

**Avant de démarrer ce TP, réaliser la partie 1.1 du TP3 afin de bien comprendre comment créer des clés étrangères et les propriétés de navigation.**

L'application développée dans ce TP permet à un utilisateur de déposer des notes sur des films (un peu comme IMDB ou TheMovieDB). Plus précisément, elle permet de :

- Créer un compte utilisateur
- Rechercher un compte utilisateur et visualiser ses informations
- Rechercher un film
- Visualiser un film
- Créer un film
- Noter un film

### 1. Création du projet

Créer un nouveau projet ASP.NET Core Web API (Cf. TP1) **SUR LE BUREAU** :

- Projet « API Web ASP.NET Core ».
- Sélectionner le framework **.NET 8**.

**Créer un nouveau dépôt GitHub et y ajouter l'application.  
Committer au fur et à mesure.**

### 2. Couche Modèle

#### 2.1. Packages NuGet

Installer les packages `Microsoft.EntityFrameworkCore.Tools` et `Npgsql.EntityFrameworkCore.PostgreSQL` dans une version compatible avec votre version de .NET Core (par exemple, la version 8.0.11 sur .NET 8). Utiliser TOUJOURS les mêmes versions pour ces 2 packages.

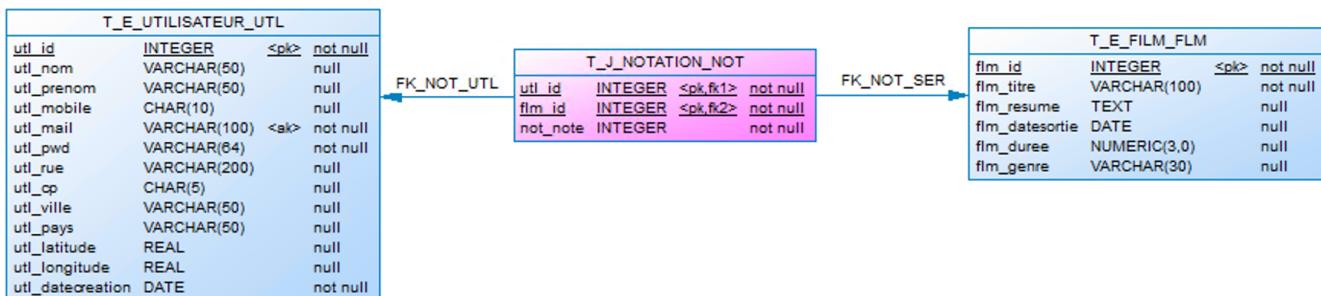
**Après test, les versions 9.0.1 de ces 2 packages fonctionnent également...**

#### 2.2. Création du modèle en Code First

Créer un dossier `Models`, puis le sous-dossier `EntityFramework`.

L'objectif est de créer **en mode Code First** le modèle de données suivant (en respectant les règles de nommage ISO/CEI 9075 : [https://www.sqlspot.com/sites/sqlspot.com/IMG/pdf/Norme\\_de\\_developpement\\_BD.pdf](https://www.sqlspot.com/sites/sqlspot.com/IMG/pdf/Norme_de_developpement_BD.pdf)), **ainsi que la classe de contexte**. Vous utiliserez les Data Annotations (plus simple) ou l'API Fluent. Vous devrez également créer les properties de navigation entre les entités.

Modèle de données :



Vous remarquerez que cet exemple est très proche de celui du TP3. Vous pourrez donc vous aider des classes `Model` générées dans le TP3.

Explications :

Vincent COUTURIER

<b>Film (t_e_film_film)</b>		
<i>Nom du champ</i>	<i>Nom de la property</i>	<i>Spécifications</i>
flm_id	FilmId	PK, int, numéro s'incrémentant automatiquement
flm_titre	Titre	string -> BD : varchar(100) not null. Index (non unique)
flm_resume	Resume	string -> BD : text
flm_datesortie	DateSortie	DateTime -> BD : date (et non timestamp (type par défaut))
flm_duree	Duree	decimal -> BD : numeric(3,0)
flm_genre	Genre	string -> BD : varchar(30)
+ Propriété de navigation vers Notation (la nommer NotesFilm).		

<b>Notation (t_j_notation_not)</b>		
<i>Nom du champ</i>	<i>Nom de la property</i>	<i>Spécifications</i>
utl_id	UtilisateurId	PK, FK, int
flm_id	FilmId	PK, FK, int
not_note	Note	int not null, doit être compris entre 0 et 5.
+ Propriétés de navigation vers Utilisateur et Film. Les nommer UtilisateurNotant et FilmNote (c'est-à-dire film noté)		

<b>Utilisateur (t_e_utilisateur_utl)</b>		
<i>Nom du champ</i>	<i>Nom de la property</i>	<i>Spécifications</i>
utl_id	UtilisateurId	PK, int, numéro s'incrémentant automatiquement
utl_nom	Nom	string? -> Varchar(50)
utl_prenom	Prenom	string? -> Varchar(50)
utl_mobile	Mobile	string? -> Char(10) [Column("utl_mobile", TypeName = "char(10)")] // Impossible de mettre char tout court => char(10) OU Utilisation de la méthode IsFixedLength()
utl_mail	Mail	string -> Varchar(100) not null. Doit être unique (clé unique)
utl_pwd	Pwd	string -> Varchar(64) not null
utl_rue	Rue	string? -> Varchar(200)
utl_cp	CodePostal	string? -> Char(5)
utl_ville	Ville	string? -> Varchar(50)
utl_pays	Pays	string? -> Varchar(50) Valeur par défaut : France
utl_latitude	Latitude	float? -> real
utl_longitude	Longitude	float? -> real
utl_datecreation	DateCreation	DateTime -> date not null (et non timestamp (type par défaut)) Valeur par défaut : now()
+ Propriété de navigation vers Notation (la nommer NotesUtilisateur)		

**Toujours écrire les noms des champs et des tables en minuscules dans le code first.**

float? ⇔ Nullable<float> : le champ peut être null dans la BD.

Pour indiquer le nom de la table : annotation [Table("T\_E\_...")] ou méthode toTable() de l'API Fluent.

PK :

- Si la property PK est bien nommée (Id ou NomTableId), EF génère automatiquement une PK s'incrémentant automatiquement (sinon, il faut ajouter les attributs [Key] et [DatabaseGenerated(DatabaseGeneratedOption.Identity)]). Au niveau du script SQL généré, cela donnera GENERATED BY DEFAULT AS IDENTITY, ce qui correspond à un Identity.

Remarque : Identity est identique à Serial mais est préférable car figurant dans la norme SQL.  
<https://stackoverflow.com/questions/55300370/postgresql-serial-vs-identity>

- On peut aussi créer une séquence explicitement :  
<https://learn.microsoft.com/fr-fr/ef/core/modeling/relational/sequences>
- PK sur 1 ou plusieurs champs : <https://learn.microsoft.com/fr-fr/ef/core/modeling/keys?tabs=data-annotations>

IsFixedLength :

<https://stackoverflow.com/questions/40782599/how-do-i-configure-fixed-length-columns-in-ef-core>

Valeur par défaut (attention, il y a 2 types de valeurs par défaut : valeur texte et instruction SQL) :  
<https://learn.microsoft.com/fr-fr/ef/core/modeling/relational/default-values>

Clé unique (index unique) :

<https://stackoverflow.com/questions/21573550/setting-unique-constraint-with-fluent-api>

<https://learn.microsoft.com/fr-fr/ef/core/modeling/relational/unique-constraints>

Contraintes sur les valeurs :

<https://learn.microsoft.com/fr-fr/dotnet/api/system.componentmodel.dataannotations.rangeattribute?view=net-8.0>

<https://learn.microsoft.com/fr-fr/dotnet/api/system.componentmodel.dataannotations.regularexpressionattribute?view=net-8.0>

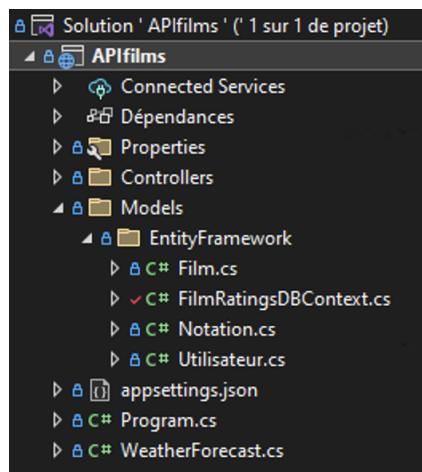
Clés étrangères : **vous respecterez les conventions de nommage** : FK\_TRIGRAMME1\_TRIGRAMME2 : fk\_not\_film, fk\_not\_utl. Le mode de suppression sur les FK sera RESTRICT :

<https://stackoverflow.com/questions/48819429/how-to-enforce-restrict-deletebehavior-for-migration-build-using-ef>

Dans la classe de contexte, ajouter la méthode `OnConfiguring` permettant de définir le serveur PostgreSQL et la base de données cibles.

Dans la méthode `OnModelCreating`, ajouter en 1<sup>ère</sup> ligne :

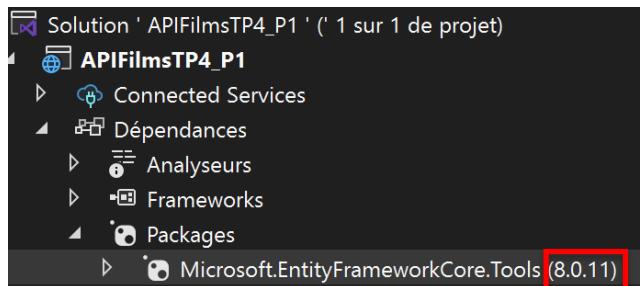
`modelBuilder.HasDefaultSchema("public");`



### 2.3. Génération de la base de données

Pour créer la base de données et corriger les erreurs du modèle, un nouvel outil est disponible dans EF Core pour gérer les évolutions du modèle : les migrations. Une migration est une liste d'instructions qui permet à l'ORM d'exécuter une série de requêtes SQL. L'avantage des migrations est de permettre à plusieurs développeurs d'avoir la même base de données et cela même lors de changements fréquents, les migrations sont exécutées et mettent à jour la base de données afin d'éviter les conflits.

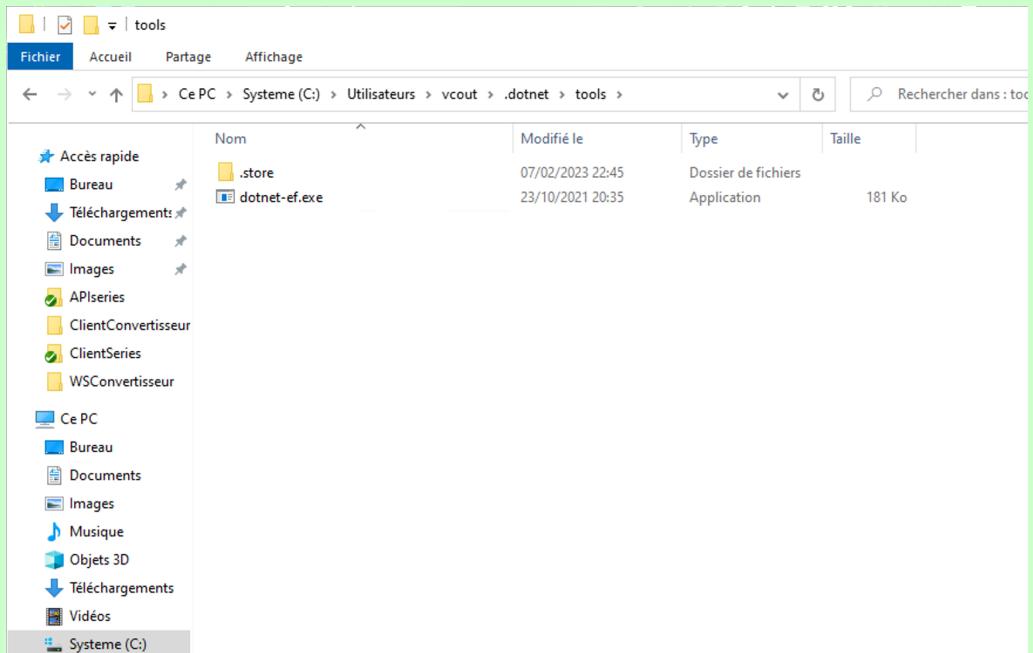
Pour activer les commandes liées aux migrations (disponible dans la CLI `Entity Framework Core-CLI .NET`), il faut installer le package `dotnet-ef` : `dotnet tool install --global dotnet-ef --version 8.0.x`. Il faut installer la même version que le package `Microsoft.EntityFrameworkCore.Tools`. Donc dans notre cas : `dotnet tool install --global dotnet-ef --version 8.0.11`



Saisir cette commande dans la console du gestionnaire de package (menu *Outils > Gestionnaire de package NuGet > Console du gestionnaire de package*) :

```
PM> dotnet tool install --global dotnet-ef --version 8.0.11
Vous pouvez appeler l'outil |à l'aide de la commande suivante|: dotnet-ef
L'outil 'dotnet-ef' (version 8.0.11) a |t été| correctement.
```

*Les outils dotnet seront installés ici :*



Pour voir la version des outils installés, vous pouvez lancer la commande `dotnet ef` dans une invite de commande à la racine du projet (i.e. là où se trouve le fichier `csproj`) :

```
C:\Users\vcout\source\repos\TP4P1\TP4P1>dotnet ef
              _/\_ \
             / \ \ \
            [E] [E] \
           / \ \ \
          /   \ \
Entity Framework Core .NET Command-line Tools 8.0.8
Usage: dotnet ef [options] [command]

Options:
  --version      Show version information
  -h|--help      Show help information
  -v|--verbose   Show verbose output.
  --no-color     Don't colorize output.
  --prefix-output Prefix output with level.

Commands:
  database    Commands to manage the database.
  dbcontext   Commands to manage DbContext types.
  migrations  Commands to manage migrations.

Use "dotnet ef [command] --help" for more information about a command.
```

*Si cela ne fonctionne pas, il faut ajouter le chemin vers l'exécutable `dotnet-ef`. Par exemple :*

```
P:\>C:\Users\vcout\.dotnet\tools\dotnet-ef.exe
  _____
 / \ \ \
 | . ) \\
 / \ \ \\
Entity Framework Core .NET Command-line Tools 8.0.8
```

*Sinon lancer la commande dotnet-ef (ou dotnet ef) dans la Console du gestionnaire de package :*

```
PM> dotnet ef
  _____
 / \ \ \
 | . ) \\
 / \ \ \\
Entity Framework Core .NET Command-line Tools 8.0.8

Usage: dotnet ef [options] [command]

Options:
  --version      Show version information
  -h|--help      Show help information
  -v|--verbose   Show verbose output.
  --no-color     Don't colorize output.
  --prefix-output Prefix output with level.

Commands:
  database      Commands to manage the database.
  dbcontext     Commands to manage DbContext types.
  migrations    Commands to manage migrations.

Use "dotnet ef [command] --help" for more information about a command.
PM>
```

*Pour désinstaller dotnet-ef : dotnet tool uninstall --global dotnet-ef*

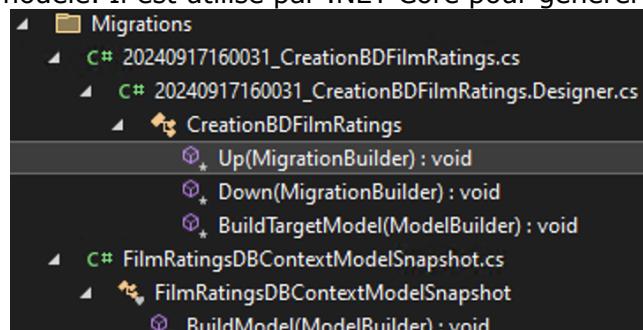
La 1<sup>ère</sup> étape est de créer une migration. Cela s'effectue en exécutant la commande suivante dans la *Console du gestionnaire de package* :

dotnet ef migrations add 'migration\_name' --project 'project\_name'  
Ou dotnet ef migrations add 'migration\_name' --project 'project\_name'.  
Saisir dotnet-ef migrations add CreationBDFilmRatings --project APIfilms  
APIfilms est le nom du projet.

**Avant de générer la migration penser à corriger les erreurs de code, sinon la génération échouera.**  
**Vous pouvez lancer le projet pour voir s'il fonctionne.**

**Si la commande précédente ne fonctionne pas dans le gestionnaire de package, l'exécuter dans une invite de commande Windows.**

Le framework va comparer l'état actuel du modèle avec la précédente migration (si elle existe) et va générer une classe (héritant de Microsoft.EntityFrameworkCore.Migrations.Migration) contenant deux méthodes : Up et Down. Ces deux méthodes contiennent les changements à appliquer pour passer d'une migration à une autre. Un fichier MonContextModelSnapshot est également généré (ou mis à jour s'il existe) et contient l'état courant du modèle. Il est utilisé par .NET Core pour générer les classes de migration.



En cas d'erreur, il est possible de supprimer la dernière migration (suppression du fichier de migration ET annulation de modification du fichier ModelSnapshot) avec la commande `dotnet-ef migrations remove --project APIfilms`. Attention, pour être supprimée, la migration ne doit pas avoir été appliquée dans la base de données (voir commande plus loin). Il est important de passer par cette commande plutôt que de supprimer manuellement le fichier pour éviter d'arriver à un état incohérent du fichier ModelSnapshot (ce qui peut être problématique pour les futures migrations, étant donné qu'elles se basent dessus).

Une fois la migration créée, il est nécessaire de l'appliquer sur la base de données pour créer effectivement les tables. Cette opération s'effectue avec la commande `dotnet-ef database update --project 'project_name'` qui appliquera l'ensemble des migrations (commande pour appliquer une migration spécifique : `dotnet-ef database update 'migration_name' --project 'project_name'`). Elle va appeler le code des méthodes `Up()` de chaque fichier de migration dans l'ordre chronologique. Dans le cas de la 1<sup>ère</sup> migration, une table `__EFMigrationsHistory` va être créée et contiendra chacune des migrations effectuées sur la base.

Saisir `dotnet-ef database update --project APIfilms`

*Remarque : si vous avez une erreur lors de l'update de la base, supprimer la migration. Corriger l'erreur et la recréer. Passer l'update sur la base de données.*

Pour annuler l'application d'une migration dans la base de données, il suffit d'appliquer la migration précédente `dotnet-ef database update 'migration_precedente' --project 'project_name'`. Dans ce cas, c'est la méthode `Down()` de la classe qui sera appelée et l'entrée correspondante de la table d'historique sera retirée.

Pour supprimer les applications de toutes les migrations et toutes les migrations :

```
dotnet ef database update 0 --project 'project_name'  
dotnet ef migrations remove --project 'project_name'
```

Il est possible de générer un script SQL avec la commande `dotnet-ef migrations script --project 'project_name'`, en spécifiant éventuellement l'étendue des migrations à appliquer avec les paramètres `-from` et `-to`. Cela peut être utile en cas de déploiement sur un nouveau serveur ou en production.

Saisir `dotnet-ef migrations script --project APIfilms`

```

CREATE TABLE IF NOT EXISTS "__EFMigrationsHistory" (
    "MigrationId" character varying(150) NOT NULL,
    "ProductVersion" character varying(32) NOT NULL,
    CONSTRAINT "PK__EFMigrationsHistory" PRIMARY KEY ("MigrationId")
);

START TRANSACTION;
CREATE TABLE t_e_film_flm (
    flm_id integer GENERATED BY DEFAULT AS IDENTITY,
    flm_titre character varying(100) NOT NULL,
    flm_resume text,
    flm_datesortie date,
    flm_duree numeric(3,0),
    flm_genre character varying(30),
    CONSTRAINT pk_ser PRIMARY KEY (flm_id)
);
CREATE TABLE t_e_utilisateur_utl (
    utl_id integer GENERATED BY DEFAULT AS IDENTITY,
    utl_nom character varying(50),
    utl_prenom character varying(50),
    utl_mobile character(10),
    utl_mail character varying(100) NOT NULL,
    utl_pwd character varying(64) NOT NULL,
    utl_rue character varying(200),
    utl_cp character(5),
    utl_ville character varying(50),
    utl_pays character varying(50) DEFAULT 'France',
    utl_latitude real,
    utl_longitude real,
    utl_datecreation date NOT NULL DEFAULT (now()),
    CONSTRAINT pk_utl PRIMARY KEY (utl_id)
);
CREATE TABLE t_j_notation_not (
    utl_id integer NOT NULL,
    flm_id integer NOT NULL,
    not_note integer NOT NULL,
    CONSTRAINT pk_not PRIMARY KEY (utl_id, flm_id),
    CONSTRAINT fk_not_film FOREIGN KEY (flm_id) REFERENCES t_e_film_flm (flm_id) ON DELETE RESTRICT,
    CONSTRAINT fk_not_utl FOREIGN KEY (utl_id) REFERENCES t_e_utilisateur_utl (utl_id) ON DELETE RESTRICT
);
CREATE INDEX ix_t_e_film_flm_titre ON t_e_film_flm (flm_titre);
CREATE UNIQUE INDEX uq_utl_mail ON t_e_utilisateur_utl (utl_mail);
CREATE INDEX "IX_t_j_notation_not_flm_id" ON t_j_notation_not (flm_id);
INSERT INTO "__EFMigrationsHistory" ("MigrationId", "ProductVersion")
VALUES ('20250207151011_CreationBDFilmRatings', '9.0.1');

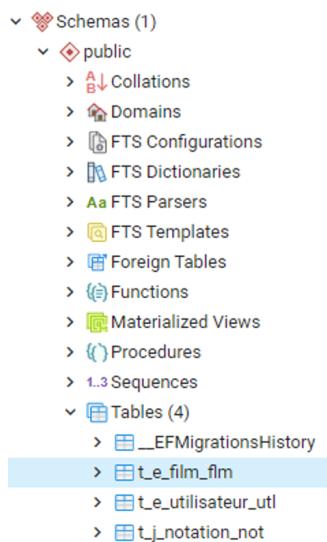
COMMIT;

```

**Normalement, vous devez obtenir le même script SQL, sinon, modifiez votre code.**

Toutes les commandes : <https://docs.microsoft.com/fr-fr/ef/core/miscellaneous/cli/dotnet>

Vérifier ensuite dans PgAdmin4 la création des tables :



On peut voir comment sont gérés les ID de Film et Utilisateur :

Constraint	Type	Value
Default	NONE	
Not NULL?	Yes	
Type	<input checked="" type="radio"/> IDENTITY	<input type="radio"/> GENERATED
Identity	BY DEFAULT	
Increment	1	
Start	1	
Minimum	1	
Maximum	2147483647	
Cache	1	
Cycled	No	

On peut voir le script (bouton droit de la souris sur le nom de la table puis *Scripts* puis *CREATE Script*) :

Query Query History

```

1 -- Table: public.t_e_film_film
2
3 -- DROP TABLE IF EXISTS public.t_e_film_film;
4
5 ✓ CREATE TABLE IF NOT EXISTS public.t_e_film_film
6 (
7     flm_id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
8     flm_titre character varying(100) COLLATE pg_catalog."default" NOT NULL,
9     flm_resume text COLLATE pg_catalog."default",
10    flm_datesortie date,
11    flm_duree numeric(3,0),
12    flm_genre character varying(30) COLLATE pg_catalog."default",
13    CONSTRAINT pk_ser PRIMARY KEY (flm_id)
14 )
15
16 TABLESPACE pg_default;
17
18 ✓ ALTER TABLE IF EXISTS public.t_e_film_film
19   OWNER to postgres;
20   -- Index: ix_t_e_film_film_titre
21
22   -- DROP INDEX IF EXISTS public.ix_t_e_film_film_titre;
23
24 ✓ CREATE INDEX IF NOT EXISTS ix_t_e_film_film_titre
25   ON public.t_e_film_film USING btree
26   (flm_titre COLLATE pg_catalog."default" ASC NULLS LAST)
27   TABLESPACE pg_default;

```

Exécuter le script `Insertion_Film_ratings_PostgreSQL.sql` qui devrait normalement fonctionner si votre base de données est conforme...

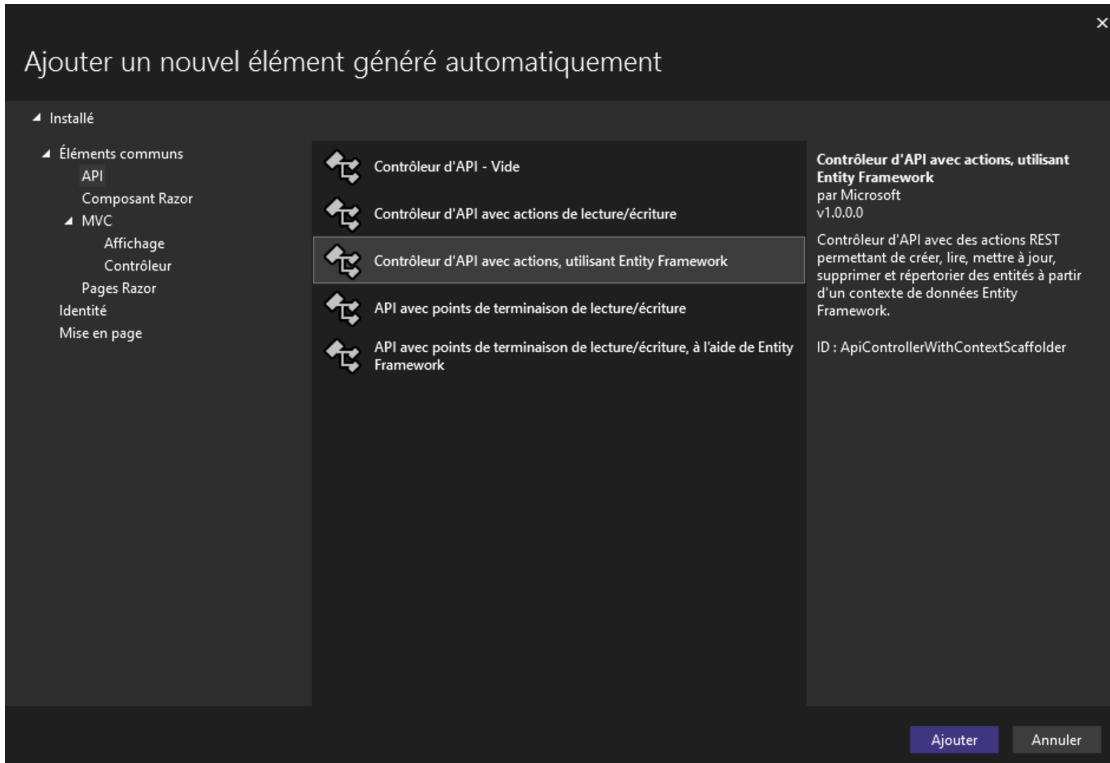
Notre couche *Model* est fonctionnelle. Cependant, elle n'est pas terminée. En effet, nous allons créer des classes partielles étendant les classes métier afin d'ajouter des contraintes non présentes dans la base de données.

En outre, l'architecture peut-être grandement améliorée. Nous verrons plus tard comment l'améliorer (création d'une classe de base contenant des propriétés techniques communes, etc.).

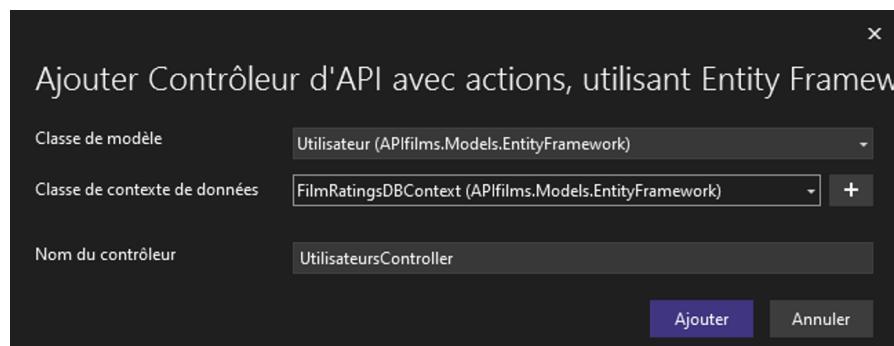
### **3. Couche Controller**

#### *3.1. Création de la couche Controller*

Dans le dossier *Controllers* de la solution, ajouter un nouveau contrôleur (*Ajouter > Contrôleur*).



Bien choisir « **Contrôleur d'API avec actions, utilisant Entity Framework** » car il s'agit d'un WS basé sur un modèle EF.



Valider l'ajout.

Le contrôleur créé s'appuie sur la classe de modèle et la classe de contexte de données que nous venons de générer.

*Pour créer le contrôleur en ligne de commande :*

```
dotnet aspnet-codegenerator controller -name MyController -async -api -m MyModel -dc MyDbContext -outDir Controllers
```

<https://learn.microsoft.com/fr-fr/aspnet/core/tutorials/first-web-api?view=aspnetcore-9.0>

*Si le scaffolding ne fonctionne pas, on peut bien sûr écrire le code à la main...*

### 3.2. Test de l'API

The screenshot shows the Swagger UI interface for the APIfilms v1 API. The 'Utilisateurs' endpoint is highlighted. Below it, several HTTP methods are listed with their respective URLs:

- GET /api/Utilisateurs
- POST /api/Utilisateurs
- GET /api/Utilisateurs/{id}
- PUT /api/Utilisateurs/{id}
- DELETE /api/Utilisateurs/{id}

Tester la méthode GET Utilisateurs avec Swagger. Vous obtiendrez l'erreur suivante.

**Code**    **Details**

**500**  
Undocumented    Error: response status is 500

Response body

```
System.InvalidOperationException: Unable to resolve service for type 'APISeries.Models.EntityFramework.SeriesDbContext' while attempting to activate 'APISeries.Controllers.UtilisateursController'.
   at Microsoft.Extensions.DependencyInjection.ActivatorUtilities.GetService(IServiceProvider sp, Type type, Type requiredBy, Boolean isDefaultParameterRequired)
   at lambda_method0(Closure , IServiceProvider , Object[])
   at Microsoft.AspNetCore.Mvc.Controllers.ControllerActivatorProvider.<>c__DisplayClass7_0.<CreateActivator>b__0(ControllerContext controllerContext)
   at Microsoft.AspNetCore.Mvc.Controllers.ControllerFactoryProvider.<>c__DisplayClass6_0.<CreateControllerFactory>b__0(ControllerContext controllerContext)
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope& scope, Object& state, Boolean& isCompleted)
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeInnerFilterAsync()
--- End of stack trace from previous location ---
   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeFilterPipelineAsync>g__Awaited|20_0(ResourceInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted)
   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Awaited|17_0(ResourceInvoker invoker, Task lastTask, IDisposable scope)
   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Awaited|17_0(ResourceInvoker invoker, Task task, IDisposable scope)
   at Microsoft.AspNetCore.Routing.EndpointMiddleware.<Invoke>g__AwaitRequestTask|6_0(Endpoint endpoint, Task requestTask, ILogger logger)
   at Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invoke(HttpContext context)
   at Swashbuckle.AspNetCore.SwaggerUI.SwaggerUIMiddleware.Invoke(HttpContext httpContext)
   at Swashbuckle.AspNetCore.Swagger.SwaggerMiddleware.Invoke(HttpContext httpContext, ISwaggerProvider swaggerProvider)
   at Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)
```

HEADERS

```
Accept: text/plain
Host: localhost:7076
User-Agent: Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36 Edg/105.0.1343.53
:method: GET
Accept-Encoding: gzip, deflate, br
Accept-Language: fr-fr;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
Referer: https://localhost:7076/swagger/index.html
Sec-GPC: 1
User-Agent: Microsoft Edge/105.0.1343.53
```

Download

Procéder comme en TP2 pour gérer l'injection de dépendances. Mettre la chaîne de connexion dans le fichier appsettings.json.

### 3.3. Modification de la couche Controller

Renommer le contrôleur GetUtilisateur en GetUtilisateurById.

Dans le contrôleur, ajouter la méthode public async Task<ActionResult<Utilisateur>> GetUtilisateurByEmail(string email) permettant de rechercher un utilisateur en fonction de son email (quelle que soit la casse).

Comment gérer plusieurs méthodes Get ? Il y a différents moyens, mais l'utilisation de la route est préférable.

Solution 1, en utilisant la route globale (au niveau de la classe) : <https://www.codeproject.com/Tips/1105755/ASP-NET-Core-Web-API-Multiple-Get-or-Post-Methods>

## Utilisateurs

<code>GET</code>	/api/Utilisateurs/GetUtilisateurs	▼
<code>GET</code>	/api/Utilisateurs/GetUtilisateurById/{id}	▼
<code>GET</code>	/api/Utilisateurs/GetUtilisateurByEmail/{email}	▼
<code>PUT</code>	/api/Utilisateurs/PutUtilisateur/{id}	▼
<code>POST</code>	/api/Utilisateurs/PostUtilisateur	▼

Le nom de l'action est celui de la méthode.

Solution 2, en complétant la route principale au niveau de chaque action :

<http://www.binaryintellect.net/articles/9db02aa1-c193-421e-94d0-926e440ed297.aspx>

Vous obtiendrez l'erreur suivante à l'exécution, même si les actions sont fonctionnelles :

```
C:\Users\Etudiant\source\repos\TP3\WSFilms\bin\Debug\netcoreapp3.0\WSFilms.exe
Info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
  Request starting HTTP/2 GET https://localhost:5001/swagger/v1/swagger.json
fail: Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware[1]
  An unhandled exception has occurred while executing the request.
System.NotSupportedException: HTTP method "GET" & path "api/Compte/{id}" overloaded by actions - WSFilms.Controllers.CompteController.GetCompteById (WSFilms), WSFilms.Controllers.CompteController.GetCompteByEmail (WSFilms). Actions require unique method/path combination for OpenAPI 3.0. Use ConflictingActionsResolver as a workaround
  at Swashbuckle.AspNetCore.SwaggerGen.SwaggerGenerator.GenerateOperations(IEnumerable`1 apiDescriptions, SchemaRepository schemaRepository)
  at Swashbuckle.AspNetCore.SwaggerGen.SwaggerGenerator.GeneratePaths(IEnumerable`1 apiDescriptions, SchemaRepository schemaRepository)
  at Swashbuckle.AspNetCore.SwaggerGen.SwaggerGenerator.GetSwagger(String documentName, String host, String basePath)
  at Swashbuckle.AspNetCore.Swagger.SwaggerMiddleware.Invoke(HttpContext httpContext, ISwaggerProvider swaggerProvider)
  at Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invoke(HttpContext context)
  at Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)
Info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
  Request finished in 305.8873000000004ms 500 text/plain
```

Swagger UI

localhost:5001/swagger/index.html

Select a definition: Film Rating API Documentation

Failed to load API definition.

Errors

Fetch error  
undefined /swagger/v1/swagger.json

Cette erreur d'ambiguïté des méthodes http est due aux annotations `[HttpGet("...")]`. Les remplacer par des `[HttpGet]`.

Vous pouvez également renommer les actions en utilisant l'annotation `[ActionName("...")]`.

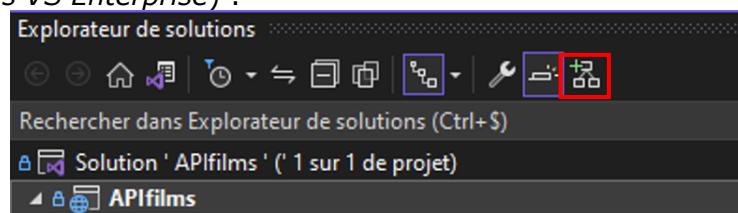
## Utilisateurs

GET	/api/Utilisateurs	▼
POST	/api/Utilisateurs	▼
GET	/api/Utilisateurs/GetById/{id}	▼
GET	/api/Utilisateurs/GetByEmail/{email}	▼
PUT	/api/Utilisateurs/{id}	▼
DELETE	/api/Utilisateurs/{id}	▼

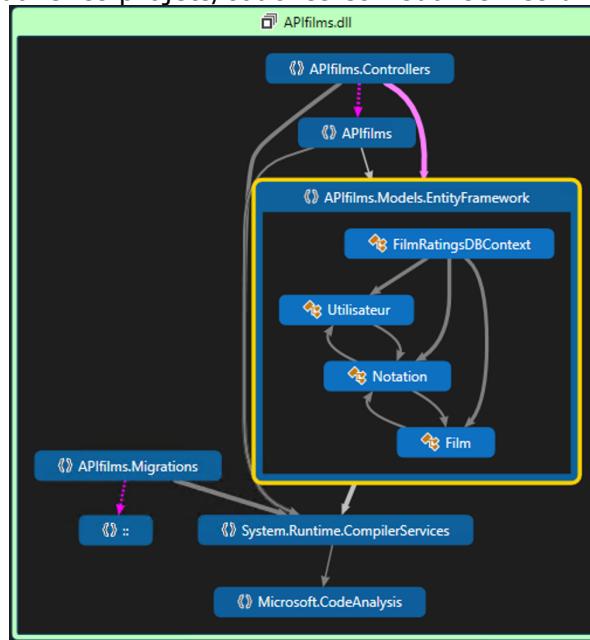
## Vous choisirez la solution que vous préférez.

Pour bien visualiser les liens entre les différents projets/couches de votre solution, vous pouvez afficher une carte de code et naviguer dans celle-ci (*la carte de code est disponible dans la version Entreprise de Visual Studio à condition d'avoir coché l'option lors de l'installation*).

Cliquer sur le projet principal. Cliquer ensuite sur le bouton de l'explorateur de solutions (cet outil est seulement disponible dans VS Enterprise) :



Vous pouvez ensuite naviguer dans les projets/couches et visualiser les différents liens.



Récupération de tous les utilisateurs :

The screenshot shows the Postman interface with a GET request to `https://localhost:7220/api/utilisateurs`. The response status is 200 OK, and the response body is a JSON array of two user objects:

```

1  [
2   {
3     "utilisateurId": 1,
4     "nom": "Calida",
5     "prenom": "Lilley",
6     "mobile": "0653938778",
7     "mail": "clilleymd@last.fm",
8     "pwd": "Toto12345678!",
9     "rue": "Impasse des bergeronnettes",
10    "codePostal": "74200",
11    "ville": "Allinges",
12    "pays": "France",
13    "latitude": 46.344795,
14    "longitude": 6.4886045,
15    "dateCreation": "2023-02-08T00:00:00",
16    "notesUtilisateur": []
17  },
18  {
19    "utilisateurId": 2,
20    "nom": "Gwendolin",
21    "prenom": "Dominguez",
22    "mobile": "0724970555",
23    "mail": "gdominguez0@washingtonpost.com",
24    "pwd": "Toto12345678!",
25    "rue": "Chemin de gom",
26    "codePostal": "73420"
}

```

On remarque que les notes ne sont pas par défaut récupérées (tableaux vides). Il serait contre-performant de les récupérer sur un Get (All).

Vous pouvez faire en sorte de les charger, si vous le souhaitez, lors de la récupération d'un seul utilisateur (chargement hâtif). Ce choix dépend du volume de données et de vos choix de développement :

- Si on décide dans la page affichant les informations du compte utilisateur de présenter les notes données, le chargement hâtif sera utilisé (Cf. TP précédent).
- Si cet affichage a lieu à la demande dans une page différente, il sera plus performant d'appeler le contrôleur `NotationsController` en passant en paramètre le n° de l'utilisateur. **Nous privilégierons cette solution.**

Recherche d'un utilisateur en utilisant l'email, par exemple [gdominguez0@washingtonpost.com](mailto:gdominguez0@washingtonpost.com) :

GET https://localhost:7220/api/utilisateurs/getbyemail/gdominguez0@washingtonpost.com

**Params** Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

**Query Params**

KEY	VALUE	DESCRIPTION	...	Bulk Edit

**Body** Cookies Headers (4) Test Results

200 OK 419 ms 469 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2   "utilisateurId": 2,
3   "nom": "Gwendolin",
4   "prenom": "Dominguez",
5   "mobile": "0724970555",
6   "mail": "gdominguez@washingtonpost.com",
7   "pwd": "Toto12345678!",
8   "rue": "Chemin de gom",
9   "codePostal": "73420",
10  "ville": "Voglans",
11  "pays": "France",
12  "latitude": 45.622154,
13  "longitude": 5.8853216,
14  "dateCreation": "2023-02-08T00:00:00",
15  "notesUtilisateur": []
16

```

Insérer un utilisateur :

```
{
  "nom": "DURAND",
  "prenom": "Marc",
  "mobile": null,
  "mail": "durand",
  "pwd": "info",
  "rue": "Chemin de Bellevue",
  "codePostal": "74940",
  "ville": "Annecy",
  "pays": "France",
  "latitude": null,
  "longitude": null,
}
```

Remarque : Il n'est pas utile de passer un ID car une séquence (IDENTITY) a été générée dans PostgreSQL.

POST https://localhost:7076/api/Utilisateurs

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Body (JSON)

```

1
2 ...
3   "nom": "DURAND",
4   "prenom": "Marc",
5   "mobile": null,
6   "mail": "durand",
7   "pwd": "info",
8   "rue": "Chemin de Bellevue",
9   "codePostal": "74940",
10  "ville": "Annecy",
11  "pays": "France",
12  "latitude": null,
13  "longitude": null
14
15
16

```

Body Cookies Headers (5) Test Results

Status: 201 Created Time: 2.25 s Size: 486 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2   "utilisateurId": 1004,
3   "nom": "DURAND",
4   "prenom": "Marc",
5   "mobile": null,
6   "mail": "durand",
7   "pwd": "info",
8   "rue": "Chemin de Bellevue",
9   "codePostal": "74940",
10  "ville": "Annecy",
11  "pays": "France",
12  "latitude": null,
13  "longitude": null,
14  "dateCreation": "2022-09-29T00:00:00",
15  "notesUtilisateur": []
16

```

On remarque que l'on vient d'insérer un utilisateur pour lequel le mail n'est pas valide. Aucune contrainte CHECK n'a été créée dans la base de données (en mode Code First), on peut donc saisir un mail non valide. Nous allons ajouter de nouvelles annotations dans la classe métier permettant d'ajouter d'autres contraintes.

Exemples d'annotations supplémentaires utiles : [Display], [Phone], [Url], [EmailAddress], [RegularExpression], etc.

En mode Code First, nous pouvons directement les ajouter dans le code des classes métier (pas besoin de créer des classes partielles).

Exemple :

```

[Required]
[Column("utl_mail")]
[EmailAddress]
[StringLength(100, MinimumLength = 6, ErrorMessage = "La longueur d'un email doit être comprise entre 6 et 100 caractères.")]
public string Mail { get; set; } = null!;

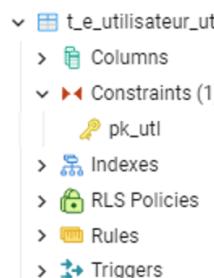
```

Créer une nouvelle migration et l'appliquer à la base de données.

Dotnet-ef migrations add NewAnnotations1 --project APIfilms

Dotnet-ef database update --project APIfilms

On remarque qu'aucune contrainte check n'a été ajoutée dans la base de données. Celle-ci ne sera gérée que par le modèle EF.



Insertion d'un utilisateur ayant un email non valide :

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <https://localhost:7076/api/Utilisateurs>
- Body:** JSON (selected)
- Request Body Content:**

```

1 {
2     "nom": "DURAND",
3     "prenom": "Marc",
4     "mobile": null,
5     "mail": "totò",
6     "pwd": "info",
7     "rue": "Chemin de Bellevue",
8     "codePostal": "74940",
9     "ville": "Annecy",
10    "pays": "France",
11    "latitude": null,
12    "longitude": null
13 }

```
- Response Status:** 400 Bad Request
- Response Headers:** Status: 400 Bad Request, Time: 410 ms, Size: 499 B
- Response Body (Pretty JSON):**

```

1 {
2     "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
3     "title": "One or more validation errors occurred.",
4     "status": 400,
5     "traceId": "00-3efe3d6b6caa629856f02cc7cd41253d-d355892a3ce21538-00",
6     "errors": [
7         "Mail": [
8             "The Mail field is not a valid e-mail address.",
9             "La longueur d'un email doit être comprise entre 6 et 100 caractères."
10        ]
11    }

```

Remarque :

- L'erreur de format de mail est renvoyée en anglais. On peut gérer sa propre erreur [EmailAddress(ErrorMessage = "...")]

### Rajouter les annotations que vous jugerez nécessaires sur les properties.

L'annotation [Phone] étant trop permissive, il est préférable d'utiliser une expression régulière : [RegularExpression(@"^0[0-9]{9}\$", ErrorMessage = "...")].

Un très bon site pour créer/tester vos expressions régulières : <https://regexr.com>

Exemples d'erreurs :

The screenshot shows the Postman application interface. At the top, the URL is `https://localhost:7220/api/utilisateurs`. The method is set to `POST`, and the body type is `JSON`. The body content is a JSON object representing a user:

```
1 {  
2     "nom": "DURAND",  
3     "prenom": "Marc",  
4     "mobile": "A",  
5     "mail": "D",  
6     "pwd": "E",  
7     "rue": "Chemin de Bellevue",  
8     "codePostal": "F",  
9     "ville": "Annecy",  
10    "pays": "France",  
11    "latitude": null,  
12    "longitude": null  
13 }
```

In the 'Body' tab, under the 'Pretty' tab, the response is shown as:

```
1 {  
2     "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  
3     "title": "One or more validation errors occurred.",  
4     "status": 400,  
5     "traceId": "00-820163c8f186ec1c5b47818e6f9a6dbc-480e0e7cb857fbee-00",  
6     "errors": {  
7         "Pwd": [  
8             "Le mot de passe doit contenir entre 12 et 20 caractères avec au moins 1 lettre majuscule, 1 chiffre et 1 caractère spécial"  
9         ],  
10        "Mail": [  
11            "The Mail field is not a valid e-mail address.",  
12            "La longueur d'un email doit être comprise entre 6 et 100 caractères."  
13        ],  
14        "Mobile": [  
15            "Le mobile doit contenir 10 chiffres"  
16        ],  
17        "CodePostal": [  
18            "Le code postal doit contenir 5 chiffres"  
19        ]  
20    }  
21 }
```

Ajouter des annotations [ProducesResponseType]. Exemples ici :

<https://learn.microsoft.com/fr-fr/aspnet/core/web-api/action-return-types?view=aspnetcore-8.0>

## 4. Tests unitaires

Ajouter un projet de tests unitaires.

Comme dans le TP2 :

- Installer le package NuGet Microsoft.EntityFrameworkCore **dans la même version que celle de l'API.**
  - L'appel du contrôleur nécessite de lui passer le contexte en paramètre. Créez le contexte dans le constructeur du test unitaire.

## **Coder les tests unitaires du WS (contrôleur).**

Il y a 2 façons de réaliser les tests :

- GET :
    - Réaliser 1 test qui réussit sur la méthode *GetAll*
      - Utilisez `context.Utilisateurs.ToList()` pour récupérer tous les utilisateurs.
    - Réaliser 1 test qui réussit et 1 qui échoue sur  *GetById*. Idem pour *GetByEmail*.

- Ex. pour récupérer l'utilisateur n°1 : context.Utilisateurs.Where(c => c.UtilisateurId == 1).FirstOrDefault()
- POST / PUT :
  - Pour tester un POST/PUT valide, il faut que le mail passé soit unique. Pour cela, il suffit, par exemple, de concaténer un timestamp ou un random au mail pour que le test réussisse à chaque fois. Random : <https://learn.microsoft.com/fr-fr/dotnet/api/system.random>

Code du POST :

```
[TestMethod]
public void Postutilisateur_ModelValidated_CreationOK()
{
    // Arrange
    Random rnd = new Random();
    int chiffre = rnd.Next(1, 1000000000);
    // Le mail doit être unique donc 2 possibilités :
    // 1. on s'arrange pour que le mail soit unique en concaténant un random ou un timestamp
    // 2. On supprime le user après l'avoir créé. Dans ce cas, nous avons besoin d'appeler la méthode DELETE de l'API
    // ou remove du DbSet.

    Utilisateur userAtester = new Utilisateur()
    {
        Nom = "MACHIN",
        Prenom = "Luc",
        Mobile = "0606070809",
        Mail = "machin" + chiffre + "@gmail.com",
        Pwd = "Toto1234!",
        Rue = "Chemin de Bellevue",
        CodePostal = "74940",
        Ville = "Annecy-le-Vieux",
        Pays = "France",
        Latitude = null,
        Longitude = null
    };

    // Act
    var result = controller.PostUtilisateur(userAtester).Result; // .Result pour appeler la méthode async de manière
    // synchrone, afin d'attendre l'ajout

    // Assert
    Utilisateur? userRecupere = context.Utilisateurs.Where(u => u.Mail.ToUpper() ==
    userAtester.Mail.ToUpper()).FirstOrDefault(); // On récupère l'utilisateur créé directement dans la BD grâce à son mail
    // unique
    // On ne connaît pas l'ID de l'utilisateur envoyé car numéro automatique.
    // Du coup, on récupère l'ID de celui récupéré et on compare ensuite les 2 users
    userAtester.UtilisateurId = userRecupere.UtilisateurId;
    Assert.AreEqual(userRecupere, userAtester, "Utilisateurs pas identiques");
}
}
```

**Rappel : n'utiliser .Result que dans les tests, et non dans l'application cliente sinon l'appel asynchrone échouera.**

- Comme déjà vu, il y a 2 types de données (modèle) non valides :
  - Données ne respectant pas les contraintes de la base de données (par exemple, attribut requis omis, clé unique violée, etc.). Dans ce cas, une erreur System.AggregateException est renvoyée. Par exemple :
 

```
SystemAggregateException: One or more errors occurred. (An error occurred while updating the entries. See the inner exception for details.) --> Microsoft.EntityFrameworkCore.DbUpdateException: An error occurred while updating the entries. See the inner exception for details. --> System.Data.SqlClient.SqlException: Impossible d'insérer la valeur NULL dans la colonne 'CPT_NOM', table 'FilmRatingsDB.dbo.T_E_COMPTE_CPT'. Cette colonne n'accepte pas les valeurs NULL. Échec de INSERT.
```

 Vous pourrez par exemple tester l'omission du nom ou le mail dupliqué. Pour tester avec MSTest une exception levée :
 <https://msdn.microsoft.com/fr-fr/library/microsoft.visualstudio.testtools.unittesting.expectedexceptionattribute.aspx>
  - Si les contraintes de la BD sont respectées, mais pas certaines annotations (expressions régulières, etc.), par exemple mobile="1", le modèle sera quand même considéré comme valide car la validation de l'état du modèle n'est déclenchée que

pendant l'exécution. Il incombe aux tests d'intégration de vérifier si la liaison du modèle fonctionne correctement.

Il est quand même possible de la tester en ajoutant un objet `ModelError` à `ModelState`.

<https://www.dotnettricks.com/learn/mvc/server-side-model-validation-in-mvc-razor>,  
<https://learn.microsoft.com/fr-fr/dotnet/api/microsoft.aspnetcore.mvc.modelbinding.modelstatedictionary.addmodelerror>

Exemple :

```
Utilisateur utilisateur = new Utilisateur()
{
    ...
    Mobile = "1",
    ...
};

string PhoneRegex = @"^0[0-9]{9}$";
Regex regex = new Regex(PhoneRegex);
if (!regex.IsMatch(utilisateur.Mobile))
{
    _controller.ModelState.AddModelError("Mobile", "Le n° de
mobile doit contenir 10 chiffres"); //On met le même message que dans la
classe Utilisateur.
}

...
```

Dans ce cas, le modèle n'est pas valide, on obtient un utilisateur null (en retour).

Malgré que le test passe, vous verrez qu'une ligne est bien insérée dans la base. En effet, il est nécessaire de vérifier dans le POST et le PUT que le modèle est valide avant d'insérer ou modifier :

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

Exemple du POST :

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<ActionResult<Utilisateur>> PostUtilisateur(Utilisateur
utilisateur)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    _context.Utilisateurs.Add(utilisateur);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetById", new { id =
utilisateur.UtilisateurId }, utilisateur);
}
```

- **DELETE :**

Cette fois il sera facile de réaliser les tests de la méthode `Delete` :

- Ajout d'un utilisateur en appelant la méthode `Add` sur le `DbSet` `Utilisateurs`. Committer l'ajout en appelant la méthode `context.SaveChanges()`
- Récupération de l'Id de l'utilisateur ajouté.
- Appel de la méthode `http Delete` en passant en paramètre l'Id de l'utilisateur créé.
- Vérification que l'utilisateur a été supprimé en le recherchant grâce à son mail ou son Id dans le `DbSet` `Utilisateurs`.

**Cette façon de réaliser des tests en utilisant le `DbSet` sous-jacent à l'API est plus simple, mais tout aussi rigoureuse.**

**En effet, le code généré par EF sur lequel s'appuient nos tests (le code lié aux `DbSet` y figure) est lui aussi en grande partie testé.**

Résultats de la couverture du code						
Hierarchy	Covered (Blocks)	Not Covered (Blocks)	Covered (Lines)	Partially Covered (Lines)	Not Covered (Lines)	
vcout_D101-S09 2023-02-08 22_32_06.coverz	468	560	269	0	874	
apifilmtests.dll	246	2	136	0	2	
apifilms.dll	222	558	133	0	872	
{ } APIfilms	0	34	0	0	26	
{ } APIfilms.Models.EntityFramework	146	46	97	0	49	
Film	0	18	0	0	19	
FilmRatingDbContext	19	4	48	0	6	
Notation	0	12	0	0	12	
Utilisateur	20	12	22	0	12	
FilmRatingDbContext.<>c	107	0	27	0	0	