TP N°5

ASP.NET Core WEB API

Enseignant : Malek Zribi

OBJECTIFS DU TP

- Créer des Endpoint pour gérer les opérations crud.
- Utiliser la technique de File upload dans une API WEB.
- Utiliser les classes DTO.
- Utiliser automapper pour le mapping des objets.
- Configuration de swagger.

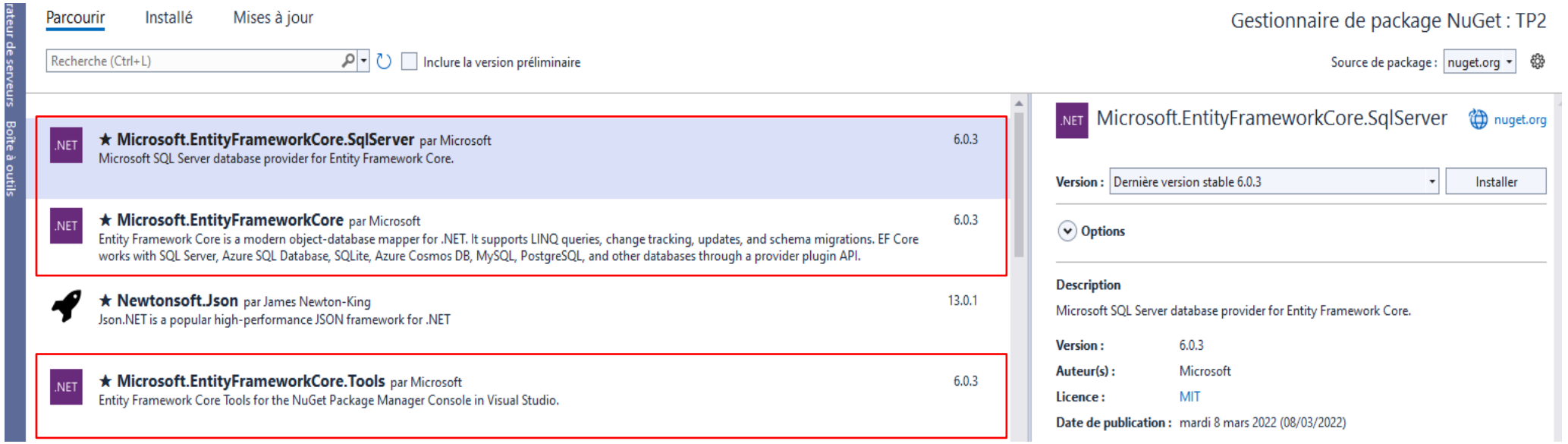
CRÉATION DES CLASSES DE DOMAINE

- Dans le cadre du développement d'une API WEB pour gérer des films avec leurs catégories, on vous donne les classes suivantes à créer dans un dossier Models :

```
public class Genre
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public byte Id { get; set; }
        [MaxLength(100)]
        public string Name { get; set; }
    }

public class Movie
    {
        public int Id { get; set; }
        [MaxLength(250)]
        public string Title { get; set; }
        public int Year { get; set; }
        public double Rate { get; set; }
        [MaxLength(2500)]
        public string Storeline { get; set; }
        public byte[] Poster { get; set; }
        public byte GenreId { get; set; }
        public Genre Genre { get; set; }
    }
```

INSTALLATION DES PACKAGES EF CORE

- Ouvrez Visual Studio 2022 et cliquez sur **Créer un nouveau projet** , de type **Application web ASP.Net Core WEB API**.
- Sous Visual Studio, Click droit sur le nom du projet → Gérer les packages Nuget
- Installer les packages NuGet suivants pour utiliser EF Core dans votre application :



CRÉATION DE LA CLASSE DE CONTEXTE

- Créer la classe de contexte :

```
public class ApplicationDbContext : DbContext
    {
 public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
        {
        }

        public DbSet<Genre> Genres { get; set; }
        public DbSet<Movie> Movies { get; set; }
    }
```

- La chaîne de connexion à la base de données à ajouter dans le fichier de configuration **appsettings.json** est la suivante :

```
"ConnectionStrings": {
    "DefaultConnection":"server=(localdb)\\MSSQLLocalDB;database=Movies;Trusted_Connection=true"
  }
```

INSCRIPTION DU SERVICE DE CONTEXTE ET CRÉATION DE LA BD

- Dans le fichier Program.cs, inscrire le service de contexte:

```
// Add services to the container.

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
```

- Sous le package manager console Lancer une migration Entity Framework et créer la base de données :

PM>Add-Migration DBCreate

PM>update-database

CRÉATION DES SERVICES REPOSITORY

- Ajouter un dossier **Services** et créer les interfaces IGenresService et IMoviesService avec les méthodes nécessaires :

```
public interface IGenresService
    {
        Task<IEnumerable<Genre>> GetAll();
        Task<Genre> GetById(byte id);
        Task<Genre> Add(Genre genre);
        Genre Update(Genre genre);
        Genre Delete(Genre genre);
        Task<bool> IsvalidGenre(byte id);
    }

public interface IMoviesService
    {
        Task<IEnumerable<Movie>> GetAll(byte genreId = 0);
        Task<Movie> GetById(int id);
        Task<Movie> Add(Movie movie);
        Movie Update(Movie movie);
        Movie Delete(Movie movie);
    }
```

CRÉATION DES CLASSES DE SERVICE

- Créer maintenant les classes de services GenresService et MoviesService :

```
public class GenresService : IGenresService
    {

        private readonly ApplicationDbContext _context;

        public GenresService(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<Genre> Add(Genre genre)
        {
            await _context.AddAsync(genre);
            _context.SaveChanges();

            return genre;
        }

        public Genre Delete(Genre genre)
        {
            _context.Remove(genre);
            _context.SaveChanges();

            return genre;
        }
```

```
public async Task<IEnumerable<Genre>> GetAll()
{
 return await _context.Genres.OrderBy(g =>
                                        g.Name).ToListAsync();
}

public async Task<Genre> GetById(byte id)
{
   return await _context.Genres.SingleOrDefaultAsync(g =>
                                                    g.Id == id);
}

      public Task<bool> IsvalidGenre(byte id)
       {
           return _context.Genres.AnyAsync(g => g.Id == id);
       }

       public Genre Update(Genre genre)
       {
           _context.Update(genre);
           _context.SaveChanges();

           return genre;
       }
   }
```

CRÉATION DES CLASSES DE SERVICE

```
using Microsoft.EntityFrameworkCore;

namespace MoviesApi.Services
{
    public class MoviesService : IMoviesService
    {
        private readonly ApplicationDbContext _context;

        public MoviesService(ApplicationDbContext context)
        {
            _context = context;
        }


        public async Task<Movie> Add(Movie movie)
        {
            await _context.AddAsync(movie);
            _context.SaveChanges();

            return movie;
        }

        public Movie Delete(Movie movie)
        {
            _context.Remove(movie);
            _context.SaveChanges();

            return movie;
        }
```

```
        public async Task<IEnumerable<Movie>> GetAll(byte genreId = 0)
        {
            return await _context.Movies
                .Where(m => m.GenreId == genreId || genreId == 0)
                .OrderByDescending(m => m.Rate)
                .Include(m => m.Genre)
                .ToListAsync();
        }

        public async Task<Movie> GetById(int id)
        {
         return await _context.Movies
                    .Include(m =>  m.Genre)
                    .SingleOrDefaultAsync(m => m.Id == id);
        }

        public Movie Update(Movie movie)
        {
            _context.Update(movie);
            _context.SaveChanges();

            return movie;
        }
    }
}
```

INSCRIPTION DES SERVICES REPOSITORY

- Dans le fichier Program.cs, inscrire les services repository.

```
// Add services to the container.

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();

builder.Services.AddTransient<IGenresService, GenresService>();
builder.Services.AddTransient<IMoviesService, MoviesService>();
```

LES CLASSES DTO

- Les classes DTO sont utilisés dans la couche de service pour renvoyer les données à la couche de présentation.
- Le plus grand avantage de l'utilisation des DTO est le découplage des applications clientes de vos structures de données internes.
- Masquer les propriétés particulières que les clients ne sont pas censés voir.
- Découpler votre couche de service de votre couche de base de données.

LES CLASSES DTO

- Créer un dossier nommé Dtos puis ajouter la classe GenreDto

- La propriété Id ne fait pas partie de cette classe, c'est un entier auto incrémentée dans la base de données, l'application cliente n'est pas censée fournir cette information.

```
namespace MoviesApi.Dtos
{
    public class GenreDto
    {
        [MaxLength(100)]
        public string Name { get; set; }
    }
}
```

GENRESCONTROLLER

- Créer un contrôleur API vide nommé GenresController

```
namespace MoviesApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class GenresController : ControllerBase
    {
        private readonly IGenresService _genresService;

        public GenresController(IGenresService genresService)
        {
            _genresService = genresService;
        }

        [HttpGet]
        public async Task<IActionResult> GetAllAsync()
        {
            var genres = await _genresService.GetAll();

            return Ok(genres);
        }
```

GENRESCONTROLLER

```
[HttpPost]
public async Task<IActionResult> CreateAsync(GenreDto dto)
{
    var genre = new Genre { Name = dto.Name };

    await _genresService.Add(genre);

    return Ok(genre);
}

[HttpPut("{id}")]
public async Task<IActionResult> UpdateAsync(byte id, [FromBody] GenreDto dto)
{
    var genre = await _genresService.GetById(id);

    if (genre == null)
        return NotFound($"No genre was found with ID: {id}");

    genre.Name = dto.Name;

    _genresService.Update(genre);

    return Ok(genre);
}
```

GENRESCONTROLLER

```
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteAsync(byte id)
{
    var genre = await _genresService.GetById(id);

    if (genre == null)
        return NotFound($"No genre was found with ID: {id}");

    _genresService.Delete(genre);

    return Ok(genre);
}
```

MOVIEDTO

- Créer la classe MovieDto qui va inclure les propriétés de la classe Movie sauf l'Id qui est auto incrémentée dans la base et la propriété de navigation Genre.

- La propriété Poster est un fichier image à envoyer à l'api donc doit être de type IFormFile.

```
namespace MoviesApi.Dtos
{
    public class MovieDto
    {
        [MaxLength(250)]

        public string Title { get; set; }

        public int Year { get; set; }

        public double Rate { get; set; }

        [MaxLength(2500)]
        public string Storeline { get; set; }

        public IFormFile? Poster { get; set; }

        public byte GenreId { get; set; }
    }
}
```

MOVIEDETAILSDTO

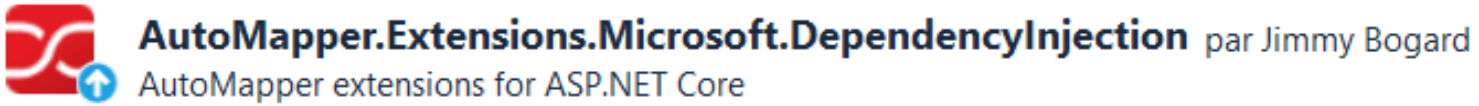
- Créer la classe MovieDetailsDto qui va servir comme modèle pour les méthodes GET, cette classe facilite l'affichage des informations par l'intégration de la propriété de navigation Genre dans le modèle et éviter d'avoir un objet Genre dans l'objet Movie.

```
namespace MoviesApi.Dtos
{
    public class MovieDetailsDto
    {
        public int Id { get; set; }

        public string Title { get; set; }

        public int Year { get; set; }

        public double Rate { get; set; }

        public string Storeline { get; set; }

        public byte[] Poster { get; set; }

        public byte GenreId { get; set; }

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

UTILISATION DE AUTOMAPPER

AutoMapper est un mappeur objet-objet, c'est-à-dire qu'il mappe un objet d'un type à un autre type.

- Pour l'utiliser il faut d'abord installer le package nuget suivant :



- Inscrire le service suivant de automapper

```
builder.Services.AddAutoMapper(typeof(Program));
```

UTILISATION DE AUTOMAPPER

- Créer un dossier nommé Helpers et ajouter à ce dossier une classe nommée MappingProfile qui sera utilisée par Automapper pour définir les opérations de mapping entre objets de l'application.

```
using AutoMapper;

namespace MoviesApi.Helpers
{
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            CreateMap<Movie, MovieDetailsDto>();
            CreateMap<MovieDto, Movie>()
                .ForMember(src => src.Poster, opt => opt.Ignore());
        }
    }
}
```

MOVIESCONTROLLER

- Créer un nouveau contrôleur API vide nommé MoviesController

```
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using MoviesApi.Services;

namespace MoviesApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MoviesController : ControllerBase
    {
        private readonly IMapper _mapper;
        private readonly IMoviesService _moviesService;
        private readonly IGenresService _genresService;

        private new List<string> _allowedExtenstions = new List<string> { ".jpg", ".png" };
        private long _maxAllowedPosterSize = 1048576;

        public MoviesController(IMoviesService moviesService, IGenresService genresService, IMapper mapper)
        {
            _moviesService = moviesService;
            _genresService = genresService;
            _mapper = mapper;
        }
```

MOVIESCONTROLLER

```
[HttpGet]
public async Task<IActionResult> GetAllAsync()
{
    var movies = await _moviesService.GetAll();

    // mapp movies List to MovieDetailsDto
    var data = _mapper.Map<IEnumerable<MovieDetailsDto>>(movies);

    return Ok(data);
}

[HttpGet("{id}")]
public async Task<IActionResult> GetByIdAsync(int id)
{
    var movie = await _moviesService.GetById(id);

    if(movie == null)
        return NotFound();

    var dto = _mapper.Map<MovieDetailsDto>(movie);

    return Ok(dto);
}
```

MOVIESCONTROLLER

```
[HttpGet("GetByGenreId")]
public async Task<IActionResult> GetByGenreIdAsync(byte genreId)
{
    var movies = await _moviesService.GetAll(genreId);
    var data = _mapper.Map<IEnumerable<MovieDetailsDto>>(movies);

    return Ok(data);
}
[HttpPost]
public async Task<IActionResult> CreateAsync([FromForm] MovieDto dto)
{
    if (dto.Poster == null)
        return BadRequest("Poster is required!");

    if (!_allowedExtenstions.Contains(Path.GetExtension(dto.Poster.FileName).ToLower()))
        return BadRequest("Only .png and .jpg images are allowed!");

    if(dto.Poster.Length > _maxAllowedPosterSize)
        return BadRequest("Max allowed size for poster is 1MB!");

    var isValidGenre = await _genresService.IsvalidGenre(dto.GenreId);

    if(!isValidGenre)
        return BadRequest("Invalid genere ID!");

    using var dataStream = new MemoryStream();

    await dto.Poster.CopyToAsync(dataStream);

    var movie = _mapper.Map<Movie>(dto);
    movie.Poster = dataStream.ToArray();

    _moviesService.Add(movie);

    return Ok(movie);
}
```

MOVIESCONTROLLER

```
[HttpPut("{id}")]
public async Task<IActionResult> UpdateAsync(int id, [FromForm] MovieDto dto)
{
    var movie = await _moviesService.GetById(id);

    if (movie == null)
        return NotFound($"No movie was found with ID {id}");

    var isValidGenre = await _genresService.IsvalidGenre(dto.GenreId);

    if (!isValidGenre)
        return BadRequest("Invalid genere ID!");

    if(dto.Poster != null)
    {
        if (!_allowedExtenstions.Contains(Path.GetExtension(dto.Poster.FileName).ToLower()))
            return BadRequest("Only .png and .jpg images are allowed!");

        if (dto.Poster.Length > _maxAllowedPosterSize)
            return BadRequest("Max allowed size for poster is 1MB!");

        using var dataStream = new MemoryStream();
        await dto.Poster.CopyToAsync(dataStream);

        movie.Poster = dataStream.ToArray();
    }

    movie.Title = dto.Title;
    movie.GenreId = dto.GenreId;
    movie.Year = dto.Year;
    movie.Storeline = dto.Storeline;
    movie.Rate = dto.Rate;

    _moviesService.Update(movie);

    return Ok(movie);
}
```

MOVIESCONTROLLER

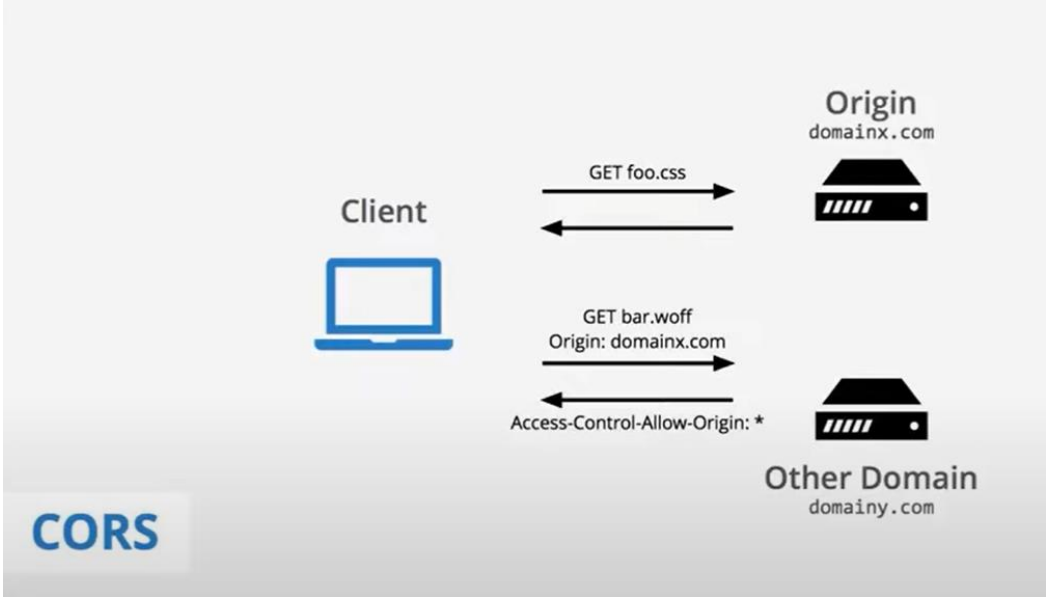
```
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteAsync(int id)
    {
        var movie = await _moviesService.GetById(id);

        if (movie == null)
            return NotFound($"No movie was found with ID {id}");

        _moviesService.Delete(movie);

        return Ok(movie);
    }
}
```

ENABLE CORS

- Pour débloquer les accès à l'API depuis une application cliente, il faut activer le service cors par l'inscription du service AddCors et l'ajout du middleware app.UseCors dans program.cs

```
builder.Services.AddCors();
```

- Ajouter le middleware app.UseCors




```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseCors(c => c.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin());

app.UseAuthorization();

app.MapControllers();

app.Run();
```

CONFIGURATION DE SWAGGER

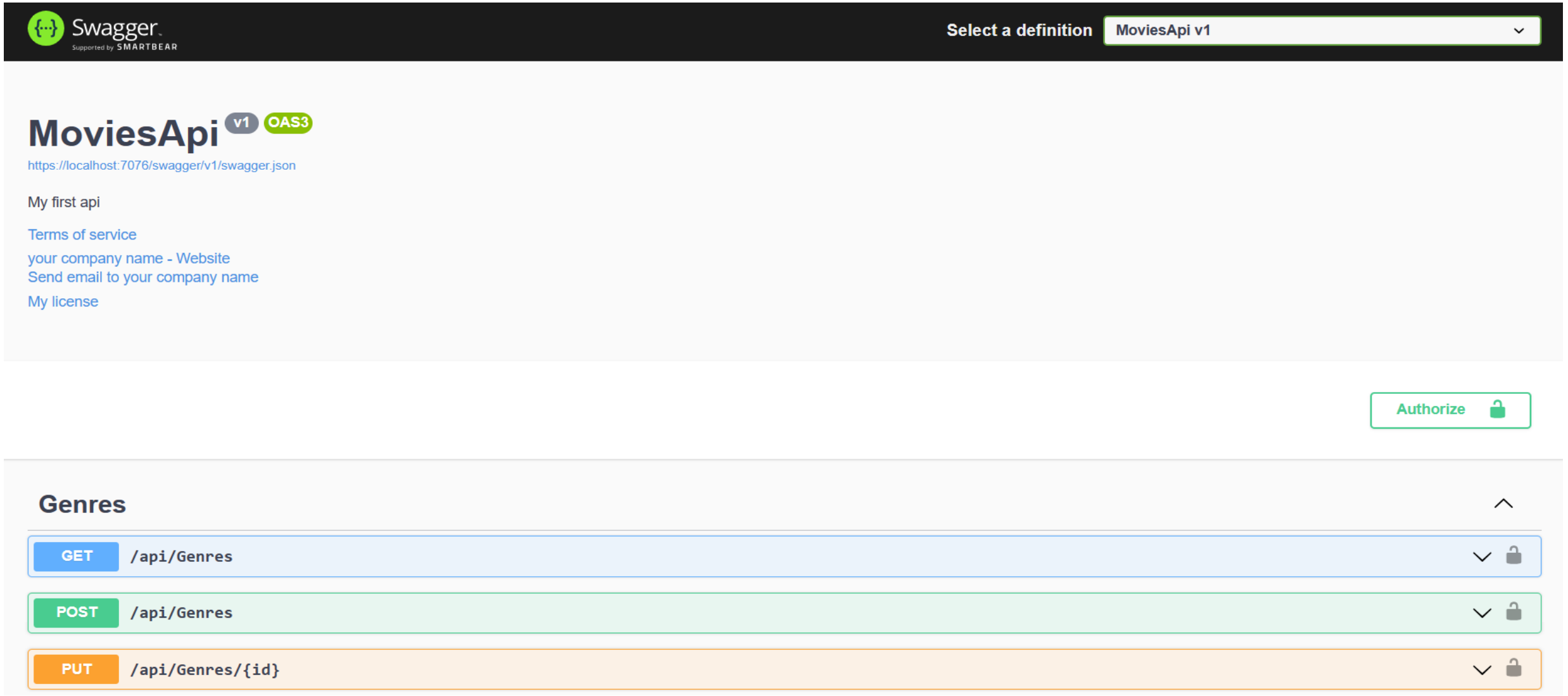
- Dans le fichier Program.cs, ajouter la configuration de swagger pour personnaliser la documentation de l'api:

```
builder.Services.AddSwaggerGen(options =>
{
    options.SwaggerDoc("v1", new OpenApiInfo
    {
        Version = "v1",
        Title = "MoviesApi",
        Description = "My first api",
        TermsOfService = new Uri("https://www.google.com"),
        Contact = new OpenApiContact
        {
            Name = "your company name",
            Email = "test@domain.com",
            Url = new Uri("https://www.google.com")
        },
        License = new OpenApiLicense
        {
            Name = "My license",
            Url = new Uri("https://www.google.com")
        }
    });
```

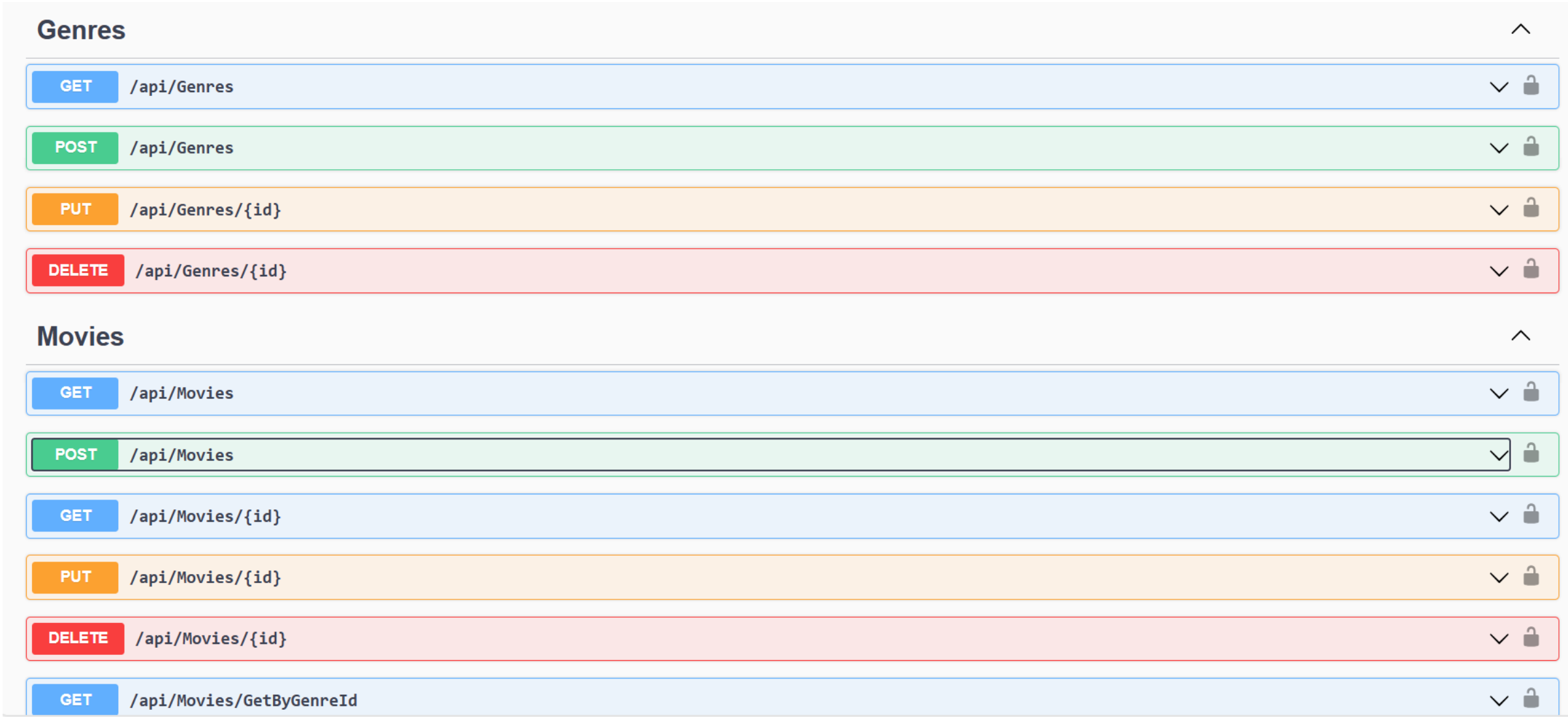
CONFIGURATION DE SWAGGER- AUTORISATION

```
options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
{
    Name = "Authorization",
    Type = SecuritySchemeType.ApiKey,
    Scheme = "Bearer",
    BearerFormat = "JWT",
    In = ParameterLocation.Header,
    Description = "JWT Authorization header using the Bearer scheme. \r\n\r\n Enter 'Bearer' [space] and then " +
    "your token in the text input below.\r\n\r\nExample: \"Bearer 12345abcdef\""
});
options.AddSecurityRequirement(new OpenApiSecurityRequirement
{
    {
        new OpenApiSecurityScheme
        {
            Reference = new OpenApiReference
            {
                Type = ReferenceType.SecurityScheme,
                Id = "Bearer"
            },
            Name = "Bearer",
            In = ParameterLocation.Header
        },
        new List<string>()
    }
});
```

EXÉCUTION SOUS SWAGGER



EXÉCUTION SOUS SWAGGER



EXÉCUTION SOUS SWAGGER

GET /api/Movies

Parameters

Cancel

No parameters

Execute | Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7076/api/Movies' \
  -H 'accept: */*'
```

Request URL

```
https://localhost:7076/api/Movies
```

Server response

Code	Details
200	Response body


```
[
  {
    "id": 1,
    "title": "qdfsdfsdf",
    "year": 2023,
    "rate": 9,
    "storeline": "ghfhfghfgh",
    "poster": "/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAUDBAQEAwUEBAQFBQUGBwwIBwcHBw8LCwkMEQ8SEhEPERETFhwXExQaFRERGCEYGh0dHx8fExciJCIeJBweHx7/2wBDAQUFBQcGBw4ICA4eFBEUHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh7/wAARCAQ4B4ADASIAAhEBAxEB/8QAHQAAAwEBAQEBAQEAAAAAAAAAAAECAwQFBgcICf/EAFYQAAICAQMDAgMHAgQDAgoAFwERAAIhAzFBE1FhBHEigZEFBhOhscHwMtEHQuHxCBQjFVIWFzVWYnJzkrPSNjd1g
```

EXÉCUTION SOUS SWAGGER