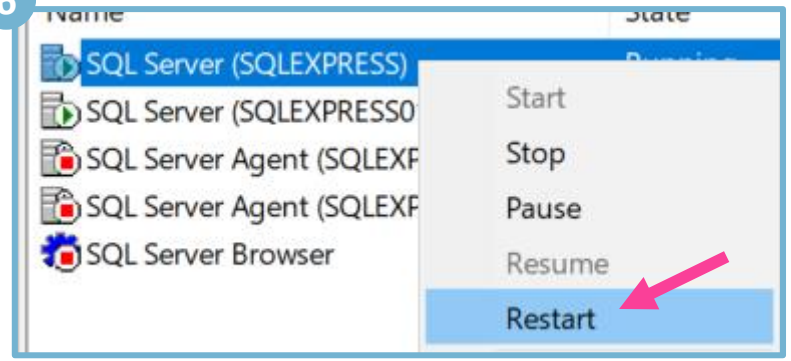
4



5



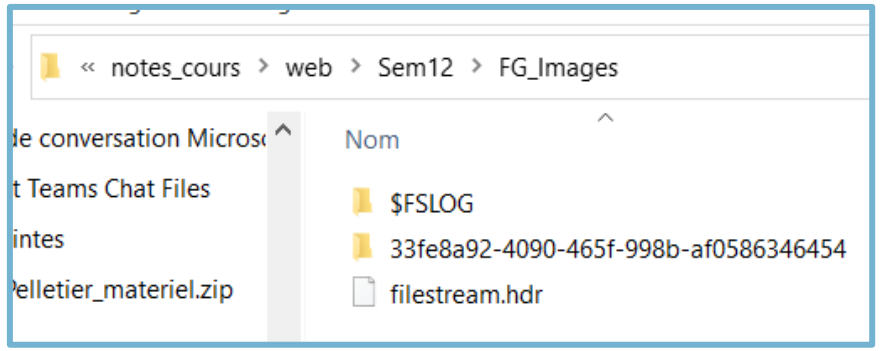
6



Gestion des images

❖ Configurer FILESTREAM avec SQL Server

♦ Voici un exemple de script SQL qui crée une BD et configure un FILESTREAM.



- Assurez-vous que le dossier parent existe
  (Ici, « C:\EspaceLabo »),
  mais pas le dossier à créer.
  (Ici, « FG_Images »)

```
CREATE DATABASE Labo11
GO
USE Sem11
GO

EXEC sp_configure filestream_access_level, 2 RECONFIGURE

ALTER DATABASE Sem11
ADD FILEGROUP FG_Images CONTAINS FILESTREAM;
GO
ALTER DATABASE Labo11
ADD FILE (
NAME = FG_Images,
FILENAME = 'C:\EspaceLabo\FG_Images'
)
TO FILEGROUP FG_Images
GO
```

Gestion des images

❖ Préparer une table qui stockera des images

♦ Les deux colonnes importantes sont Identifiant et FichierImage.

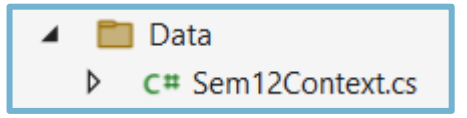
- Sans colonne de type uniqueidentifier NOT NULL ROWGUIDCOL, il sera impossible d'ajouter la colonne de type varbinary(max) FILESTREAM car chaque fichier a besoin d'un identifiant unique (généré aléatoirement dans ce cas) pour être stocké.

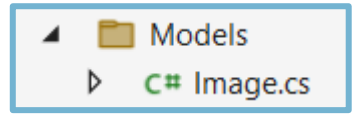
- De plus, ces deux contraintes doivent avoir été créées **avant de pouvoir ajouter la colonne pour le fichier.**

- Enfin, on peut ajouter la colonne qui s'occupera de ranger le fichier et de garder une **référence** vers le fichier.
- On a mis NULL ici car l'image est **optionnelle**, mais c'est libre à vous.
- Gardez bien à l'esprit que l'image **n'est pas dans la BD** : elle est dans le File System, mais la BD s'occupe de l'y ranger.

```
CREATE TABLE Images.Image(
    ImageID int IDENTITY(1,1),
    Nom nvarchar(100) NOT NULL,
    Identifiant uniqueidentifier NOT NULL ROWGUIDCOL,
    CONSTRAINT PK_Image_ImageID PRIMARY KEY (ImageID)
);
GO

ALTER TABLE Images.Image ADD CONSTRAINT UC_Image_Identifiant
UNIQUE (Identifiant);
GO

ALTER TABLE Images.Image ADD CONSTRAINT DF_Image_Identifiant
DEFAULT newid() FOR Identifiant;
GO

ALTER TABLE Images.Image ADD
FichierImage varbinary(max) FILESTREAM NULL;
GO
```

Gestion des images





- Préparer une table qui stockera des images
  - Au moment opportun, on peut ensuite scaffold les **Models** dans notre application Web si la BD est dans l'état souhaité.

```
dotnet ef dbcontext scaffold Name=Sem11 Microsoft.EntityFrameworkCore.SqlServer -o
Models --context-dir Data --data-annotations --force
```

  - N'oubliez pas l'option --force si jamais des **Models** existants doivent être écrasés. (Remplacés)

```
[Table("Image", Schema = "Images")]
[Index("Identifiant", Name = "UC_Image_Identifiant", IsUnique = true)]

public partial class Image
{
    [Key]
    [Column("ImageID")]

    public int ImageId { get; set; }
    [StringLength(100)]

    public string Nom { get; set; } = null!;

    public Guid Identifiant { get; set; }

    public byte[]? FichierImage { get; set; }
}
```

- Remarquons que le type pour le fichier image est un array de bytes et que le type de l'identifiant unique du fichier est Guid.

Gestion des images

❖ Upload une image dans l'application Web

♦ ViewModel et Action

```
public class ImageUploadVM
{
    4 references
    public IFormFile? FormFile { get; set; }

    6 references
    public Image Image { get; set; } = null!;
}
```

- Pour que l'utilisateur puisse envoyer le fichier à l'aide d'un formulaire dans la vue Razor, nous utiliserons un IFormFile qui accompagne l'**objet** à ajouter dans la BD. Dans cet exemple il est optionnel car l'image est une donnée optionnelle dans la BD.

- L'insertion est assez standard, à l'exception qu'on récupère le **fichier** dans le IFormFile et qu'on l'intègre à notre objet sous forme d'array de bytes.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create(ImageUploadVM imageVM)
{
    if (ModelState.IsValid)
    {
        if(imageVM.FormFile != null && imageVM.FormFile.Length >= 0)
        {
            MemoryStream stream = new MemoryStream();
            await imageVM.FormFile.CopyToAsync(stream);
            byte[] fichierImage = stream.ToArray();

            imageVM.Image.FichierImage = fichierImage;
        }
        _context.Add(imageVM.Image);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(imageVM.Image);
}
```

Gestion des images

❖ Upload une image dans l'application Web

♦ Vue Razor

```
public class ImageUploadVM
{
    4 references
    public IFormFile? FormFile { get; set; }

    6 references
    public Image Image { get; set; } = null!;
}
```

```
@model Sem12.ViewModels.ImageUploadVM
```

- On peut commencer par **auto-générer** une vue Razor avec le template Create.

Les modifications les plus importantes sont :
- Utiliser le ViewModel qu'on a créé pour pouvoir utiliser un IFormFile.
- Ajouter enctype= dans le <form>.
- Modifier un peu l'input du fichier pour l'envoyer via le IFormFile. (Car par défaut l'input auto-généré ne permet pas exactement d'envoyer un fichier)

```
<form asp-action="Create" enctype="multipart/form-data">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Image.Nom" class="control-label"></label>
        <input asp-for="Image.Nom" class="form-control" />
        <span asp-validation-for="Image.Nom" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label for="FormFile">Fichier image : </label>
        <input id="FormFile" name="FormFile" type="file" accept="image/*" class="form-control-file" />
        <span asp-validation-for="FormFile" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
</form>
```

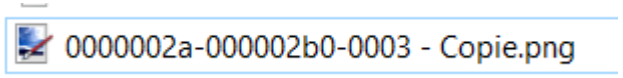
Gestion des images

❖ Upload une image dans l'application Web

- ♦ Coup d'œil dans le File Group :
  - ○ Si vous fouillez un peu dans le dossier qui est utilisé par FILESTREAM, vous pourrez retrouver le fichier de l'image qui été téléversé dans l'application Web.

Nom	Modifié le	Type	Taille
0000002a-000002b0-0003	2023-04-16 15:05	Fichier	463 Ko

  - ○ Si vous êtes vraiment curieux, vous pouvez en faire une **copie**, la renommer avec son extension, et ouvrir l'image :



Gestion des images

❖ Afficher des images dans l'application Web

♦ ViewModel et Action

• Attention, ce n'est pas le même ViewModel que pour **upload** les images. Cette fois le champ supplémentaire servira à stocker un énorme string qui représente l'image.

• Cette action envoie la liste des objets de type Image à la vue Razor, mais elle envoie également tous les fichiers images sous forme d'un groooooos string. (Ça fait une réponse HTTP potentiellement très lourde ! Cela dit, le client n'aura pas à faire des requêtes HTTP supplémentaires pour afficher les images individuellement)

• Il n'est pas toujours stratégique d'envoyer directement plein d'images en même temps que les données importantes qui sont plus légères, mais il est intéressant de voir que c'est possible alors nous abordons cette manière de faire dans ce cours.

```
public class ImageViewModel
{

    public Image Image { get; set; } = null!;
    2 references
    public string? ImageUrl { get; set; }
}
```

```
public async Task<IActionResult> Index()
{
    if(_context.Images == null)
    {
        return Problem("Entity set 'Sem12Context.Images'  is null.");
    }

    List<ImageViewModel> ivm = await _context.Images
        .Select(x => new ImageViewModel
        {
            Image = x, ImageUrl = x.FichierImage == null ? null : $"{Convert.ToBase64String(x.FichierImage)}"
        })
        .ToListAsync();

    return View(ivm);
}
```

Ce bidule « convertit » l'image en string pour l'envoyer

Attention aux fautes de frappe ! Ce string est facile à casser.

Assurez-vous que votre image soit au format **.png**. On aurait pu noter le MimeType lors de l'upload pour gérer plus de formats.

Gestion des images

- Ceci n'est qu'un très court extrait du string qui représente une image envoyée au client. Au total, le string fait 631 900 caractères et pèse autour de 600 Ko pour cette image en particulier.


```
"
56HAXXq7nlaa32nfeRpi6JIkTId4QjZDPkkkqKkC4Ul3dgR/jm+tcM/wRd2+B85QhGWLUva3OLH77jWmrO7qwoF+H3ekUABKAAFVPeca32bemc2KpE5coyRIwHkOxMoVP5//X/+H7PXQJWnzA3I88HGKpxSW
lVVyilA/iJHW6j8bzHo6irlWyijM3aNzWBALjtcSThEr3dqJMg/CqhfTfebvRqd3aYX+hjrz4ye9kyMwpqmJBGYMjfX0BjG+2LNN50DL/OqMdlTErvtkhr5SHnjPl0syYxiqnYyBlVeveiEl+qmeVlNO/kqG
ImVMT3ag7Wtwlucx5uXhO/HC/Wq1TE6Gbu9KHbFFqBuqMNvfNBkbLzyEjc7Iyya+KYMdVwe8P+6k1c6PXRVqzxpNj2wwKSSGgq+G6k5n2o4MGlXqXYEqtJVZNBcPjCX/+AggA4PvRlHkv7isHsDo/ADIA36x
Nj/tUUcIZcYIhwh205TkIYGSZgzaCDlq5q+em8ktVssTn8gGI/q1T4gMD0iV8ZrEi+KIr2qyjjfv9QJHw4P+5wOSo6pAQzWuBDUapv0bwJ6XW6nGoxxa7dd1clPI3T20nXEwMzDbMFBLDh45zovXGh9mZOTu
JXHw6ASY7idM4w1/HHwp9S5Gj6mfcn6MeF8GcW9VLgYbt1Dr2xM9802/7pCAenFZOROyFIKdxJVrBKlRA5Gg8zTUQRJIInxcJDOol6RpJEeuyr0iMOYUiMImsFlQWw6U16gVjAoMq10GWbglPM15nq5KX+5yd
N1bR35QLGrOSMVUrOqSJv31RrungynQ221jrLyHRkM7zUUMyTUsVnsrrlmZOgs+l5Ochy2ti85En8CxAF5P38sxYVD8ycr/R4wjxT3i9NJQoayHS52pootU1wBenbLXMeck/dKIaDbpFrRbDl2IhjXPumS121
4XH7DyRcFluINU/lkg5a+hP9QEFXuBQnd3BEm30mo82fhFcvsvAdvrYbCwD7ddBNS2WoWVSH/oaPelSZPg/L4mM4XZhq7h2ji+RL9v29HNOUpaYsxKQtmHZ6IGSXgGUgMjFd0ORi5lh2DfSVmpvvvfKCQwvY
R0JW6rlKdKkRlOSfDfVCg018brC+PszczTQpgp9gURvWinG3Gt8e2al7T98QX4x4I5kTGD+Ryw2iBQU2kIbj+j7ZKERo6pbnnLXAsqiTSFkbZnhICQ9g0XrJRqE5H+cNHuRZPmYL5kD42V25r30RugLm620jC
nOc3PYtABH6d8fHHYeA8tx2b6FB2Ygvt7JT7TteD6/bsenYpUO9hOfw0xd7S+NP7QA/7K/qOrcwFSgQ7RgTtHnjnGzoemlXDbPyxM9PPLQSG8/n/fEIiBuZS82vzl8P9BSReGXvUQCGXxvzH+6uQeis74/lqn
v3M77thdEb5aP5T+IPDFXJUVTZeRbrDYbn6ZmqHvJVUtxfjxPIqwFdAh+cWhibYmF6kEhFfnXlw/ytEUf8I12SvxuWoK3DA8wETwjFSE4z9E9F8P0YUmpdLXxKxRGQLHRZ2WYcZRdxuktpfm4JYuXIEt10qX6
g/5kSZc1a7066rADwauN4x9pM+JxkiT2uhVRbqEAilvmUu6Ak2TpWkpzv8f9cUsld6Gdvt0PWF+oFDb83TRmjHcf0LAX0Lyx+kKorGRSrpIdUIqaBXF3lD6Nxa9vr8wGlfa+hoS30YJQ02aiVaTxcft2ZCPzM
iST7kvg+RCD02nAnyZI9X0EE+zpG/b0WRYX0Kc6c4MI5WH1chfAJkb+VMPyYxTgK78uJHc7Sa8bqfboeAqtpdpbMCm8To+vSrUkct0FaNd8PE+oiSKO/MKiEs/SOeO0rD+JDZ8TtVNRxoOetSA6nDvRwu6QDr
jNd21OhD3gyJ1QWh9wWqnvg9RErqalZgGde3iNBcefl0B0oVgUdtLoINjMZQ5b6UjsGwNQlCPVipY2BcJN0gFXaS6Vm3sXYjPsotQIBX8NUCv473dG3Bdg+OPzHJL02fWNM7y7UN08nD9DIjDp+N4e64Mb6Zc
Ql7lPpMGDuQeUpTGpOraKwZpfnGxewW4HlcuGV8acibBMSbMuHmRXgfngZHKz3NPCL5tha/p9wzUx5aSz/hzGke46BRthxBnEOkCyQAN2wLRrgUs9GIxIhlAPgTGkARHEC6NQXVtb69jUttrIrxKOzeD/rUyJ
PdXKVhU1I4hmowhBS7t3YhpExo0gVx+jKcluOwCap+uNEPtjQOCiPg4GR6Bx42pIXwhsLY058k0TWn9WD78MiEPKgt7uQiwkNMsDOD3NL0B79H6QcWyMn7MJ4c0cly7b3o45cf0c/CnAVcML1K80vC+Ar1CRwi
+nzwWGsF5NbApSelWzabHyc/enhXNPbO+867/I9xHOpf8dkT8Hp0mDDUcwJolWLlZJ4Uw4hhNpGpMyC/ulTPYV0L5e3ojL/keJVRup+BJEZ7y2xpSGWbi5+cvs4rf+WJMjXdPGWXBBo0xMiKGQisAyx1J7IxW
lkCq6ValoSL6HQZk5DX94jB5+KmyX93ZvwGINCiALsGlvGr6qvtTDNs6Gf4hxbCz2TXdOjCHUJ88r+czZeNUmaL5/twCp6VyoRbCoSF8amF7s8804j1/q8OWANhdF47VGfAyYuBFqGAsJTVJJKszzdc2E6kSQ
Q+CMCEJTxlRCWmWrdTdJ50tgFyJnW7gS4Z2H8HNRkoFeSTupNgkEcZyTQuek2k7qNZ/blw4iSkYqeTVIobSuVud009Fu3xKs9HMh2agRHajTMtCmrYIZkRC4kL1WUCJRLEENaw/WQPNomwpeGxj/Nwu9MCrfX
OLmo2nVTh0NMd6BhconoLN96RPigxh06Vxin78kwp8xo5cnxxlxbixHczVtoHlsAV3QeC+GLcw3EYhGwWV6Hn9GYPBLIebwejt5JCxEnH6vfTCHf01qXwLOqSKJKkXatPLrjNSTH1HlU12RqQVeHZ1R5qqiK0
zau44YrxvSdSzsq3XOUjyoeV7LMaSDzPD16NUHyopfotdNX92tEchORCHSawBFBGEQXBrbj5P34LZmUSi4xZ9oK6T9DlKVkAqaohFb1IxgouqHh+Lbpx3aHXR4uheuRVs3gVDYpARH5bx7DYgPSS+5tF3BsDP
zXXw1ELow2zHOCWE0ke3Axf4fJ7cepqISCmQiUl3UwLuBZb3X9HYjkuWlBAX6RCM3ncYcmStIFi+A7hkpSc+C4g5FSAM0MQrobZlYGL9XQrjYpC7dSSkY0rZmTspE2mQphQAJmsWKVE9DO0lMVKyx+JnBOTUrt
JK5bQTqw2sYN6vtN2P+aqdUPIzm4vBSpGFqzL4amJATUslC2J/kVyr6PKi7HmnkTZuWS5mXAdVYko0phDOIXRGcQjQX0n4Xqa6pDb/Szl8DqCuRAu18DxNVwsjJFhNAk7pQHDuxHUcj2XcCndebvwQ3XrheBD
rVu54yNUZfx7ubKkWhpuUk0sy4pIRUd66NR0Cm9PSQ/l/+H/+HlH0ZogsLLF+Bu/FKI9lScz6mFa7IjCH1Zla3EIA8pewk6qOigVpNNiS0kWmNeGBVd6Kjkpzbj Hue5JMAMRHFSUtEUukJY6lyrz9BgPR/X+R
tziFULj0W73Eqs7zUf56dt5ARkg414Wmxmo5YF72qzq6vJ5eQ9ZRrRilt3KURWPMq+nQL4uo2hlQZTrS6MAHGtz3GlwbOR+2lzrQ7oz+XbW+GjkkdA5dIZ8MU0mvk9DcoOdtJ2o9K5pWOz9cCB1OnvzIb2ud5
+nqIo6SZA240f3lBacPaW/+x70pPHvtXNL8SwoHZw4tHvSbK9Ju3POeKNwS3mte/AQNGaj0jSuGP5cLooL6lCCXDqpoILBnEKYXCsKpud3oibkIRH4OKY7rSKc4x2m3cwMW+Qlh2FFE7IdBDEn49/uROciP/d
WD/RmJyFeXowI+dEtM4d2qkgyIZYHiY++gvNbI31VC0I6ovD8K6OxcHR77ybtX0t2cAeRcpyPqUWEpVts7zjUZSmdNJAuyKS+X5erVa8Rv0eCE/VFeWVZHnp1WuGYXd+ogn1fbgTNB6aNZJHMvfGpTZEAs0mU
C7SujsjlylW/ypK8DOKIWy5pfwp1F/AnVlp7PDOA9tWxAymAjhEPrEJZTM6I8krgvNg2z23TBm8Cd8SG3P3T/jRU5aH3+vyp8EjmQ6x2F9u3uvBZ+JY0OwDIPdkX71t1NRy00e+cPaXhtyYPbY4s6lfHN5PKP
l+UCt0fyv9TqmXVwYJa7VavAasR32q615iZXpXoIBbRSkhr3ali4Z/Q1PrOnCDk/otK+BnnNFsjWCqJ0Q6CEJD/OnpgPTJ0kbTSfVZL4X47B/g1GZBG5cDtUZY6fEAtRTWrub8KfhMe4DwZSdjYgox0K/ad0d
bLfNq0VUtcs7AtFWclW5EsHJuR+3y7Jtvtpttw+rle54zI6FQinatFuvt+JPsi6KVJ5OJkpFlqn6mOUnESOxqKxSSfRO3uiCriS9pfmJ8kWWSwCalcRgXYFo1c600SkcObgSfxpkGxT0KdRZp3yN3G2HrexdaU
f9KJngUJ06XN18VH8TwA7C1ORZgNWUfSlkZ9CN0S63IIFz2wuHX8dJz3UXelwQFiO+nZMrAn7RyJjTr2VIHb9Ra/LHSi5CuATDjfmfj2r6dSqYPu6Htb0Il+amHOijf4UMDEwHLc8va61kSc8GrTZYGKVkvTl
Sl8d42p2OP7N9u6IzONfG3rlh127/eqK64bk1rur95YgEdHYqzuaGELUTAj0k4cSfrgZr+mwRmk5a6zXp65CCHyl/CriL/NmY159m9RkHU/YVUHenc3WwY3bPApGJFFBgmxI/vVRusnyX5w95fr/ZvN3t3q3W
d4dC/Em+ejkq3+iCmX5C0j8iKSYkzgR/yo5ldajKYywv0Ttu0uk//UrFKjuu8oMyWS5JbcWflCIIPk84+Os86J88tXDCyJE1xZ8EyuJka1TVJ91Z9RhsMakfwxUalTolievmumhsD+EyViD1rD4LhiyqbKkjY
9CxMOjGHJ4hgXPbC3+VuarhOi7Xn1B67XI3jnTNjp1XRLpqLVbc8ae59E1eA6+g09b8SZ9R2FbKAX9R2OAlDtTojdvoaXAFOBJtLzUs50/DM/koJD3RYDx+g+grk6sLwzwl/dWq7NMytS2vRhp2FWqn15H6k0
xfhfYHos6BT+mYq+g6V0X7Sy2BpnZM4BJJOP8//9//9lEwE2mCujDU7qQVT80nM6DWi/kT4leOqKkDaCnURUXDHZ+vFvGUfQXUdhMfElLG60/K8NQ2GQQaJyOvQhGjbVZts+wuyx9W63er1bvN9kO+ehCdKrNN
lW/KKi9O2UnNy6oUXzLU7/U6U8qqw/H4eCyeRKRWa/VL1EpC3KQTnVplh3X+aZV/zLPnPD/AnxLHUnIcYFpyJBalhMHgDBb2AH+KfxMgCvXR4aEjM0e7EIfzhPoQGEbiiK+OQeKCtbYrZ8MauTr3ev5I6aAbDG
7KjkICF23dyFe+SC9GeZoYtRvxeQbUw9P7T/MSlAu653Hu5hchNaz5k8CH/2I/nmSIwkunb7ZbQ8pfadzO18ZLNJfEM1qeX9a+Mn8SbG1mrPrKWjGeGe/r/X3hwPWbDwV/zITLB6oovQiir3Io15STirpA27lm
oOMtqPbCwBWkfi3mTwl6E0avk85OzSnXoJa38Kdrwz8dx6VQ//SP56JTwXXgY8q+Atwd4t4cAZGhRBSqquJbcir0tpasspWIUV5t89X9evVuvXm/Xr/LV+JPb6vVQ1luTtW6Wu3y9a4ss8PxdCrTc0+ozvPNZ
r3dbjab1al4fnz87nh8Epfa3cnE6Xjcn06FqFYuRpU9rrLfb/Lf5qtHUSju3zHXNsmPHnH7T+wnLv3aXgZnOmKugoclPjQOySB2lnJuWnsDDuXpexYTdZ9nuoOYXsJXlgFo/GonXtEfKR1041b+ZNjTV3Lyc/C
nxjFUv4aTLXiQap3LdA8MfKkT+CYP08HS4k+BZqdtKgrPx5fQ2ZmJnvevNG5c9MZU+frZRcvxy9rX50/C7FCNKmOQppQ0rQaau2GUs7113Fo4N+fzrLaNMRMuH6ii9MJxT7qUDvAnF3StDAx0C8yJKTsXkqfJ
7fxJaE8Zw53syiwBTYeCOQV8GBmbGtNxXIro32rJOYWPKftSYL3iCXFlmq0TDq15pGm1y7JNxQqTyM06q3jUSRESeVLD1erNbvdhd/eT3e6b1fp9ld0Xp+2p2hZlXpTrMltX+bosuW0n/sRhqjb5ar2GPCmJDD
3vH5+fPlblUeRpd7cpy0L8qSiO4k9Zecizj+vsV9vVv1zl8KcqO0kDC1kVC046LlY8HcVzVPkqrvJNauOypI00T187zJISkcqe3LT2BtHqmpWR+v+RPwlDAp+dP00/Py5D7THDLrs6SicatdAdWcdXf9bwKg+Y
Q156BGYR5EHTWO4sVsQ5PzhGilJXW9pp2VhoTkGbF/P5SPYnNA9MBy23L2s/C38S5sVqVJlaj2u44oIbnmUWjlofNE8KhulO6HfVQC2loemsWX86BJXh//VRekY/GsOetBCz5yIknS/iT0JcfeghOy7qYmzKGS
49Q/WL+ZOAJ+MjNHjCvAB2UhdRBvNahxJSqF6MiI+u381bl8SQxGngSSrMMnGm7Sq/z/P7qoJFZZWIFMypOMmB9d3dm/v7r7a7r/PVh3z9rszelNm2rLaHY7k/lic0r0g4y0zAacFtu/Vmna/E0MrseDwdDs9VV
azX+Xoib0SzChafVMctvMJLUP/dOv9/rTM/HaULcf2CAlGolOrytMqPef68yq/p2SmAODczqWX8SWKXcldtNLi69dJhN7qz7OthIX/iL9zwaL4OOKZSVihrfRF/EnwWi9bOx/Dz46+MF5mYvF6dkcS00equsJZ
```


Le string n'est-il pas aussi *mignon* que l'oiseau qu'il représente ? ♡

Gestion des images

❖ Afficher des images dans l'application Web

♦ Vue Razor

```
public class ImageViewModel
{

    public Image Image { get; set; } = null!;
    2 references
    public string? ImageUrl { get; set; }
}
```

```
@model IEnumerable<Sem12.ViewModels.ImageViewModel>
```

On peut auto-générer une vue avec le template List pour notre Model. Les changements les plus importants à faire sont :

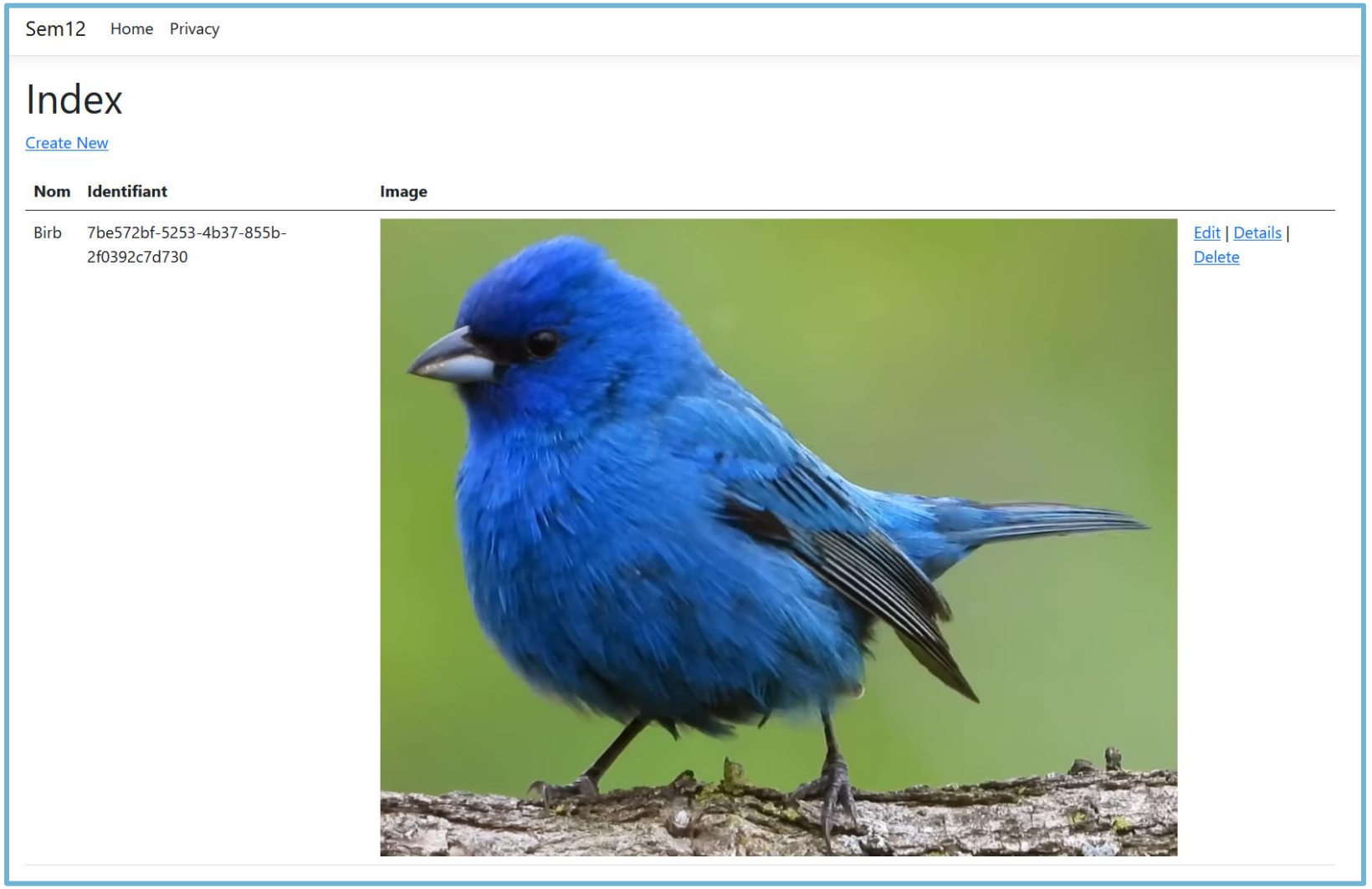
- Utiliser le ViewModel qu'on a créé pour recevoir les strings qui représentent les images.
- Afficher l'image dans un élément <img> tel que dans l'exemple. Ce n'est pas très complexe : on intègre le string dans l'attribut src.

```
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Image.Nom)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Image.Identifiant)
            </th>
            <th>
                Image
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Image.Nom)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Image.Identifiant)
            </td>
            <td>
                <img alt="@(item.Image.Nom)" src="@(item.ImageUrl)" />
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.Image.ImageId">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.Image.ImageId">Details</a> |
                <a asp-action="Delete" asp-route-id="@item.Image.ImageId">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

Gestion des images

- N’hésitez pas à redimensionner les images avec du CSS.

- Dans le cours Prog Web Services (4W6), redimensionner des images sur le serveur pour alléger les fichiers à envoyer au client est déjà abordé, alors nous ne le ferons pas à nouveau dans ce cours.



Gestion des images

❖ Supprimer les images

- ♦ Vous remarquerez peut-être qu'après avoir supprimé une rangée qui contient un fichier avec FILESTREAM dans la BD, le fichier ne disparait pas du File Group... Est-ce un fichier zombie qui traînera pour toujours ?! 🧟😨
  - ○ Pas de panique 😌 : FILESTREAM vient avec un système de *Garbage Collection* discret qui supprime les fichiers jugés obsolètes après un certain nombre de transactions.
    - Pas besoin de comprendre en détails, mais gardez à l'esprit qu'il ne faut pas compter sur le fait que les fichiers restent, mais il ne faut pas non plus espérer qu'ils soient supprimés instantanément.
  - ○ Si vous souhaitez expédier le *Garbage Collector* pour voir qu'il supprime bien les images obsolètes, vous pouvez utiliser ce code SQL sur votre BD :

```
CHECKPOINT
GO

EXEC sp_filestream_force_garbage_collection  'Sem12'
GO
```

- Remplacez Sem12 par le nom de votre BD, entre apostrophes.