Secteur Tertiaire

Formation CDA

Module 3 - Développer une application web

Activité 2 - Développer la partie backend

API

1. Sommaire

2. Prérequis

2.1. Base de données

Créer une base de données « Jeux Video » sur MS SQL Server ou MySQL.

Elle contient 3 tables :

- consoles avec 3 champs :
  - id, entier, clé primaire, auto incrémenté ;
  - marque, chaine de 45 caractères, non null ;
  - modele, chaine de 45 caractères, non null ;
- genres avec 2 champs :
  - id, entier, clé primaire, auto incrémenté ;
  - nom, chaine de 45 caractères, non null ;
- jeux avec 4 champs :
  - id, entier, clé primaire, auto incrémenté ;
  - nom, chaine de 45 caractères, non null ;
  - dateDeSortie, date, non null ;
  - genre, entier, non null, relié au id de la table genre.

2.2. Outils

Installez l'extension **vscode-solution-explorer**

Tapez dans le terminal (Terminal > new Terminal) ;

`dotnet tool install -g dotnet-aspnet-codegenerator` [1]

`dotnet tool install -g dotnet-ef`

Si par la suite, vous avez un message disant que les outils ne sont pas à jour, il faudra faire :

`dotnet tool list -g` [2]

`dotnet tool update -g dotnet-xxx` [3]

XXX étant le nom de l'outil à mettre à jour.

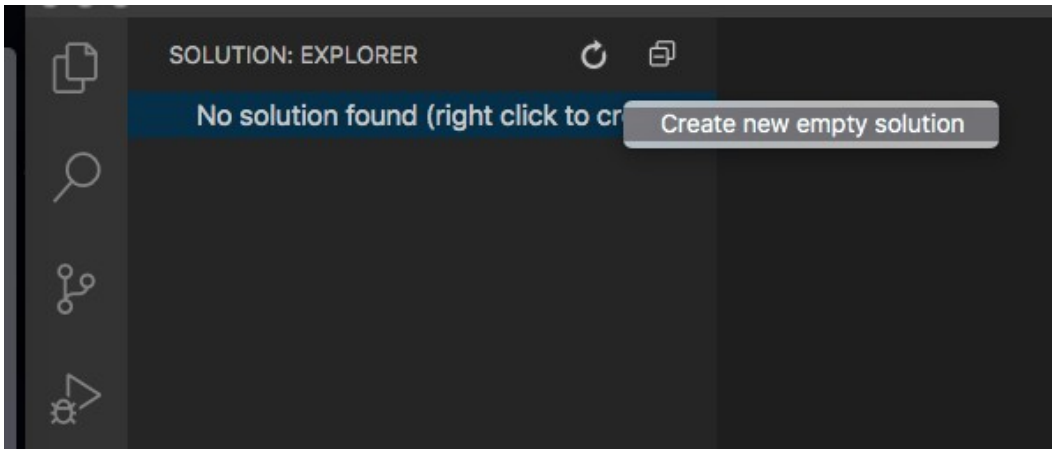
1 https://docs.microsoft.com/fr-fr/dotnet/core/tools/dotnet-tool-install
2 https://docs.microsoft.com/fr-fr/dotnet/core/tools/dotnet-tool-list
3 https://docs.microsoft.com/fr-fr/dotnet/core/tools/dotnet-tool-update

3. Création de la solution

Dans le terminal, rendez vous dans le dossier où vous souhaitez créer la solution et tapez :

`dotnet new sln -n JeuxVideo` [4]

Vous pouvez aussi passer par l'interface :



4. Gestion du projet

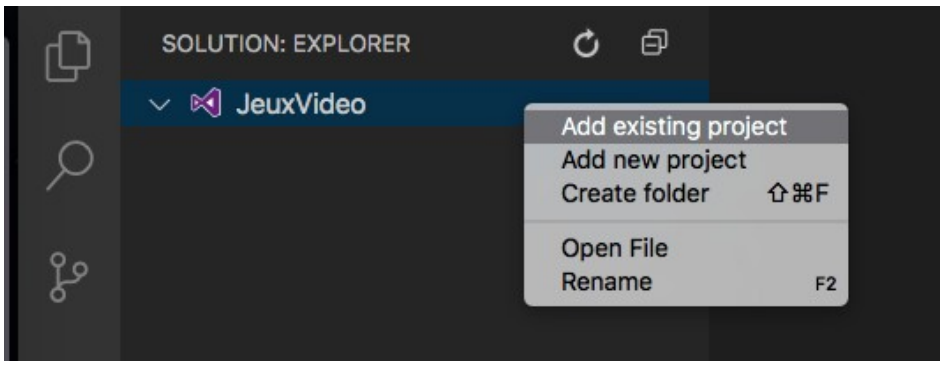
4.1. Création et ajout du projet à la solution

Dans le terminal, tapez :

`dotnet new webapi -o WebApi` [5]

`dotnet sln add WebApi\WebApi.csproj`

Vous pouvez aussi passer par l'interface :



4 https://docs.microsoft.com/fr-fr/dotnet/core/tools/dotnet-sln
5 https://docs.microsoft.com/fr-fr/dotnet/core/tools/dotnet-new#webapi

4.2. Ajout des packages

Dans le terminal, tapez :

```
cd WebApi
```

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```
[6]

```
dotnet add package
Microsoft.VisualStudio.Web.CodeGeneration.Design
```

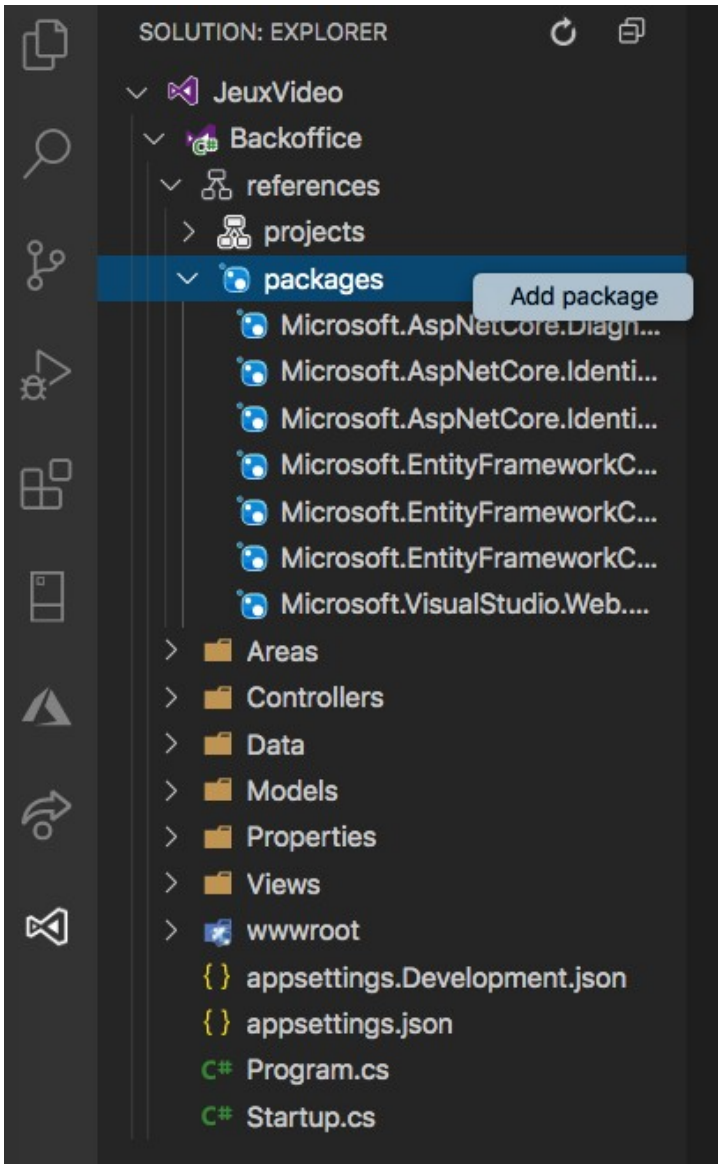
```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Si vous utilisez MySQL, ajoutez :

```
dotnet add package Pomelo.EntityFrameworkCore.MySql
```

Vous pouvez aussi passer par l'interface :

6 https://docs.microsoft.com/fr-fr/dotnet/core/tools/dotnet-add-package



4.3. Nettoyage

Supprimer les fichiers d'exemple **WeatherForecast.cs** et **Controllers\ WeatherForecastController.cs.**

4.4. Configuration

Modifiez le fichier **appsettings.json** en ajoutant :

```
"ConnectionStrings": {
    "ApiDbContextConnection": xxx
}
```

où xxx est la chaine de connexion [7] vers votre base de données JeuxVideo.

Modifiez le fichier S**tartup.cs** en ajoutant avant services.AddControllers() :

7 https://www.connectionstrings.com/

```
services.AddDbContext<ApiDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("ApiDbConte
xtConnection")));
```
ou

```
services.AddDbContext<ApiDbContext>(options =>
options.UseMySql(Configuration.GetConnectionString("ApiDbContextCo
nnection")));
```
si vous utilisez MySQL.

4.5. Création du context

Créez un dossier Data puis un fichier **ApiDbContext.cs** à l'intérieur contenant :

```
using System;
using Microsoft.EntityFrameworkCore;

namespace WebApi.Data
{
  public partial class ApiDbContext : DbContext
  {
    public ApiDbContext()
    {}

    public ApiDbContext(DbContextOptions<ApiDbContext> options)
        : base(options)
    {}
  }
}
```
Dans le fichier **Startup.cs**, ajoutez :

```
using WebApi.Data;
using Microsoft.EntityFrameworkCore;
```

4.6. Récupération de la structure des données

Le scaffolding permet de générer les modèles directement depuis la base de données.

Dans le terminal, tapez en remplaçant xxx par la chaine de connexion vers la base JeuxVideo :

```
dotnet ef dbcontext scaffold "xxx"
Microsoft.EntityFrameworkCore.SqlServer -c ApiDbContext --context-
dir Data -o Models -f
```
[8]
ou

```
dotnet ef dbcontext scaffold "xxx"
Pomelo.EntityFrameworkCore.MySql -c ApiDbContext –context-dir Data
-o Models -f
```
si vous utilisez MySQL.

Un fichier **ScaffoldingReadMe.txt** peut apparaître à la racine du projet.

8 https://docs.microsoft.com/fr-fr/ef/core/managing-schemas/scaffolding?tabs=dotnet-core-cli

Afin de sécuriser la connexion à la base, modifiez le fichier **Data\ApiDbContext.cs** en supprimant la ligne :

```
#warning To protect potentially sensitive information in your connection string, you should move it out of source code. See http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing connection strings.
```

Et la suivante :

```
optionsBuilder.UseSqlServer("xxx", …
```

4.7. Data Annotations et Fluent API

Lors de la commande scaffold, nous avons utilisé l'option -d qui utilise les data annotations pour configurer le modèle [9]. Si elle est absente, c'est Fluent API qui est utilisée [10].

4.8. Création des controllers

Dans le terminal, tapez :

```
dotnet aspnet-codegenerator controller -name ConsolesController -async -api -m Consoles -dc ApiDbContext -outDir Controllers
```
[11]

```
dotnet aspnet-codegenerator controller -name JeuxController -async -api -m Jeux -dc ApiDbContext -outDir Controllers
```

```
dotnet aspnet-codegenerator controller -name GenresController -async -api -m Genres -dc ApiDbContext -outDir Controllers
```

4.9. Premier lancement

Vous pouvez tester un premier lancement en tapant dans le terminal :

```
dotnet run -c Debug
```
[12]

4.10. Migration de la base JeuxVideo

Les migrations permettent de synchroniser les modifications effectuées dans le code et la base de données [13].

Dans le terminal, tapez :

```
dotnet ef migrations add InitialCreate -c ApiDbContext -o Data/Migrations
```

puis supprimer le contenu de la méthode Up de la migration apparue dans Data\Migrations et appliquez la :

9 https://docs.microsoft.com/fr-fr/ef/core/modeling/#use-data-annotations-to-configure-a-model
10 https://docs.microsoft.com/fr-fr/ef/core/modeling/#use-fluent-api-to-configure-a-model
11 https://docs.microsoft.com/fr-fr/aspnet/core/fundamentals/tools/dotnet-aspnet-codegenerator?view=aspnetcore-3.1
12 https://docs.microsoft.com/fr-fr/dotnet/core/tools/dotnet-run
13 https://docs.microsoft.com/fr-fr/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli

```
dotnet ef database update -c ApiDbContext
```

4.11. Documentation

Nous allons utiliser Swashbuckle [14] afin de documenter notre API. Dans le terminal, tapez :

```
dotnet add package Swashbuckle.AspNetCore
```

Dans le fichier **Startup.cs**, ajoutez :

```
using Microsoft.OpenApi.Models;
```

et après services.AddDbContext() :

```
services.AddSwaggerGen(options =>
    options.SwaggerDoc("v1", new OpenApiInfo {
        Title = "My API", Version = "v1"
    })
);
```

puis dans la méthode Configure :

```
app.UseSwagger();
app.UseSwaggerUI( options =>
    options.SwaggerEndpoint("/swagger/v1/swagger.json", "My API
V1"));
```

Relancez dans le terminal :

```
dotnet run -c Debug
```

et rendez vous sur https://localhost:5001/swagger/ pour tester.

4.12. Sécurisation des accès

Si vous relancez

```
dotnet run -c Debug
```

et que vous essayez d'accéder à l'URL /Jeu sans être connecté, vous verrez que c'est possible. Il faut donc sécuriser l'accès aux URL de l'API. Nous allons utiliser le service Auth0 [15].

Créez et configurez votre compte sur `https://auth0.com/`.

Installez le package en tapant dans le terminal :

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
```

Ajoutez au fichier **appsettings.json** :

14 https://docs.microsoft.com/fr-fr/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-3.1

15 https://auth0.com/docs/quickstart/backend/aspnet-core-webapi/01-authorization

```
"Auth0": {
    "Authority": "https://xxx.eu.auth0.com",
    "Audience": "https://localhost:5001/"
}
```

où xxx est lle nom de votre compte Auth0.

Ajoutez au fichier **Startup.cs** :

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
```

puis après services.AddDbContext() :

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
.AddJwtBearer(options =>
    {
        options.Authority = Configuration.GetSection("Auth0")
["Authority"];
        options.Audience = Configuration.GetSection("Auth0")
["Audience"];
    });
```

Ajoutez aux controllers :

```
using Microsoft.AspNetCore.Authorization;
```

puis juste après [ApiController] :

```
[Authorize]
```

Si vous ne voulez pas d'authentification sur toutes les méthodes, il est préférable d'ajouter [Authorize] au dessus des méthodes Put, Post et Delete seulement. Ainsi, les Get sont accessibles sans Token.

Modifiez **Startup.cs** :

```
services.AddSwaggerGen(options => {
    options.SwaggerDoc("v1", new OpenApiInfo {
        Title = "My API", Version = "v1"
    });

    OpenApiSecurityScheme securityDefinition = new
OpenApiSecurityScheme() {
      Name = "Bearer",
      BearerFormat = "JWT",
      Scheme = "bearer",
      Description = "Specify the authorization token.",
      In = ParameterLocation.Header,
      Type = SecuritySchemeType.Http,
    };
    options.AddSecurityDefinition("jwt_auth", securityDefinition);

    OpenApiSecurityScheme securityScheme = new
```

```
OpenApiSecurityScheme() {
        Reference = new OpenApiReference() {
            Id = "jwt_auth",
            Type = ReferenceType.SecurityScheme
        }
    };

    OpenApiSecurityRequirement securityRequirements = new
OpenApiSecurityRequirement() {
        {securityScheme, new string[] { }},
    };
    options.AddSecurityRequirement(securityRequirements);
});
```

puis ajoutez juste avant app.UseAuthorization() :

```
app.UseAuthentication();
```

4.13. CORS

Les navigateurs interdisent l'appel vers un autre domaine que celui du site d'origine, ce qui est gênant pour notre API. Nous allons remédier à ça en activant CORS. [16]

Ajoutez aux services dans **Startup.cs :**

```
services.AddCors(options => {
    options.AddDefaultPolicy( builder => {
        builder
            .WithOrigins("https://localhost:5001")
            .AllowAnyHeader()
            .AllowAnyMethod();
     });
});
```

puis :

```
app.UseCors();
```

Pensez à adapter le port en changeant 5001 si vous en utilisez un autre.

4.14. Proxy

Pour améliorer les performances, nous pouvons demander à ce que les requêtes vers la base de données soient effectuées au dernier moment [17].

Ajoutez le package en tapant dans le terminal :

```
dotnet add package Microsoft.EntityFrameWorkCore.Proxies
```

Modifiez **Startup.cs** :

16 https://docs.microsoft.com/fr-fr/aspnet/core/security/cors?view=aspnetcore-3.1
17 https://docs.microsoft.com/fr-fr/ef/core/querying/related-data#lazy-loading

```
services.AddDbContext<ApiDbContext>(options => options
    .UseLazyLoadingProxies()
    .UseSqlServer(Configuration.GetConnectionString("ApiDbContextC
onnection"))
);
```

5. Gestionnaire d'images

5.1. Création du modèle

Créez un fichier **Images.cs** dans Models :

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace BackOffice.Entities
{
    [Table("images")]
    public partial class Images
    {
        [Key]
        [Column("uid")]
        [StringLength(36)]
        public String Uid { get; set; }

        [Required]
        [Column("type")]
        [StringLength(32)]
        public string Type { get; set; }

        [Required]
        [Column("data")]
        public byte[] Data { get; set; }

        [Required]
        [Column("created")]
        public DateTime Created {get;set;}

        [Required]
        [Column("width")]
        public int Width { get; set; }

        [Required]
        [Column("height")]
        public int Height { get; set; }
    }
}
```

Modifiez le fichier **Data\ApplicationDbContext.cs** en ajoutant :

```
public virtual DbSet<Images> Images { get; set; }
```

et :

```
modelBuilder.Entity<Images>(entity =>
{
    entity.HasKey(e => e.Uid);

    entity.ToTable("images");

    entity.Property(e => e.Uid)
        .HasColumnName("uid")
        .HasMaxLength(36);

    entity.Property(e => e.Created)
        .HasColumnName("created")
        .HasDefaultValueSql("(getdate())");

    entity.Property(e => e.Data)
        .IsRequired()
        .HasColumnName("data");

    entity.Property(e => e.Height).HasColumnName("height");

    entity.Property(e => e.Type)
        .IsRequired()
        .HasColumnName("type")
        .HasMaxLength(32);

    entity.Property(e => e.Width).HasColumnName("width");
});
```

Modifiez ensuite le modelBuilder des Jeux en ajoutant :

```
entity.Property(e => e.Image)
    .HasColumnName("image")
    .HasMaxLength(36);
```

et :

```
entity.HasOne(d => d.ImageNavigation)
    .WithMany(p => p.Jeux)
    .HasForeignKey(d => d.Image);
```

Modifiez le fichier **Jeux.cs** pour ajouter une propriété image :

```
public string Image { get; set; }

public virtual Images ImageNavigation { get; set; }
```

5.2. Migration

Créez et jouez une migration pour ajouter la table images :

```
dotnet ef migrations add Images -c ApplicationDbContext -o Data\
Migrations

dotnet ef database update -c ApplicationDbContext
```

5.3. Contrôleur des jeux

Modifiez **Controllers\JeuxController.cs** pour inclure les images :

```
public async Task<ActionResult<IEnumerable<Jeux>>> GetJeux()
{
    var jeux = _context.Jeux
        .Include(j => j.GenreNavigation)
        .Include(j => j.ImageNavigation);

    return await jeux.ToListAsync();
}
```

et :

```
public async Task<ActionResult<Jeux>> GetJeux(int id)
{
    var jeux = await _context.Jeux
        .Include(j => j.GenreNavigation)
        .Include(j => j.ImageNavigation)
        .FirstOrDefaultAsync(j => j.Id == id);

    if (jeux == null)
    {
        return NotFound();
    }

    return jeux;
}
```

6. Améliorations

6.1. Boucle récursive

Si vous appelez api/jeux, vous verrez que le genre se résume à l'identifiant et ne donne pas son nom. Nous allons corriger ça.

Modifiez **Controllers/JeuxController.cs** :

```
// GET: api/Jeux
[HttpGet]
public async Task<ActionResult<IEnumerable<Jeux>>> GetJeux()
{
    var jeux = _context.Jeux.Include(j => j.GenreNavigation);
    return await jeux.ToListAsync();
}
```

Si vous relancez

```
dotnet run -c Debug
```

vous verrez qu'une erreur se produit parce que le jeu appelle le genre et que le genre liste les jeux qui lui sont rattachés. Ces jeux ont à leur tour un genre et on arrive à une boucle. Corrigeons ça.

Ajoutez le package :

```
dotnet add package Microsoft.AspNetCore.Mvc.NewtonsoftJson
```

Modifiez services.AddControllers() dans **Startup.cs** :

```
services.AddControllers().AddNewtonsoftJson(options =>
{
    options.SerializerSettings.ReferenceLoopHandling =
        Newtonsoft.Json.ReferenceLoopHandling.Ignore;
});
```

6.2. Éléments inutiles

Nous voyons que l'image d'un jeu liste en retour le jeu auquel elle est rattachée, ce qui n'est pas très utile.

Modifiez **Images.cs** en ajoutant :

```
using Newtonsoft.Json;
```

puis en modifiant :

```
[JsonIgnore]
public virtual ICollection<Jeux> Jeux { get; set; }
```

Par ailleurs, sur un jeu, nous voyons que nous avons le genre et genreNavigation, ce qui fait double emploi. Nous allons masquer le genre et renommer genreNavigation avec PropertyName. Pour ça, modifiez **Models\Jeux.cs** :

```
using System;
using Newtonsoft.Json;

namespace WebApi.Models
{
    public partial class Jeux
    {
        [JsonProperty(Order = 1)]
        public int Id { get; set; }

        [JsonProperty(Order = 2)]
        public string Nom { get; set; }

        [JsonProperty(Order = 3)]
        public DateTime DateDeSortie { get; set; }

        [JsonIgnore]
```

```
        public int Genre { get; set; }

        [JsonIgnore]
        public string Image { get; set; }

        [JsonProperty(PropertyName = "genre", Order = 4)]
        public virtual Genres GenreNavigation { get; set; }

        [JsonProperty(PropertyName = "image", Order = 5)]
        public virtual Images ImageNavigation { get; set; }
    }
}
```

6.3. Ordre

Pour faire apparaître les propriétés dans un ordre précis dans le JSON, vous pouvez utiliser Order. Un exemple avec **Models\Genres.cs** !

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

namespace WebApi.Models
{
    public partial class Genres
    {
        public Genres()
        {
            Jeux = new HashSet<Jeux>();
        }

        [JsonProperty(Order = 1)]
        public int Id { get; set; }

        [JsonProperty(Order = 2)]
        public string Nom { get; set; }

        [JsonProperty(Order = 3)]
        public virtual ICollection<Jeux> Jeux { get; set; }
    }
}
```

7. En cas de souci

Avant toute chose, commencez par tapez dans le terminal ;

```
dotnet clean
```