Asp.Net Core, Précis et concis

(Version .NET Core 5)

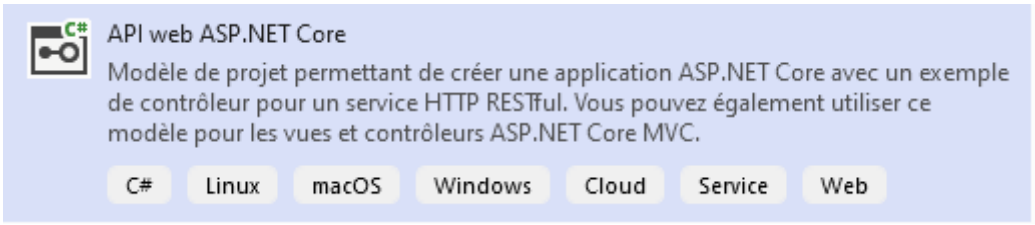
Table des matières

Création de Projet

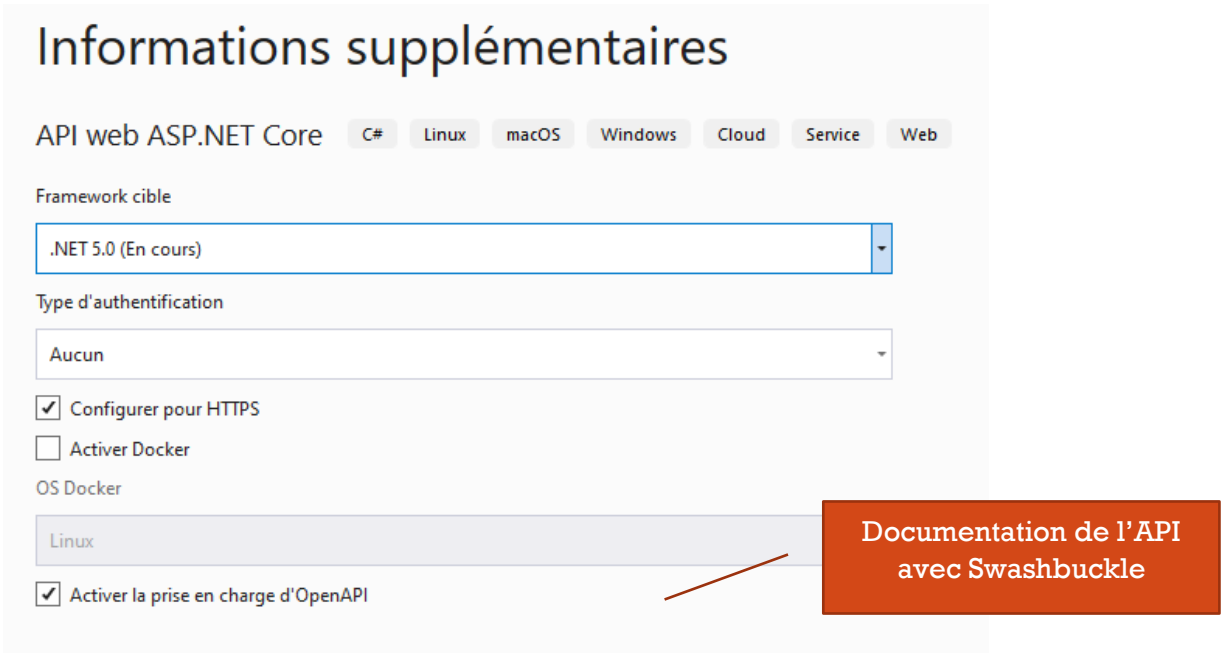
2 possibilités :

- Avec Visual Studio (solution la plus simple)
- Avec VS Code

Avec Visual Studio



Ne pas configurer l'authentification



Avec VS Code

Installer le SDK .NET Core de la version en cours (5.0 par exemple) pour l'os ciblé.

https://dotnet.microsoft.com/download/dotnet/5.0

Créer le dossier qui contiendra le code de l'api, puis depuis une invite de commande

```
dotnet new api
```

Ajout de packages au .csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
```

```
    <PackageReference Include="Swashbuckle.AspNetCore" Version="5.6.3" />
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="5.0.11" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.11" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="5.0.11">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>
    </PackageReference>
  </ItemGroup>

</Project>
```

On peut copier le code « PackageReference » depuis le site de packages nuget .
Exemple :

https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer

VS Code propose de restaurer sinon

```
dotnet restore
```

Entity Framework

Installation en global

```
dotnet tool install --global dotnet-ef
```

Puis pour ajouter une migration

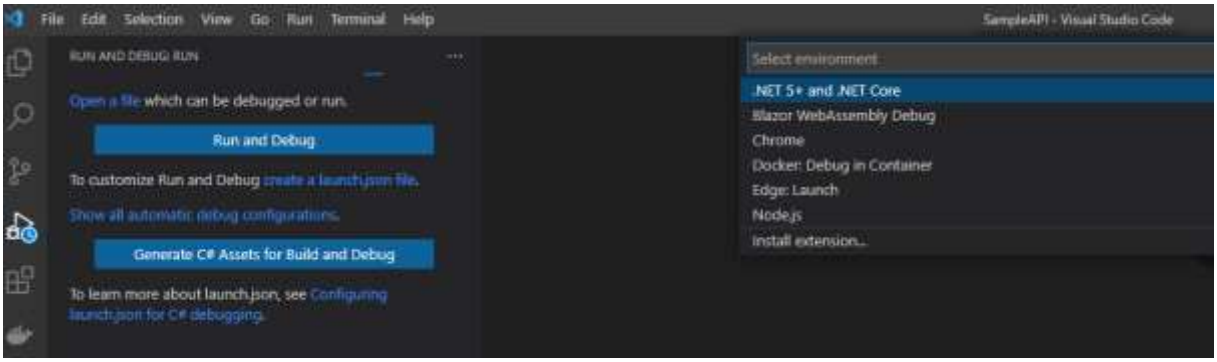
```
dotnet ef migrations add InitialCreate
```

Pour mettre à jour la base

```
dotnet ef database update
```

Pour débugger

« Run and Debug » … « .NET 5+ and .NET Core »

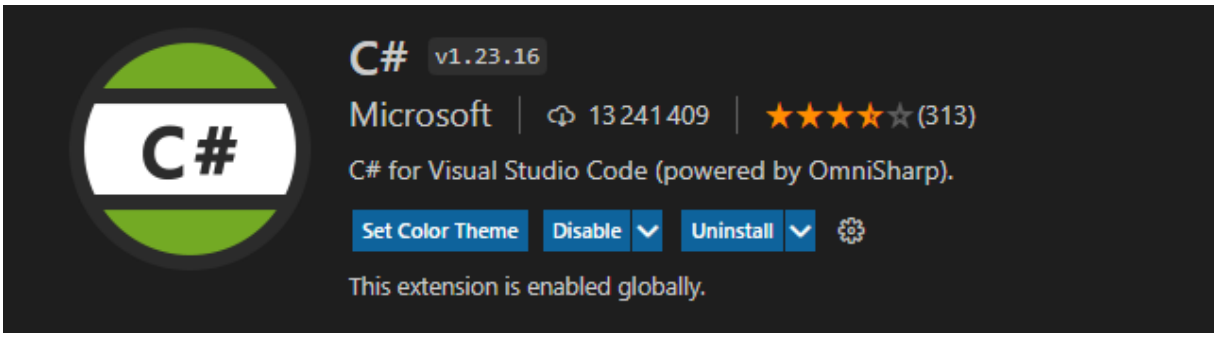


Puis lancer et ouvrir la page https://localhost:5001/swagger/index.html pour afficher la documentation de l'API

Extensions VS Code

Pour résoudre facilement les « using »

Debug, etc.



Prises de décisions

- ➔ Base de données ? **Entity Framework** ?
- ➔ Le projet aura-t-il besoin d'**authentification** ? Commencer par ajouter le model et les users au DbContext pour la migration initiale de la base.
- ➔ Controllers … **Versioning** ? le mieux est de penser assez tôt si l'API devra supporter le versioning

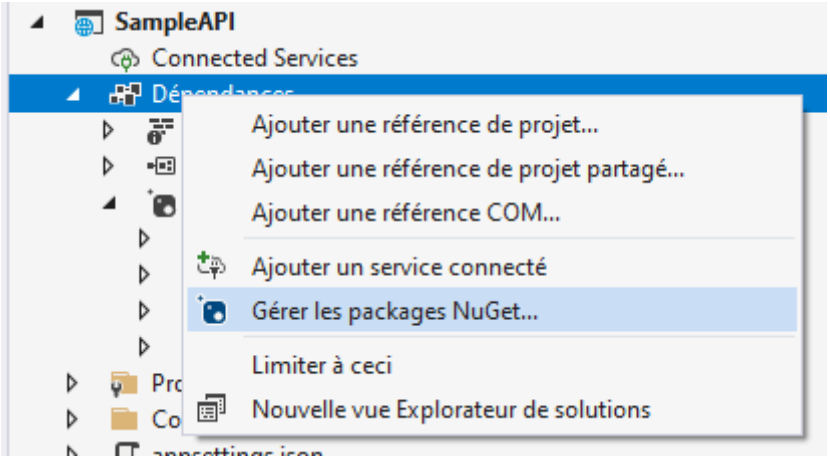
Entity Framework

Installation

Plusieurs possibilités :

- Packages NuGet
- En ligne de commande avec la console du gestionnaire de packages
- Ajout de PackageReferences au *csproj

Avec le **gestionnaire de packages Nuget**



Avec la **console du gestionnaire de packages**

```
Install-Package Microsoft.EntityFrameworkCore
```

Pour Sql Server

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

… et pour mettre à jour la base de données avec le CLI

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

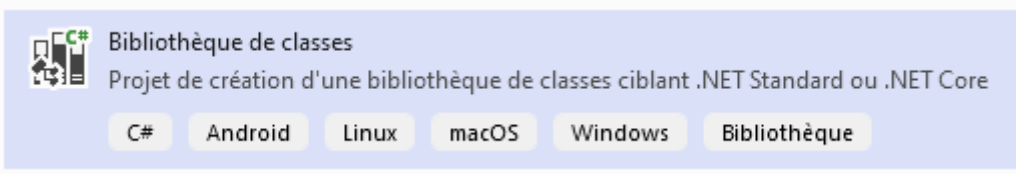
***.csproj**

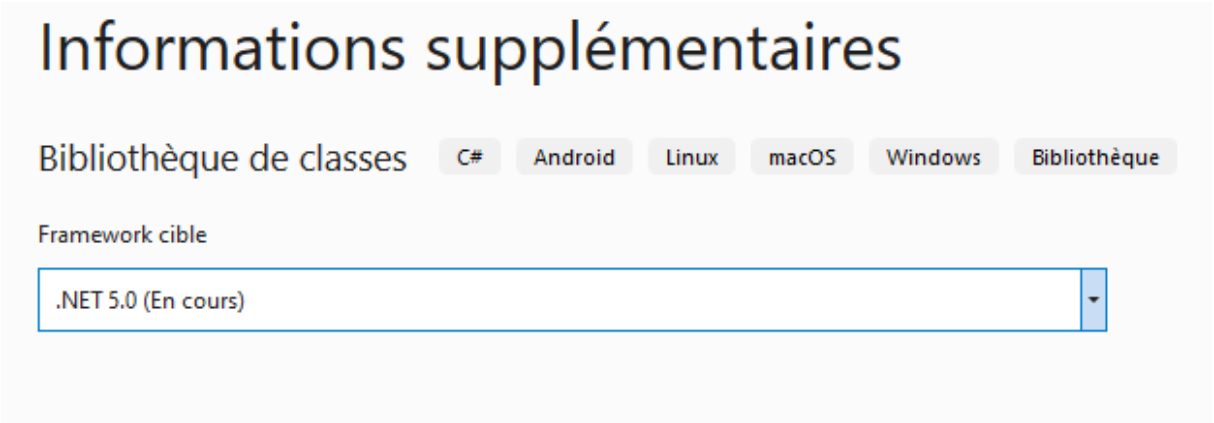
Ou ajout directement à l'ItemGroup du *.csproj (les dépendances sont installées à la sauvegarde du fichier)

```
 <PackageReference Include="Microsoft.EntityFrameworkCore" Version="5.0.11" />
<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.11" />
<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="5.0.11">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
</PackageReference>
```

Cas d'un projet Data

Création du projet « Data », choisir Bibliothèque de classes .NET Core





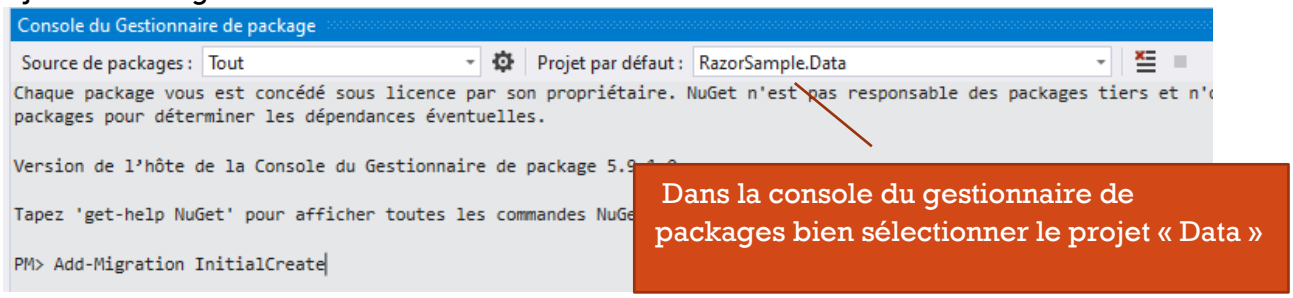
- Ajouter les package références au projet Data et au projet Principal

```
<PackageReference Include="Microsoft.EntityFrameworkCore" Version="5.0.11" />
<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.11" />
<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="5.0.11">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
</PackageReference>
```

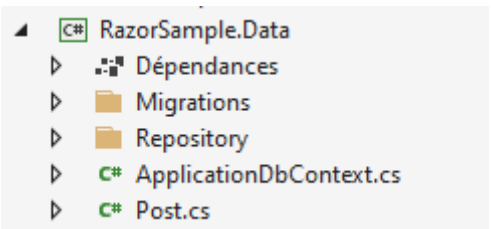
- Le projet principale doit ajouter référencer le projet « Data »
- Créer le DbContext et les entities
- Ajouter la chaine de connexion au projet principal et enregistrer le DbContext

```
services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

- Ajouter les migrations et modifier la base.



… puis « Add-Migration InitialCreate » et « Update-Database »



DbContext, entities

Création d'un dossier « **Data** » (ou d'un projet *.Data) qui contiendra le **DbContext , entities utilisées par le DbContext, repositories** et les migrations. Exemple de DbContext

```
using Microsoft.EntityFrameworkCore;

namespace SampleAPI.Data
{
   public class ApplicationDbContext : DbContext
   {
      public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
         : base(options)
      {

      }
      public DbSet<Post> Posts { get; set; }
      public DbSet<User> Users { get; set; }
   }
}
```

Et enregistrement du DbContext dans Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

   services.AddControllers();
   services.AddSwaggerGen(c =>
   {
      c.SwaggerDoc("v1", new OpenApiInfo { Title = "SampleAPI", Version = "v1" });
   });
}
```

Création d'un dossier « **Entities** »

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

namespace SampleAPI.Data
{
   public class User
   {
      [Key]
      public int Id { get; set; }
      public string Email { get; set; }
      public string Password { get; set; }
      public string Role { get; set; }
      [NotMapped]
      public string Token { get; set; }
   }
}
```

Exemple de model avec relation

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

namespace SampleAPI.Data
{
    public class Post
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Title { get; set; }
        [Required]
        public string Content { get; set; }

        [Required]
        public int UserId { get; set; }

        [ForeignKey("UserId")]
        public User User { get; set; }
    }
}
```

Chaine de connexion : ajout d'une **chaine de connextion** à « appsettings.json »

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=(LocalDb)\\MSSQLLocalDB;Initial Catalog=SampleDb;Integrated Security=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

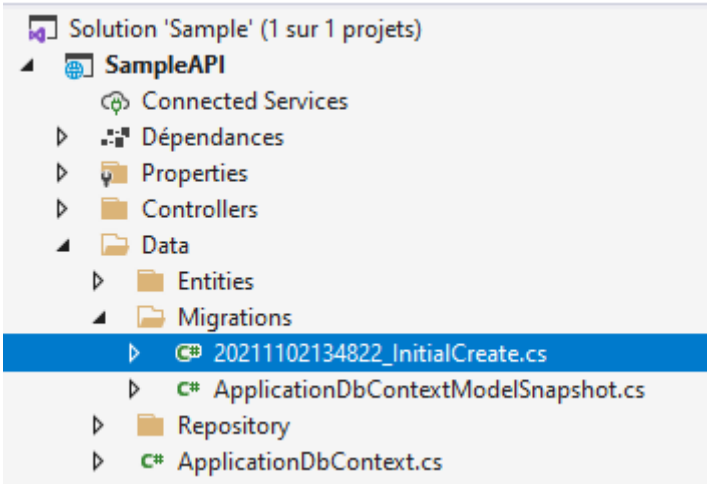
On a l'intelliSense dans le fichier « appsettings.json »

Création de migrations

Exemple création de migration initiale (depuis la console de gestionnaire de packages par exemple)

```
add-migration InitialCreate
```

On peut glisser le dossier « Migrations » dans le dossier « Data »



Mise à jour de la base de données

```
update-database
```

DbInitializer

Permet d'ajouter des données à la création de la base. Dans dossier/projet « Data »
Exemple

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using System;
using System.Linq;

namespace AspNetCoreMvcSample.Data.Initializer
{
   public interface IDbInitializer
   {
      void Initialize();
   }

   public class DbInitializer : IDbInitializer
   {
      private readonly ApplicationDbContext context;
      private readonly UserManager<IdentityUser> userManager;
      private readonly RoleManager<IdentityRole> roleManager;

      public DbInitializer(ApplicationDbContext context, UserManager<IdentityUser> userManager,
RoleManager<IdentityRole> roleManager)
      {
         this.context = context;
         this.roleManager = roleManager;
         this.userManager = userManager;
      }


      public void Initialize()
      {
         try
         {
            if (context.Database.GetPendingMigrations().Count() > 0)
            {
               context.Database.Migrate();
            }
         }
         catch (Exception ex)
         {}

         // create roles
         if (context.Roles.Count() > 0) return;
```

```
        roleManager.CreateAsync(new IdentityRole("Admin")).GetAwaiter().GetResult();
        roleManager.CreateAsync(new IdentityRole("Customer")).GetAwaiter().GetResult();


        // create admin

        userManager.CreateAsync(new ApplicationUser
        {
           UserName = "admin@mail.com",
           Email = "admin@mail.com",
           EmailConfirmed = true,
           FirstName = "Test",
           LastName = "Admin"
        }, "Admin2345_").GetAwaiter().GetResult();

        ApplicationUser user = context.ApplicationUsers.Where(u => u.Email == "admin@mail.com").FirstOrDefault();
        userManager.AddToRoleAsync(user, "Admin").GetAwaiter().GetResult();
     }
   }
}
```

Enregistrement dans Startup.cs

```
services.AddScoped<IDbInitializer, DbInitializer>();
```

Et exécution dans configure

On inject IDbInitializer dans la méthode « Configure »

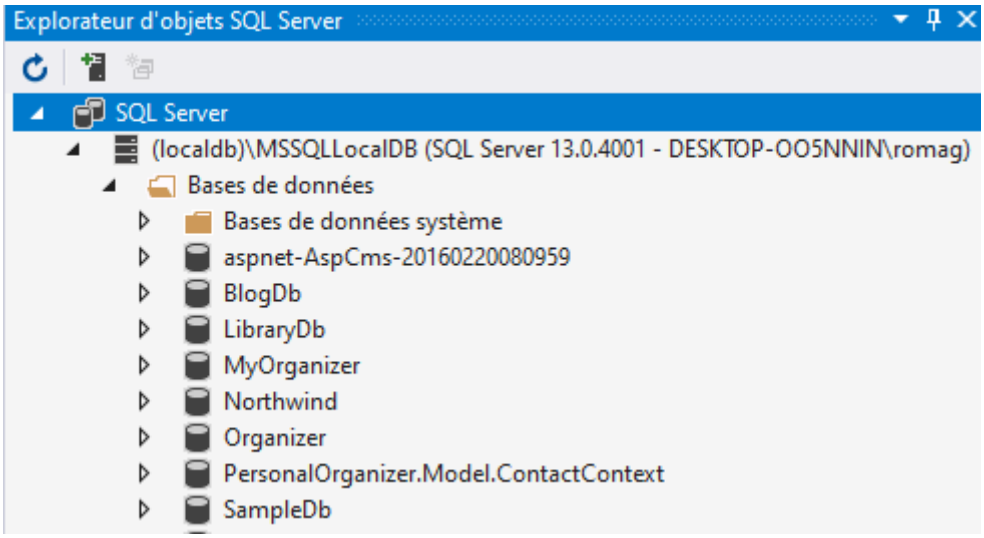
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
IDbInitializer initializer)
```

Et juste avant le routing on exécute

```
initializer.Initialize();
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
         pattern: "{controller=Home}/{action=Index}/{id?}");
     endpoints.MapRazorPages();
});
```

Pour manager les bases de données

- On peut télécharger SQL Server Management Studio (SSMS) (odbc driver)
- Avec l'explorateur d'objets SQL Server de Visual Studio



Démarrer LocalDb si besoin, depuis une invite de commande

```
sqllocaldb start
```

Repositories côté serveur

Création d'un dossier « Repository »

On injecte le DbContext dans les repositories

Note la méthode « Include » requiert

```
using System.Linq;
using Microsoft.EntityFrameworkCore;
```

Exemple de repository (avec relation)

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;

namespace SampleAPI.Data
{
   public interface IPostRepository
   {
      bool CreatePost(Post post);
      bool DeletePost(Post post);
      Post GetPost(int id);
      ICollection<Post> GetPosts();
      bool PostExists(int id);
      bool Save();
      bool UpdatePost(Post post);
   }

   public class PostRepository : IPostRepository
   {
      private readonly ApplicationDbContext context;
      public PostRepository(ApplicationDbContext context)
      {
         this.context = context;
      }

      public ICollection<Post> GetPosts()
```

```
    {
        return context.Posts.Include(x => x.User).ToList();
    }

    public Post GetPost(int id)
    {
        return context.Posts.Include(x => x.User).FirstOrDefault(x => x.Id == id);
    }

    public bool PostExists(int id)
    {
        return context.Posts.Any(x => x.Id == id);
    }

    public bool CreatePost(Post post)
    {
        context.Posts.Add(post);
        return Save();
    }

    public bool UpdatePost(Post post)
    {
        context.Posts.Update(post);
        return Save();
    }

    public bool DeletePost(Post post)
    {
        context.Posts.Remove(post);
        return Save();
    }

    public bool Save()
    {
        return context.SaveChanges() >= 0;
    }
  }
}
```

Enregistrement des repositories dans Startup.cs

```
services.AddScoped<IUserRepository, UserRepository>();
services.AddScoped<IPostRepository, PostRepository>();
```

Controllers

API Documentation (Swashbuckle)

Liens utiles

- https://swagger.io/docs/
- https://docs.microsoft.com/fr-fr/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-5.0&tabs=visual-studio

Si Swashbuckle n'est pas installé

```
Install-Package Swashbuckle.AspNetCore
```

Startup.cs ajout de la dépendance

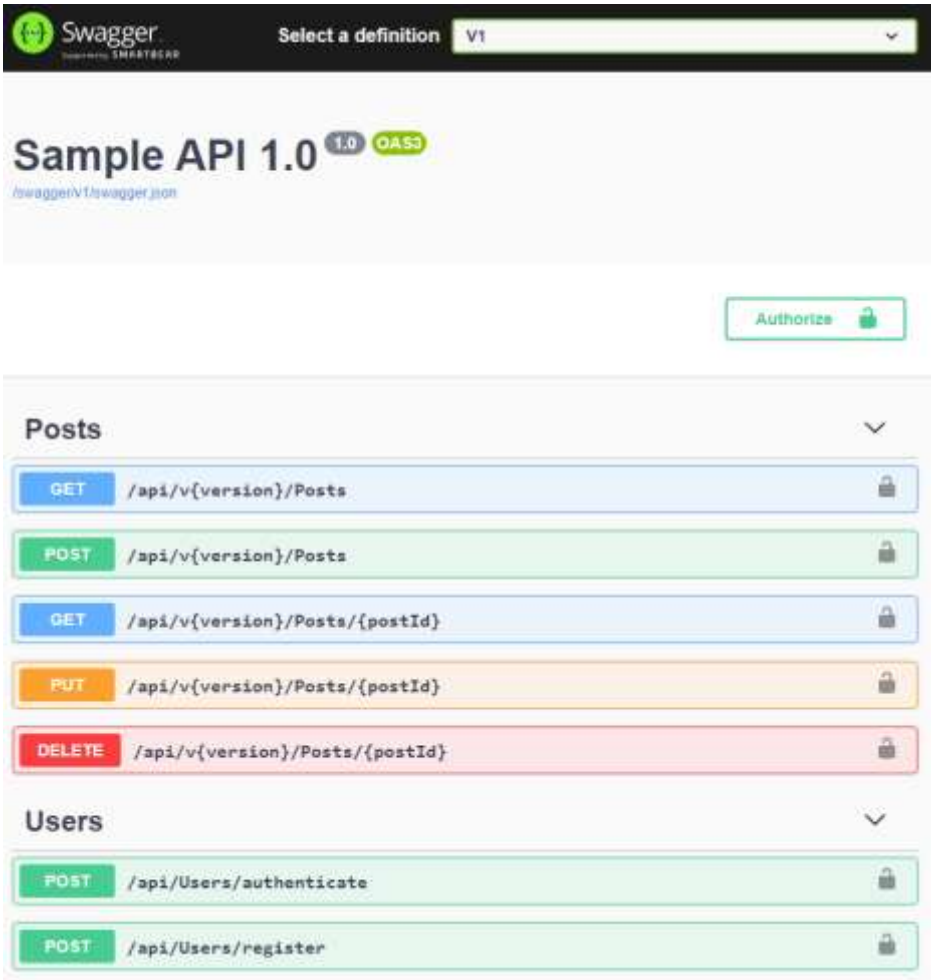
```
        services.AddSwaggerGen(c =>
        {
            c.SwaggerDoc("v1", new OpenApiInfo { Title = "SampleAPI", Version = "v1" });
        });
```

Et dans la méthode Configure

```
if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseSwagger();
            app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"SampleAPI v1"));
        }
```

On peut voir le fichier JSON en naviguant vers
« https://localhost:5001/swagger/v1/swagger.json » avec les différentes routes (le port dépend du profil)

Et la page « https://localhost:5001/swagger/index.html » permet d'afficher la documentation de l'API (le port dépend du profil)



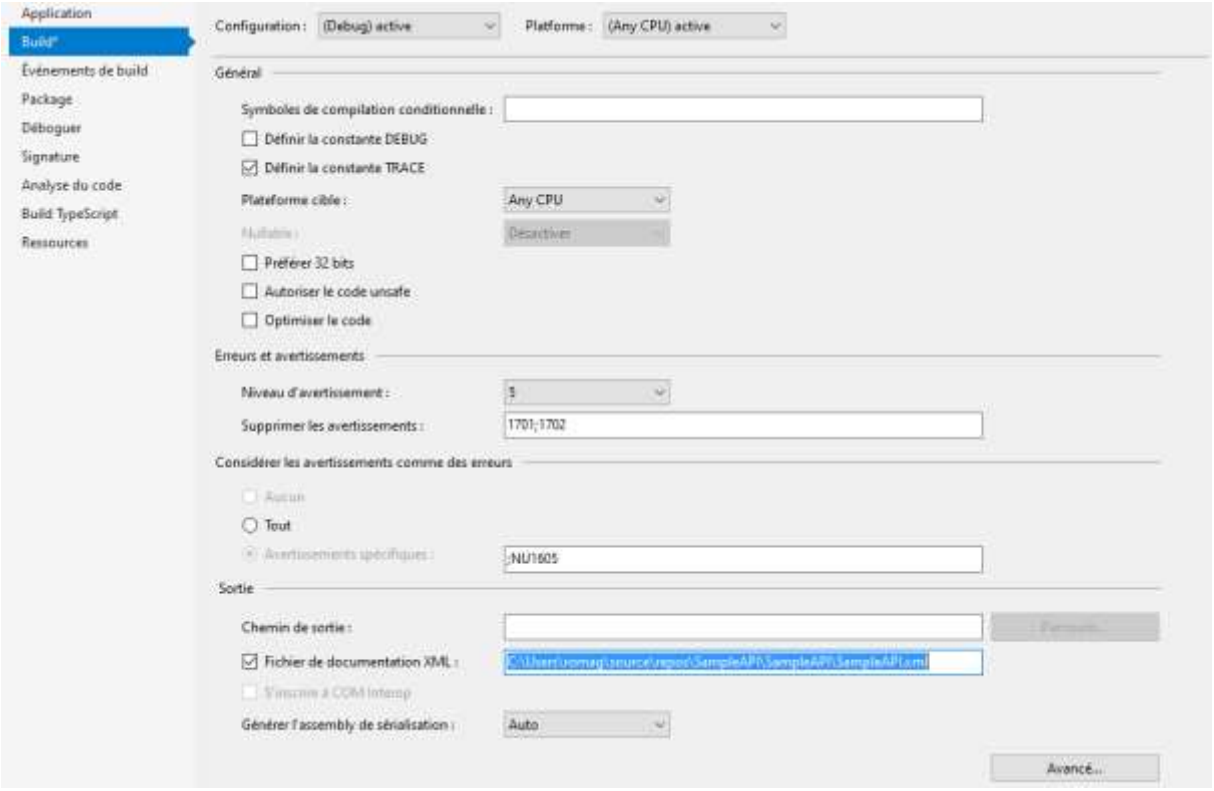
Les commentaires au dessus des méthodes du controleur sont affichés. Exemple

```
        /// <summary>
        /// Get the post list.
        /// </summary>
        /// <returns>The post list</returns>
        [HttpGet]
        [ProducesResponseType(200, Type = typeof(List<PostDto>))]
```

```
    public IActionResult GetPosts()
    {
        // retourne une list de dto
        var posts = repository.GetPosts();
        var postDtos = new List<PostDto>();
        foreach (var post in posts)
        {
            postDtos.Add(mapper.Map<PostDto>(post));
        }
        return Ok(postDtos);
    }
```

Les attributs « ProducesResponseType », « ProducesDefaultResponseType » sont utilisés également pour la page de documentation

Activer la documentation XML dans les paramètres du projet, onglet « Build »




```
services.AddSwaggerGen(options =>
{
    options.SwaggerDoc("v1", new OpenApiInfo { Title = "SampleAPI", Version = "v1"
});
    //
    var xmlCommentFile =
$"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
    var cmlCommentsFullPath = Path.Combine(AppContext.BaseDirectory,
xmlCommentFile);
    options.IncludeXmlComments(cmlCommentsFullPath); // ici
});
```

Versioning

Installation

```
Install-Package  Microsoft.AspNetCore.Mvc.Versioning
Install-Package  Microsoft.AspNetCore.Mvc.Versioning.ApiExplorer
```

Ou ajout des PackageReferences au csproj

```
<PackageReference Include="Microsoft.AspNetCore.Mvc.Versioning" Version="5.0.0" />
<PackageReference Include="Microsoft.AspNetCore.Mvc.Versioning.ApiExplorer" Version="5.0.0" />
```

Création d'une classe de configuration de swagger

```
using Microsoft.AspNetCore.Mvc.ApiExplorer;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Options;
using Microsoft.OpenApi.Models;
using Swashbuckle.AspNetCore.SwaggerGen;
using System;
using System.Collections.Generic;
using System.IO;
using System.Reflection;

namespace SampleAPI
{
   public class ConfigureSwaggerOptions : IConfigureOptions<SwaggerGenOptions>
   {
      readonly IApiVersionDescriptionProvider provider;

      public ConfigureSwaggerOptions(IApiVersionDescriptionProvider provider) => this.provider = provider;

      public void Configure(SwaggerGenOptions options)
      {
         foreach (var description in provider.ApiVersionDescriptions)
         {
            options.SwaggerDoc(
               description.GroupName, new OpenApiInfo()
               {
                  Title = $"Sample API {description.ApiVersion}",
                  Version = description.ApiVersion.ToString()
               });

         }

         options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
         {
            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\n" +
            "Example: \"Bearer 12345abcdef\"",
            Name = "Authorization",
            In = ParameterLocation.Header,
            Type = SecuritySchemeType.ApiKey,
            Scheme = "Bearer"
         });

         options.AddSecurityRequirement(new OpenApiSecurityRequirement()
            {
               {
                  new OpenApiSecurityScheme
                  {
                     Reference = new OpenApiReference
                     {
                        Type = ReferenceType.SecurityScheme,
                        Id = "Bearer"
                     },
                     Scheme = "oauth2",
                     Name = "Bearer",
                     In = ParameterLocation.Header,

                  },
```

Uniquement avec JWT

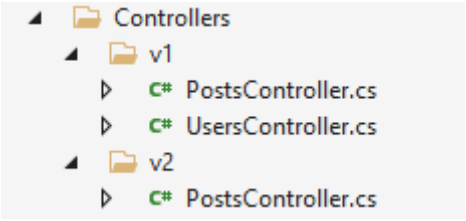
```
                new List<string>()
            }
        });

        //var xmlCommentFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
        //var cmlCommentsFullPath = Path.Combine(AppContext.BaseDirectory, xmlCommentFile);
        //options.IncludeXmlComments(cmlCommentsFullPath);
    }
  }
}
```

Enregistrement dans Startup.cs (configure services)

```
services.AddApiVersioning(options =>
{
   options.DefaultApiVersion = new ApiVersion(1, 0);
   options.ReportApiVersions = true;
   options.AssumeDefaultVersionWhenUnspecified = true;
});
services.AddVersionedApiExplorer(options => options.GroupNameFormat =
"'v'VVV");
services.AddTransient<IConfigureOptions<SwaggerGenOptions>,
ConfigureSwaggerOptions>();
services.AddSwaggerGen();
```

On peut désormais avoir des controllers avec plusieurs versions

```
   [Authorize]
   [ApiVersion("1.0")]
   [Route("api/v{version:apiVersion}/[controller]")]
   // [Route("api/[controller]")]
   [ApiController]
   //[ApiExplorerSettings(GroupName = "SampleOpenAPISpec")]
   public class PostsController : ControllerBase
```



Un controller « v2 » juste pour l'exemple

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

namespace SampleAPI.Controllers
{
   [Authorize]
   [ApiVersion("2.0")]
   [Route("api/v{version:apiVersion}/posts")]
   [ApiController]
   public class PostsV2Controller : ControllerBase
   {
```

```
        [HttpGet]
        [ProducesResponseType(200)]
        public IActionResult GetPosts()
        {
            return Ok("API V2");
        }
    }
}
```

Liens utiles

- https://codewithmukesh.com/blog/api-versioning-in-aspnet-core-3-1/
- https://github.com/dotnet/aspnet-api-versioning/wiki/API-Documentation#aspnet-core

Mapping avec AutoMapper

Va permettre de faire du mapping : les controllers reçoivent et retournent des models

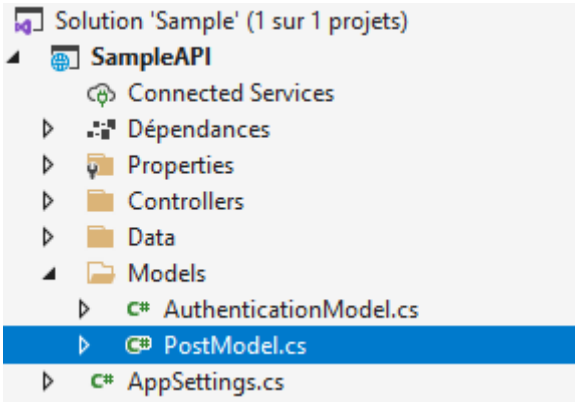
Installation

```
Install-Package AutoMapper
Install-Package AutoMapper.Extensions.Microsoft.DependencyInjection
```

Ou ajout des PackageReferences au csproj

```
<PackageReference Include="AutoMapper" Version="10.1.1" />
<PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="8.1.1" />
```

Création d'un dossier « Models » avec les models



Exemple de Model

```
using System.ComponentModel.DataAnnotations;

namespace SampleAPI.Models
{
    public class PostModel
    {
        [Key]
        public int Id { get; set; }
        [Required]
```

```
        public string Title { get; set; }
        [Required]
        public string Content { get; set; }
        [Required]
        public int UserId { get; set; }
    }
}
```

Création d'un **profile** dans le dossier « Data » permettant de configurer le mapping

```
using AutoMapper;
using SampleAPI.Models;

namespace SampleAPI.Data
{
    public class SampleAPIProfile : Profile
    {
        public SampleAPIProfile()
        {
            CreateMap<Post, PostModel>().ReverseMap();
        }
    }
}
```

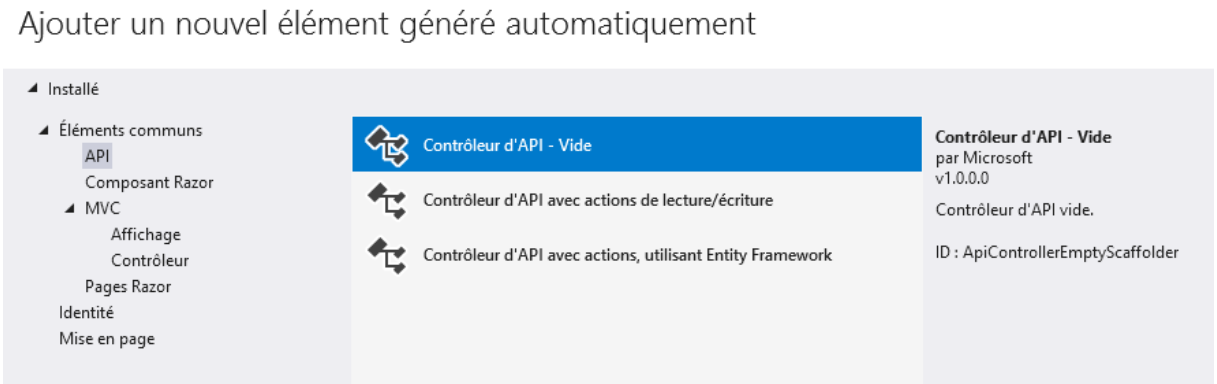
Enregistrement dans **Startup.cs** (grace aux extensions d'AutoMapper)

```
services.AddAutoMapper(typeof(SampleAPIProfile));
```

Dans les controllers on fait du mapping dans un sens ou dans l'autre selon les besoins

```
var post = mapper.Map<Post>(model); // model => entity
var result = mapper.Map<PostModel>(post); // entity => model
```

Création d'un controller API



Exemple de controller « PostController »

```
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using SampleAPI.Data;
using SampleAPI.Models;
using System;
using System.Collections.Generic;

namespace SampleAPI.Controllers
{
   [Authorize]
   [ApiVersion("1.0")]
   [Route("api/v{version:apiVersion}/[controller]")]
   // [Route("api/[controller]")]
   [ApiController]
   //[ApiExplorerSettings(GroupName = "SampleOpenAPISpec")]
   public class PostsController : ControllerBase
   {
      private readonly IPostRepository repository;
      private readonly IMapper mapper;

      public PostsController(IPostRepository postRepository, IMapper mapper)
      {
         this.repository = postRepository;
         this.mapper = mapper;
      }

      /// <summary>
      /// Get the post list.
      /// </summary>
      /// <returns>The post list</returns>
      [HttpGet]
      [ProducesResponseType(200, Type = typeof(List<PostModel>))]
      [ProducesResponseType(StatusCodes.Status500InternalServerError)]
      public IActionResult GetPosts()
      {
         try
         {
            var posts = repository.GetPosts();
            var results = new List<PostModel>();
            foreach (var post in posts)
            {
               results.Add(mapper.Map<PostModel>(post));
            }
            return Ok(results);
         }
         catch (Exception)
         {
            return this.StatusCode(StatusCodes.Status500InternalServerError, "Database Failure");
         }
      }

      [HttpGet("{postId:int}", Name = "GetPost")]
      [ProducesResponseType(200, Type = typeof(PostModel))]
      [ProducesResponseType(404)]
      [ProducesDefaultResponseType]
      public IActionResult GetPost(int postId)
      {
         try
         {
            var post = repository.GetPost(postId);
            if (post == null)
               return NotFound();

            var result = mapper.Map<PostModel>(post);
            return Ok(result);
         }
         catch (Exception)
```

```
    {
      return this.StatusCode(StatusCodes.Status500InternalServerError, "Database Failure");
    }
  }

  [HttpPost]
  [ProducesResponseType(StatusCodes.Status201Created, Type = typeof(PostModel))]
  [ProducesResponseType(StatusCodes.Status400BadRequest)]
  [ProducesResponseType(StatusCodes.Status500InternalServerError)]
  public IActionResult CreatePost([FromBody] PostModel model)
  {
    try
    {
      if (model == null)
        return BadRequest(ModelState);

      var post = mapper.Map<Post>(model);
      if (!repository.CreatePost(post))
      {
        ModelState.AddModelError("", $"Something went wrong when saving the record {post.Title}");
        return StatusCode(StatusCodes.Status500InternalServerError, ModelState);
      }
      return CreatedAtRoute("GetPost", new { postId = post.Id }, post);
    }
    catch (Exception)
    {
      return StatusCode(StatusCodes.Status500InternalServerError, "Database Failure");
    }
  }

  [HttpPut("{postId:int}", Name = "UpdatePost")]
  [ProducesResponseType(StatusCodes.Status200OK)]
  [ProducesResponseType(StatusCodes.Status400BadRequest)]
  [ProducesResponseType(StatusCodes.Status500InternalServerError)]
  public IActionResult UpdatePost(int postId, [FromBody] PostModel model)
  {
    try
    {
      if (model == null || postId != model.Id)
        return BadRequest(ModelState);

      var post = mapper.Map<Post>(model);
      if (!repository.UpdatePost(post))
      {
        ModelState.AddModelError("", $"Something went wrong when updating the record {post.Title}");
        return StatusCode(500, ModelState);
      }
      return Ok();
    }
    catch (Exception)
    {
      return this.StatusCode(StatusCodes.Status500InternalServerError, "Database Failure");
    }
  }

  [HttpDelete("{postId:int}", Name = "DeletePost")]
  [ProducesResponseType(StatusCodes.Status200OK)]
  [ProducesResponseType(StatusCodes.Status404NotFound)]
  [ProducesResponseType(StatusCodes.Status500InternalServerError)]
  public IActionResult DeletePost(int postId)
  {
    try
    {
      if (!repository.PostExists(postId))
        return NotFound();

      var post = repository.GetPost(postId);
      if (!repository.DeletePost(post))
      {
        ModelState.AddModelError("", $"Something went wrong when deleting the record {post.Title}");
        return StatusCode(StatusCodes.Status500InternalServerError, ModelState);
      }
      return Ok();
    }
```

```
        catch (Exception)
        {
            return this.StatusCode(StatusCodes.Status500InternalServerError, "Database Failure");
        }
    }
  }
}
```

Version Asynchrone

Il est possible de créer des controllers (retournent des « Task ») et des repositories en version async avec les méthodes async d'Entity Framework

Startup.cs

Les controllers sont automatiquement enregistrés avec

```
services.AddControllers();
```

Authentication JWT

Installation

```
Install-Package Microsoft.AspNetCore.Authentication.JwtBearer
```

Ou ajout de PackageReference dans le *.csproj

```
<PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="3.1.20" />
```

AppSettings

Ajout à « appsettings.json » d'une section « AppSettings » avec un « secret » utilisé pour la génération de tokens jwt.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=(LocalDb)\\MSSQLLocalDB;Initial
Catalog=SampleDb;Integrated Security=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "AppSettings": {
    "Secret": "My secret string"
  }
}
```

Création d'une classe **AppSettings** à la racine du projet

```
namespace SampleAPI
{
```

```
    public class AppSettings
    {
        public string Secret { get; set; }
    }
}
```

Enregistrement Startup.cs et configuration de JWT (configure services)

```
var appSettingsSection = Configuration.GetSection("AppSettings");
services.Configure<AppSettings>(appSettingsSection);
var appSettings = appSettingsSection.Get<AppSettings>();

var key = Encoding.ASCII.GetBytes(appSettings.Secret);
services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(x =>
{
    x.RequireHttpsMetadata = false;
    x.SaveToken = true;
    x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer = false,
        ValidateAudience = false
    };
});
```

Repository

```
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Text;

namespace SampleAPI.Data
{
    public interface IUserRepository
    {
        User Authenticate(string email, string password);
        bool IsUniqueUser(string email);
        User Register(string email, string password);
        int? ValidateToken(string token);
    }

    public class UserRepository : IUserRepository
    {
```

```
    private readonly ApplicationDbContext context;
    private readonly AppSettings appSettings;

    public UserRepository(ApplicationDbContext context, IOptions<AppSettings>
appsettings)
    {
        this.context = context;
        this.appSettings = appsettings.Value;
    }

    public User Authenticate(string email, string password)
    {
        var user = context.Users.SingleOrDefault(x => x.Email == email && x.Password
== password);
        if (user == null)
            return null;

        user.Token = GenerateJwtToken(user);
        user.Password = "";
        return user;
    }

    private string GenerateJwtToken(User user)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        var key = Encoding.ASCII.GetBytes(appSettings.Secret);
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new Claim[] {
                new Claim("id", user.Id.ToString()),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.Role, user.Role)
            }),
            Expires = DateTime.UtcNow.AddDays(7),
            SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }

    public int? ValidateToken(string token)
    {
        if (token == null)
            return null;

        var tokenHandler = new JwtSecurityTokenHandler();
        var key = Encoding.ASCII.GetBytes(appSettings.Secret);
        try
        {
            tokenHandler.ValidateToken(token, new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
```

```
                IssuerSigningKey = new SymmetricSecurityKey(key),
                ValidateIssuer = false,
                ValidateAudience = false,
                ClockSkew = TimeSpan.Zero
            }, out SecurityToken validatedToken);

            var jwtToken = (JwtSecurityToken)validatedToken;
            var userId = int.Parse(jwtToken.Claims.First(x => x.Type == "id").Value);

            return userId;
        }
        catch
        {
            return null;
        }
    }

    public bool IsUniqueUser(string email)
    {
        var user = context.Users.SingleOrDefault(x => x.Email == email);
        if (user == null)
            return true;

        return false;
    }

    public User Register(string email, string password)
    {
        var user = new User()
        {
            Email = email,
            Password = password,
            Role = "User" // Admin
        };

        context.Users.Add(user);
        context.SaveChanges();

        user.Password = "";
        return user;
    }
  }
}
```

Controller

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using SampleAPI.Data;
using SampleAPI.Models;
using System;

namespace SampleAPI.Controllers
```

```
{
   [Authorize]
   [Route("api/[controller]")]
   [ApiController]
   public class UsersController : ControllerBase
   {
      private readonly IUserRepository repository;
      public UsersController(IUserRepository userRepository)
      {
         repository = userRepository;
      }

      [AllowAnonymous]
      [HttpPost("authenticate")]
      public IActionResult Authenticate([FromBody] AuthenticationModel model)
      {
         try
         {
            var user = repository.Authenticate(model.Email, model.Password);
            if (user == null)
               return BadRequest(new { message = "Email or password is incorrect" });

            return Ok(user);
         }
         catch (Exception)
         {
            return StatusCode(StatusCodes.Status500InternalServerError, "Database
Failure");
         }
      }

      [AllowAnonymous]
      [HttpPost("register")]
      public IActionResult Register([FromBody] AuthenticationModel model)
      {
         try
         {
            bool ifUserNameUnique = repository.IsUniqueUser(model.Email);
            if (!ifUserNameUnique)
               return BadRequest(new { message = "User already exists" });

            var user = repository.Register(model.Email, model.Password);
            if (user == null)
               return BadRequest(new { message = "Error while registering" });
            return Ok();
         }
         catch (Exception)
         {
            return this.StatusCode(StatusCodes.Status500InternalServerError, "Database
Failure");
         }
      }
   }
}
```

Model

```
using System.ComponentModel.DataAnnotations;

namespace SampleAPI.Models
{
   public class AuthenticationModel
   {
      [Required]
      public string Email { get; set; }

      [Required]
      public string Password { get; set; }
   }
}
```

Entity

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

namespace SampleAPI.Data
{
   public class User
   {
      [Key]
      public int Id { get; set; }
      public string Email { get; set; }
      public string Password { get; set; }
      public string Role { get; set; }
      [NotMapped]
      public string Token { get; set; }
   }
}
```

Autorisation

Il suffit d'ajouter l'attribut « Authorize » :

- Au-dessus du controller pour l'appliquer à toutes les méthodes du controller, on peut toutefois utiliser l'attribut « AllowAnonymous » pour que certaines méthodes du controller ne le requièrent pas
- Au-dessus des méthodes désirées du controller

Il faut passer en header de requete http « Authorization : Bearer <token …> »

Autorisation selon le role

```
[Authorize(Roles = "Admin")] // 403 forbidden
```

Autorisation selon une policy que l'on ajoute

Cors

Startup.cs : dans configure services

```
services.AddCors();
```

Dans configure

```
app.UseCors(config => config
   .AllowAnyOrigin()
   .AllowAnyMethod()
   .AllowAnyHeader());
```

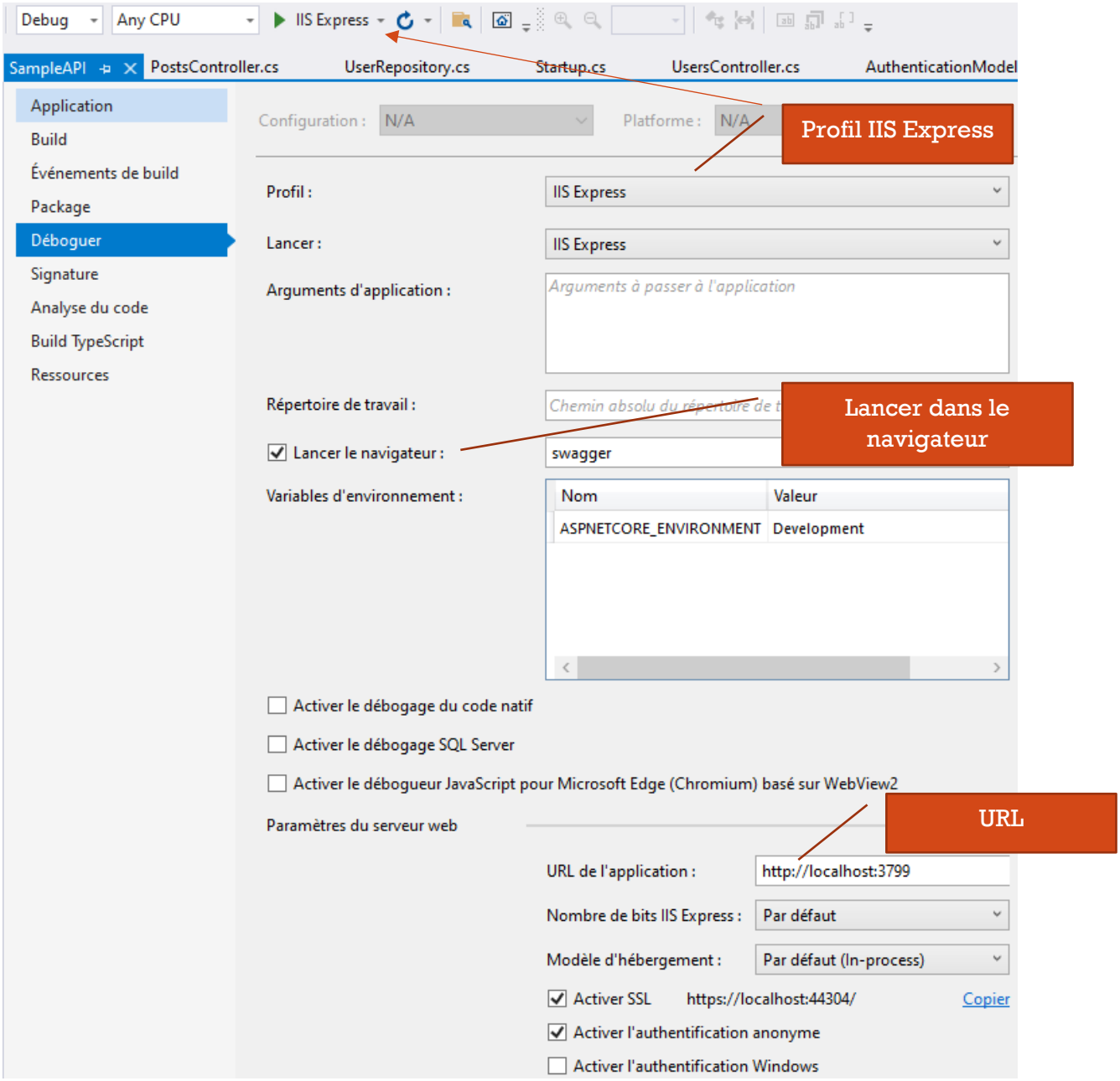
Ou si on veut spécifier l'origine par exemple

```
app.UseCors(config => config
      .AllowAnyOrigin()
      .AllowAnyMethod()
      .WithOrigins("http://sample.com"));
```

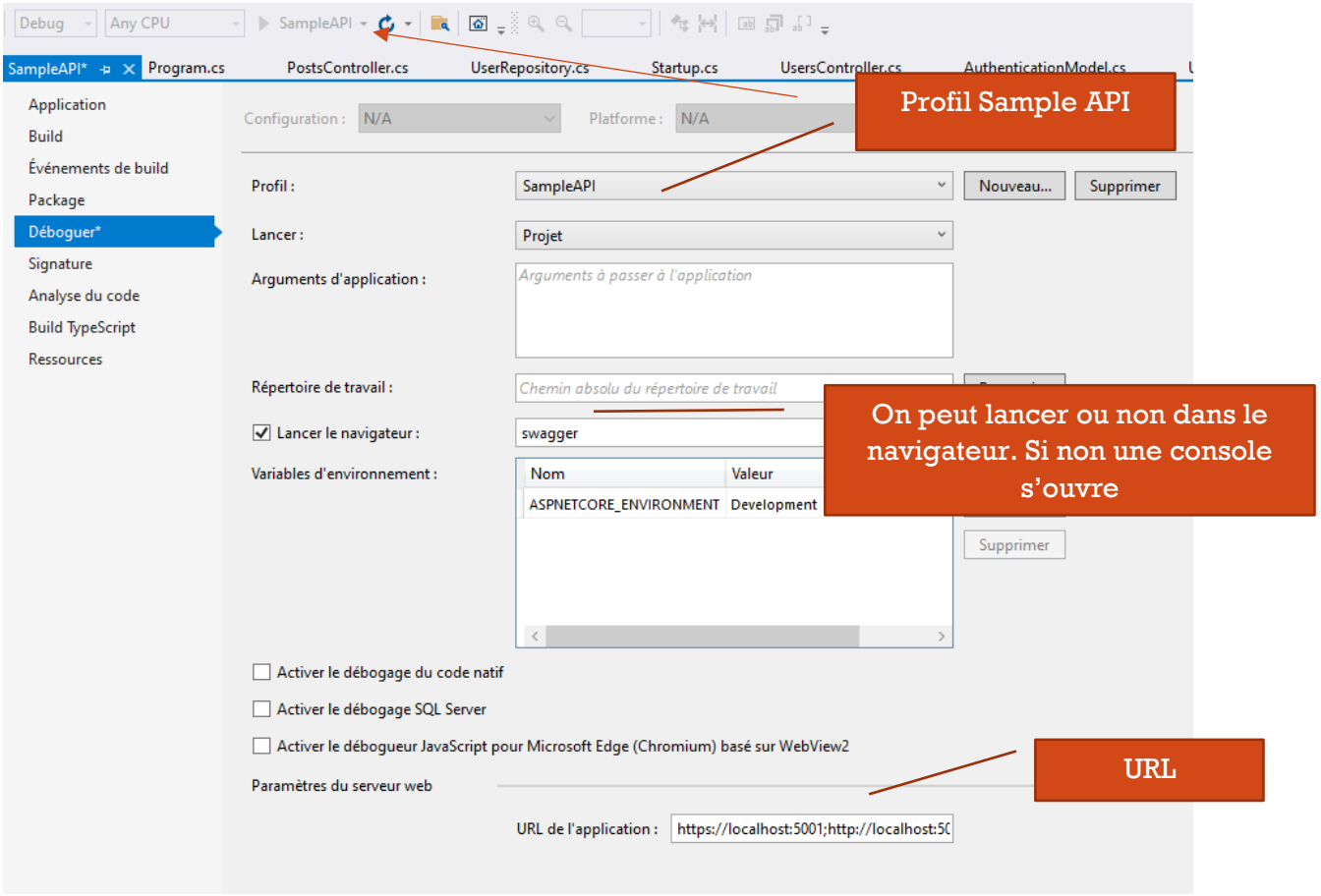
Déboggage

Plusieurs solutions :

- Profil IIS Express : lancer dans le navigateur avec Swashbuckle avec un port quelconque précisé



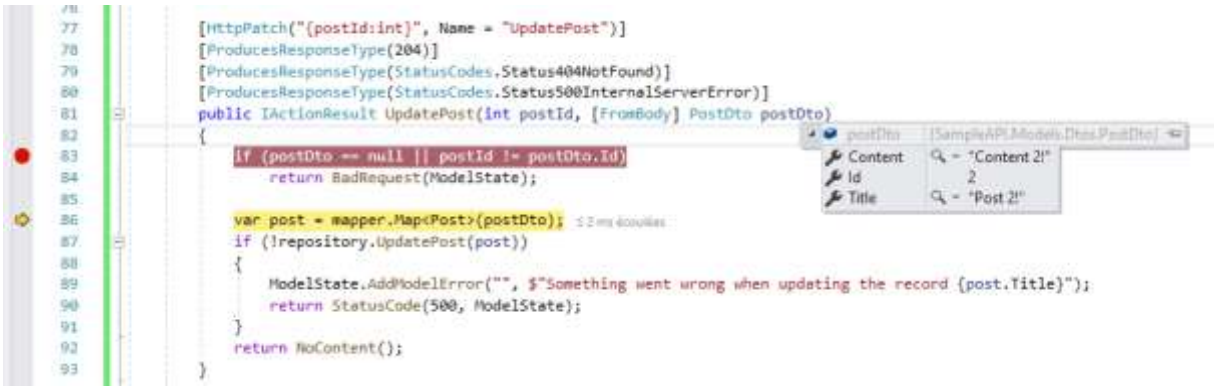
- Profil de l'application et décocher lancer dans le navigateur sur le port 5001 ou 5000



On utilise alors Postman (extension Chrome) pour tester

Breakpoints

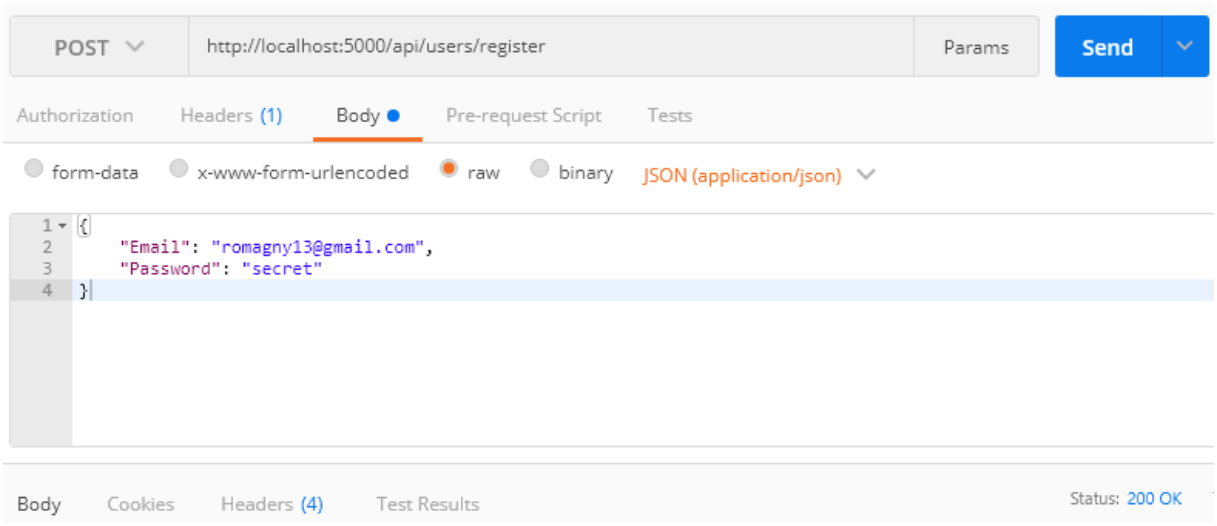
Même avec PostMan on peut mettre des breakpoints



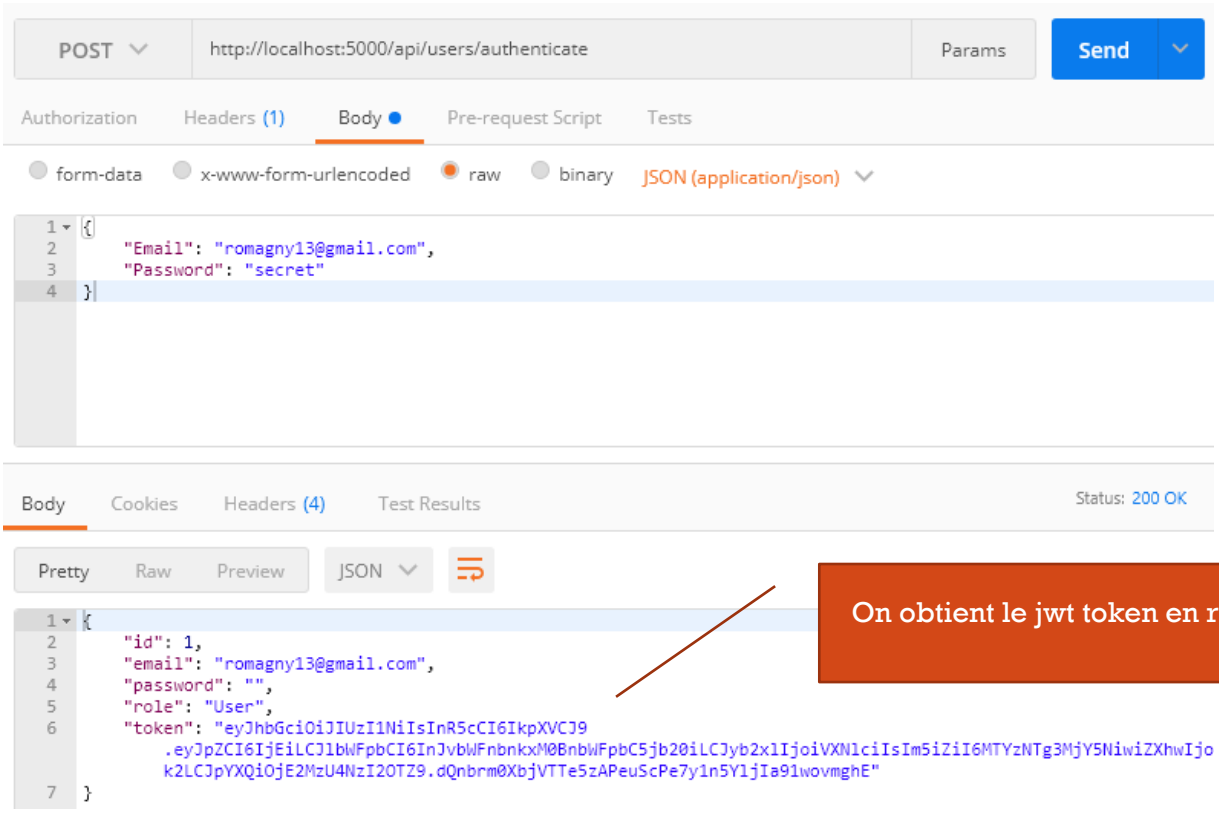
```
[HttpPatch("{postId:int}", Name = "UpdatePost")]
[ProducesResponseType(204)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public IActionResult UpdatePost(int postId, [FromBody] PostDto postDto)
{
    if (postDto == null || postId != postDto.Id)
        return BadRequest(ModelState);

    var post = mapper.Map<Post>(postDto);
    if (!repository.UpdatePost(post))
    {
        ModelState.AddModelError("", $"Something went wrong when updating the record {post.Title}");
        return StatusCode(500, ModelState);
    }
    return NoContent();
}
```

Exemple de requêtes avec Postman

Enregistrement de user

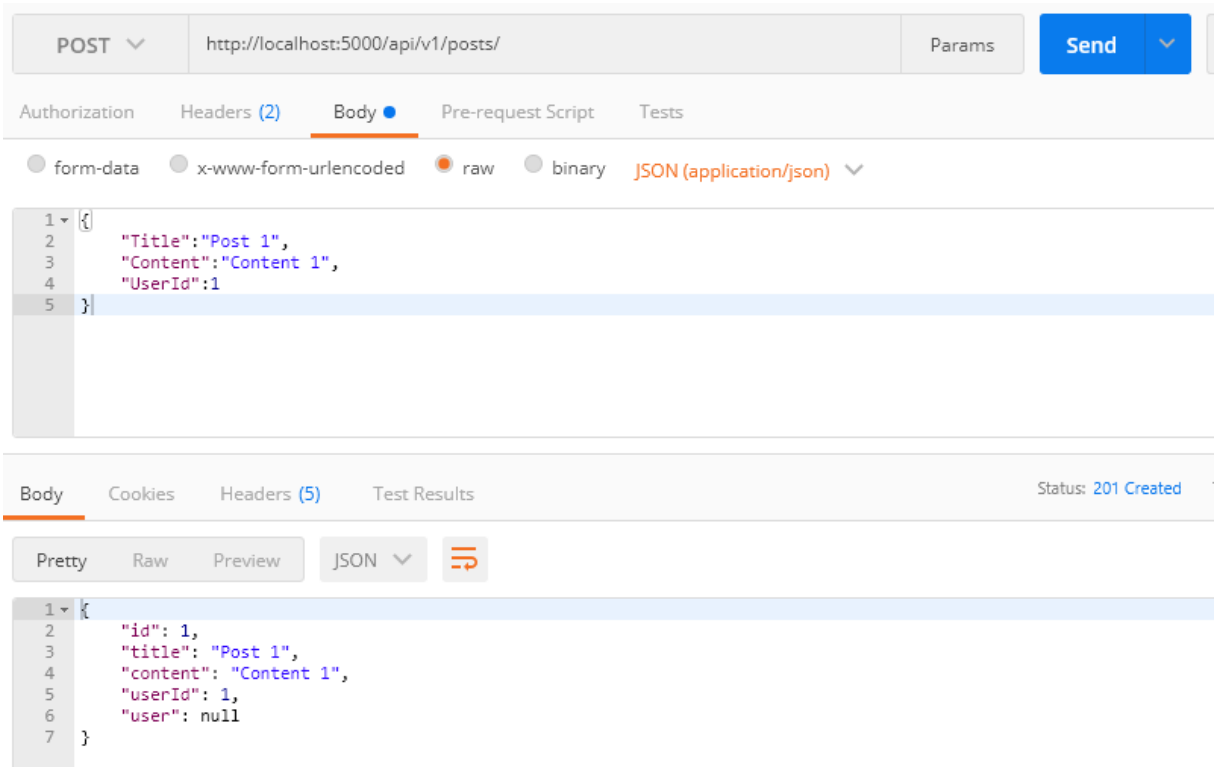


Authenticate

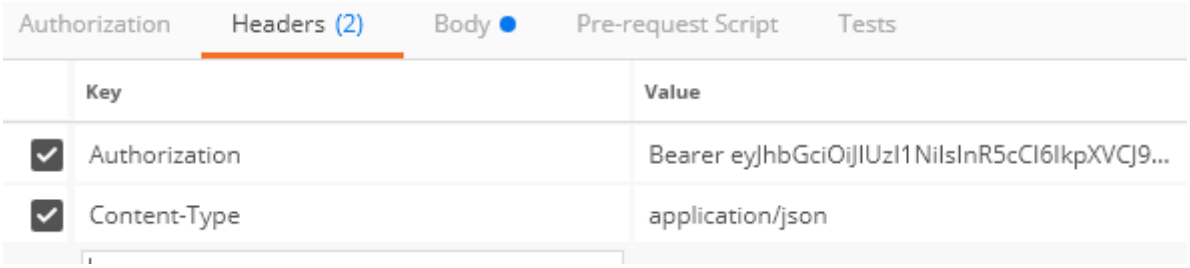


On obtient le jwt token en réponse

Création de post demandant à passer le token (sans le token on a une réponse de status 401 unauthorized)

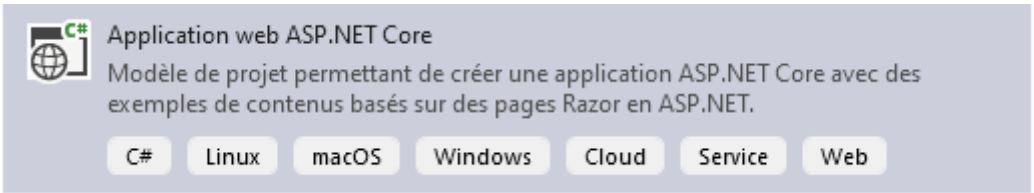


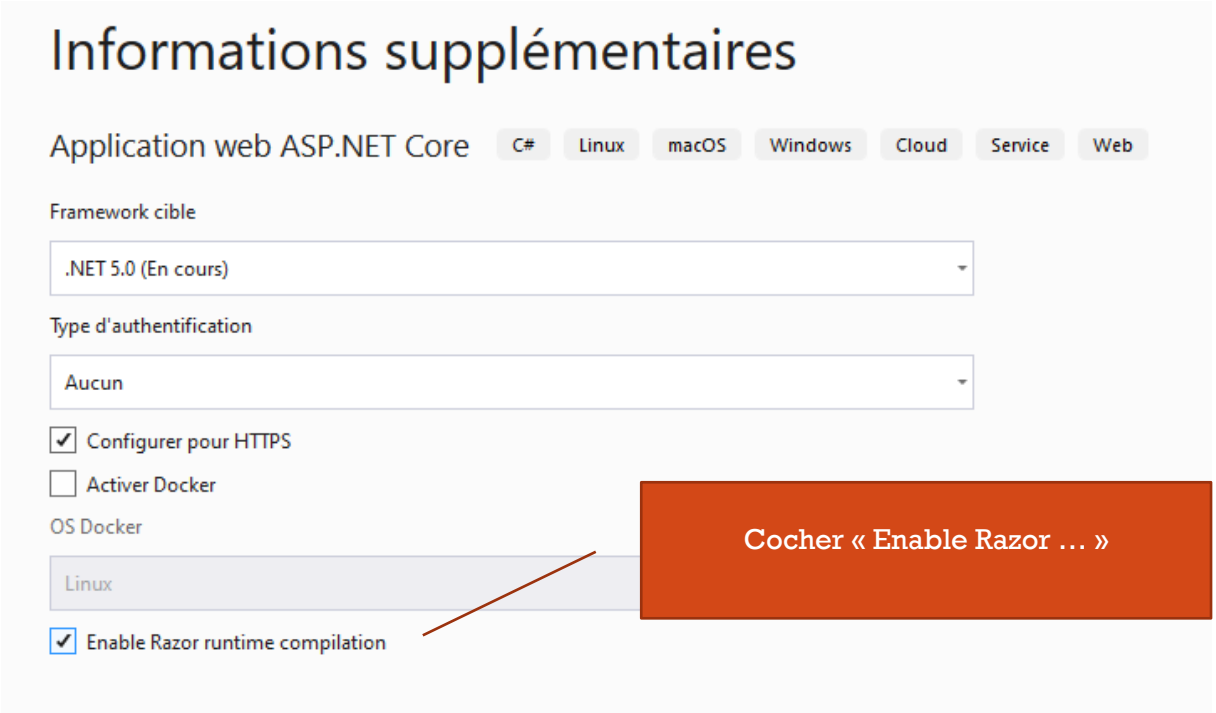
(headers)



Razor Pages

Création d'un projet



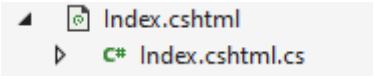


Pages

Le routing est basé sur le système de fichiers/dossiers avec le dossier « Pages »

Index.cshtml est la page d'accueil du site.

Les pages sont constituées d'une partie visuelle et d'une partie code (code-behind)

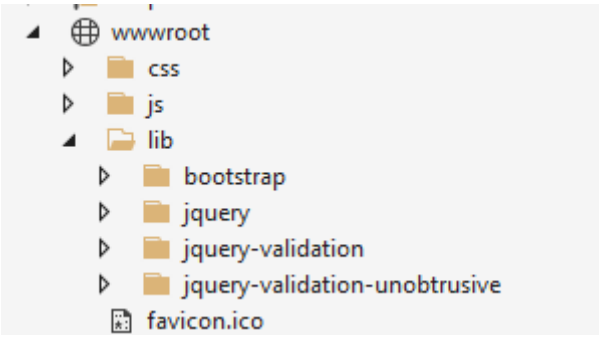


Ajout de links de styles / js (CDN)

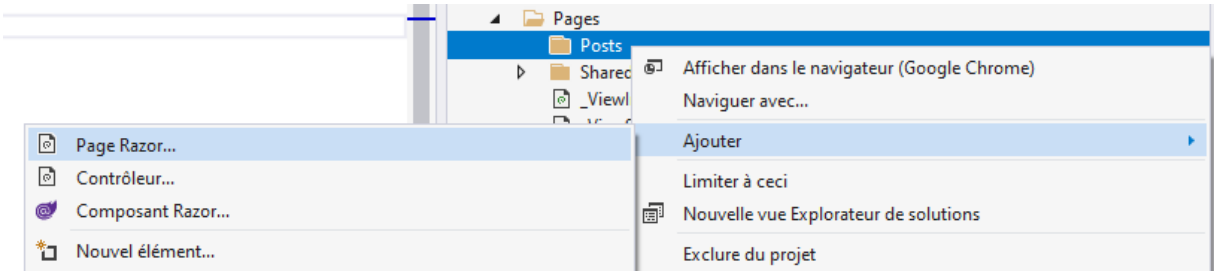
Il faut les ajouter à « _Layout.cshtml » (dossier « Shared ») dans le head et dans le pied de la page pour les scripts

Le dossier « wwwroot »

Permet d'ajouter les assets (fichiers locaux qui seront « public ») utilisés par le site

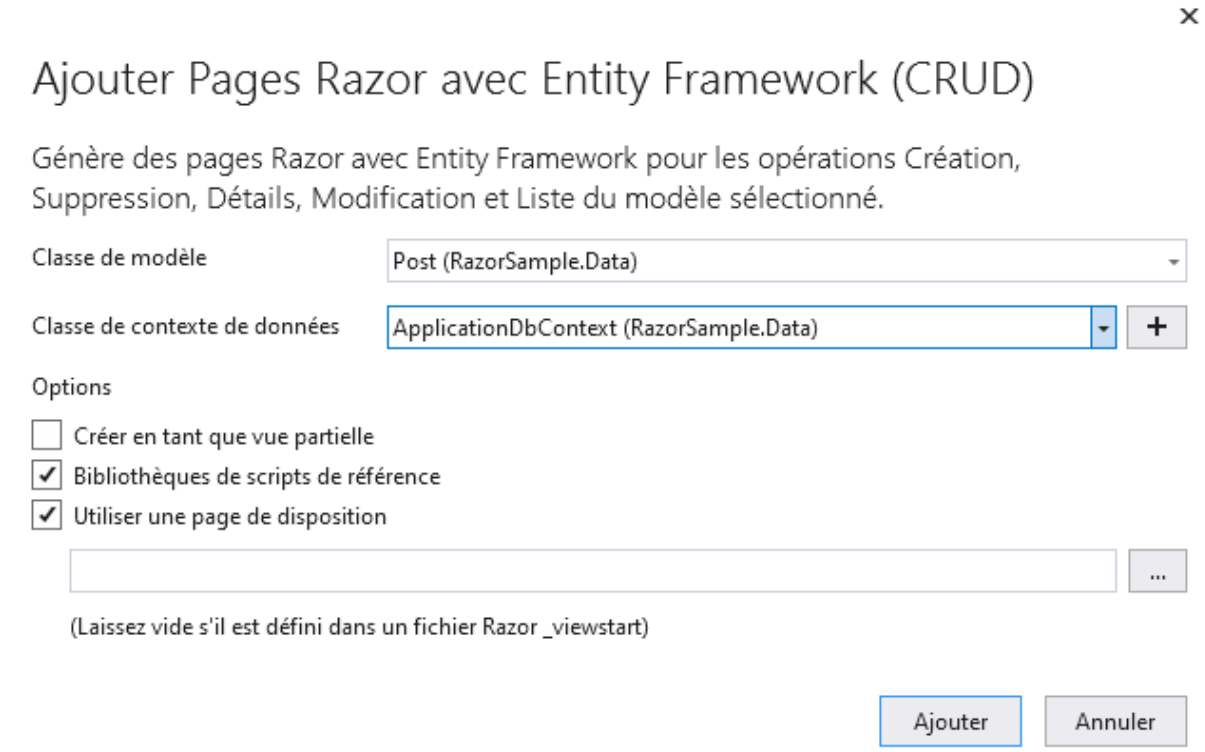


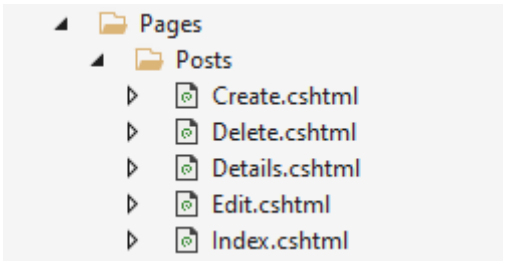
Ajout de Page Razor

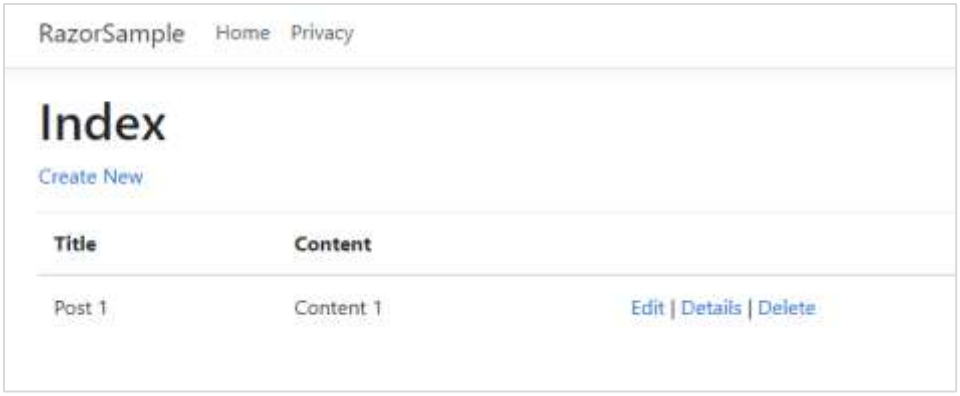


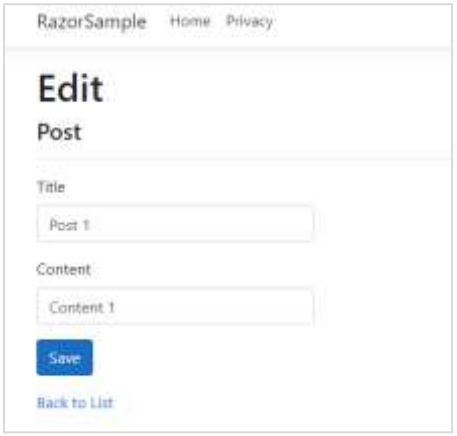
Pages avec CRUD

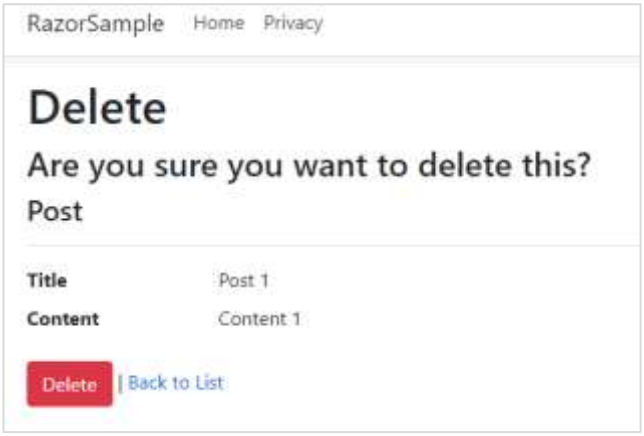
On peut également générer rapidement les pages CRUD





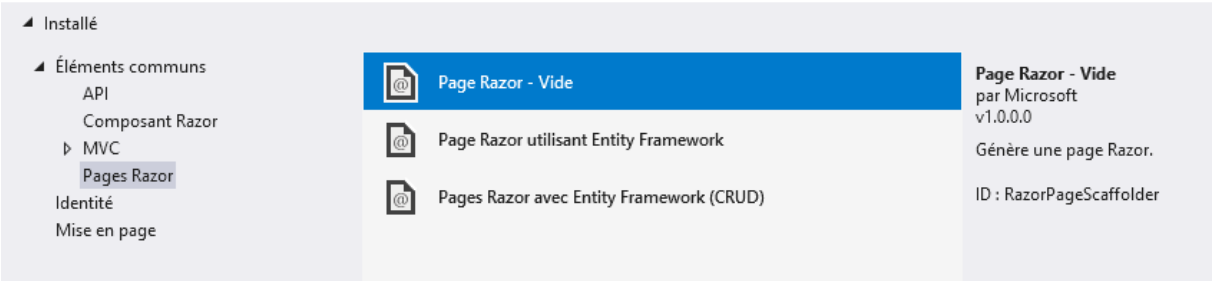






A partir de page vide

Ajouter un nouvel élément généré automatiquement



Ajout de « Index.cshtml » pour le dossier « Posts » affichant une liste de posts. On injecte le DbContext.

Les pages ont des handlers :

- OnGet
- OnPost, OnPostDelete (méthode Delete appelée depuis la vue)

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorSample.Data;
```

```
namespace RazorSample.Pages.Posts
{
   public class IndexModel : PageModel
   {
      private readonly ApplicationDbContext context;

      public IndexModel(ApplicationDbContext context)
      {
         this.context = context;
      }

      public IList<Post> Posts { get; set; }

      public async Task OnGet()
      {
         Posts = await context.Posts.ToListAsync();
      }

      // OnPost + Delete
      public async Task<IActionResult> OnPostDelete(int id)
      {
         var post = await context.Posts.FindAsync(id);
         if (post == null)
            return NotFound();

         context.Posts.Remove(post);
         await context.SaveChangesAsync();

         return RedirectToPage("Index");
      }
   }
}
```

Vue

```
@page
@model RazorSample.Pages.Posts.IndexModel
@{
   ViewData["Title"] = "Index";
}

<p>
   <a asp-page="Upsert">Create New</a>
</p>
@if (Model.Posts.Count > 0)
{
   <table class="table">
      <thead>
         <tr>
            <th>
               @Html.DisplayNameFor(model => model.Posts[0].Title)
            </th>
            <th>
```

```
                @Html.DisplayNameFor(model => model.Posts[0].Content)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>

        @foreach (var item in Model.Posts)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Content)
                </td>
                <td>
                    <a asp-page="Upsert" asp-route-id="@item.Id">Edit</a> |
                    @*<a onclick="return confirm('Are you sure you want to delete this post?')" asp-page-handler="Delete" asp-route-id="@item.Id" >Delete</a>*@

                    <a href="#" onclick="Delete('api/Posts?id=' + @item.Id)">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
}
else
{
    <p>No posts.</p>
}
```

Appelle « OnPostDelete » (Delete en asp-page-handler) en passant le paramètre avecasp-route-id (asp-route + id)

Autre méthode de deletion avec controller

Create + Edit = Upsert

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorSample.Data;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace RazorSample.Pages.Posts
{
    public class UpsertModel : PageModel
    {
        private readonly ApplicationDbContext context;

        public UpsertModel(ApplicationDbContext context)
        {
            this.context = context;
        }

        [BindProperty]
        public Post Post { get; set; }

        public async Task<IActionResult> OnGet(int? id)
```

Besoin de bind la propriété sinon dans onPost on aurait null

```
    {
        if (id == null)
        {
            Post = new Post();
            return Page();
        }
        else
        {
            Post = await context.Posts.FirstOrDefaultAsync(x => x.Id == id);
            if (Post == null)
                return NotFound();

            return Page();
        }
    }

    public async Task<IActionResult> OnPost()
    {
        if (ModelState.IsValid)
        {
            if (Post.Id == 0)
                context.Posts.Add(Post);
            else
                context.Posts.Update(Post);

            await context.SaveChangesAsync();

            return RedirectToPage("Index");
        }
        return RedirectToPage();
    }
  }
}
```

Adaptation des labels selon ajout ou édition

```
@page
@model RazorSample.Pages.Posts.UpsertModel
@{
   ViewData["Title"] = Model.Post.Id == 0 ? "Create" : "Edit";
}

<h1>@(Model.Post.Id == 0?  "Create" : "Edit")</h1>

<h4>Post</h4>
<hr />
<div class="row">
   <div class="col-md-4">
      <form method="post">

         @if (Model.Post.Id != 0)
         {
            <input type="hidden" asp-for="Post.Id"/>
         }

         <div asp-validation-summary="ModelOnly" class="text-danger"></div>
         <div class="form-group">
            <label asp-for="Post.Title" class="control-label"></label>
            <input asp-for="Post.Title" class="form-control" />
```

Ajout d'un champ hidden avec id si le model a un id (edition)

```
            <span asp-validation-for="Post.Title" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Post.Content" class="control-label"></label>
            <input asp-for="Post.Content" class="form-control" />
            <span asp-validation-for="Post.Content" class="text-danger"></span>
        </div>
        <div class="form-group">
            <input type="submit" value="@(Model.Post.Id == 0?  "Create" : "Edit")"
class="btn btn-primary" />
        </div>
    </form>
  </div>
</div>

<div>
  <a asp-page="Index">Back to List</a>
</div>

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Affichage de valeur avec asp-for

Affichage des errors « asp-validation-for »

Pour la validation jQuery (avec data annotations sur entities) et affiche des errors dans la page

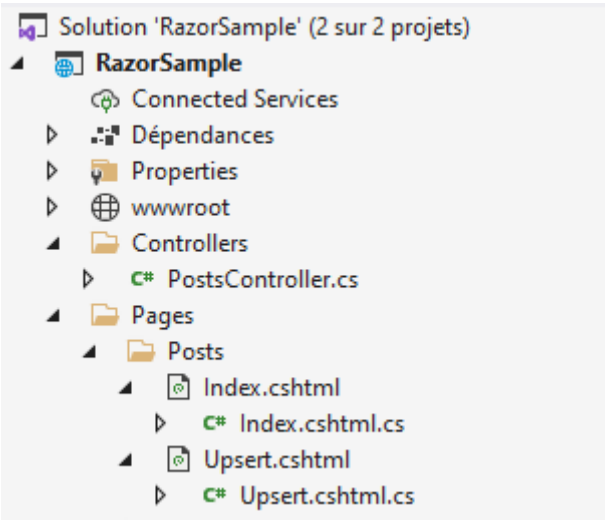
Interaction avec controller

On peut très bien ajouter des controllers. Il faut enregistrer dans Startup.cs

```
services.AddControllers();
```

Et configurer

```
app.UseEndpoints(endpoints =>
{
            endpoints.MapRazorPages();
            endpoints.MapControllers();
});
```



Exemple on crée une méthode delete (appelée en javascript)

```
using Microsoft.AspNetCore.Mvc;
using RazorSample.Data;
using System;
using System.Threading.Tasks;

namespace RazorSample.Controllers
{
   [Route("api/[controller]")]
   public class PostsController : Controller
   {
      private readonly ApplicationDbContext context;

      public PostsController(ApplicationDbContext context)
      {
         this.context = context;
      }

      [HttpDelete]
      public async Task<IActionResult> Delete(int id)
      {
         try
         {
            var post = await context.Posts.FindAsync(id);
            if (post == null)
            {
               return Json(new { success = false, message = "Error while deleting post" });
            }

            context.Posts.Remove(post);
            await context.SaveChangesAsync();
            return Json(new { success = true, message = "Post deleted" });
         }
         catch (Exception)
         {
            return Json(new { success = false, message = "Error while deleting post" });
         }
      }
   }
}
```

On ajoute au fichier site.js (du dossier « wwwroot ») une function utilisant la libraire sweetalert pour afficher une boite de confirmation un peu plus jolie

```
function Delete(url) {
   swal({
      title: "Are you sure?",
      text: "Are you sure you want to delete this post ?",
      icon: "warning",
      buttons: true,
      dangerMode: true,
   })
      .then((willDelete) => {
         if (willDelete) {
```

```
            $.ajax({
                method: "DELETE",
                url: url,
                success: function (data) {
                    if (data.success) {
                        toastr.success(data.message);
                        window.location.reload();
                    }
                    else {
                        toastr.error(data.message);
                    }
                }
            })
        }
    });
}
```

On affiche un toast et on reload la page en cas de suppression

Dans la page on crée un lien qui appelle la function javascript en passant l'url de l'api

```
<a href="#" onclick="Delete('api/Posts?id=' + @item.Id)">Delete</a>
```

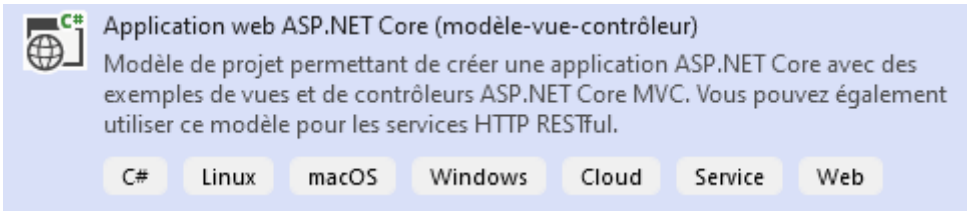
Les links ajoutés à _Layout.cshtml

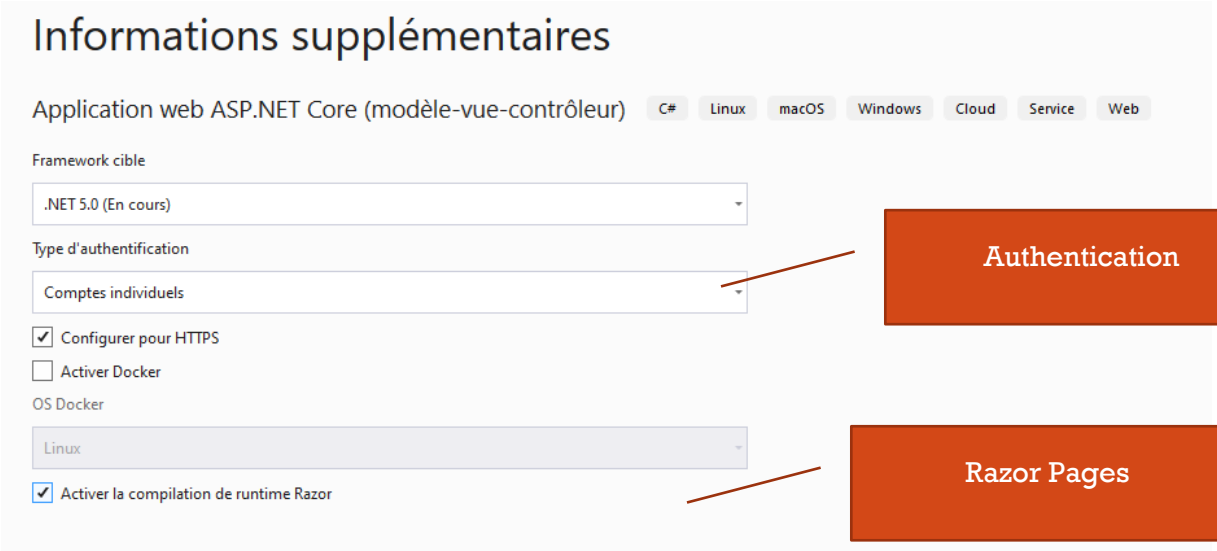
```
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/sweetalert/1.1.3/sweetalert.min.css" />
```

Et scripts

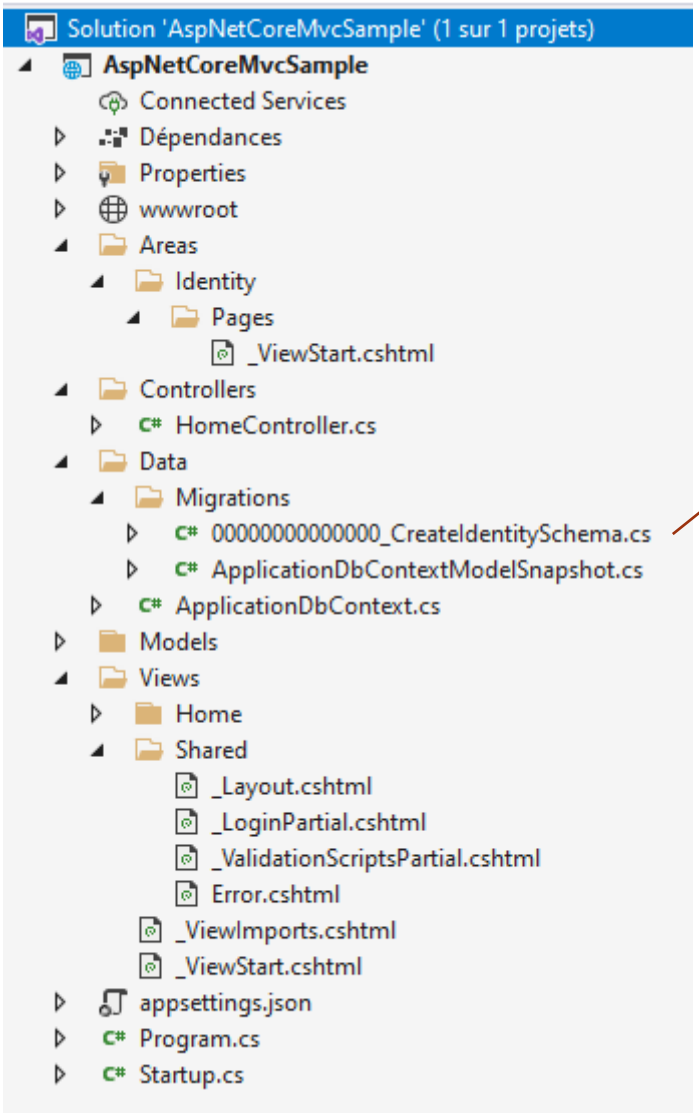
```
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
 <script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
```

MVC

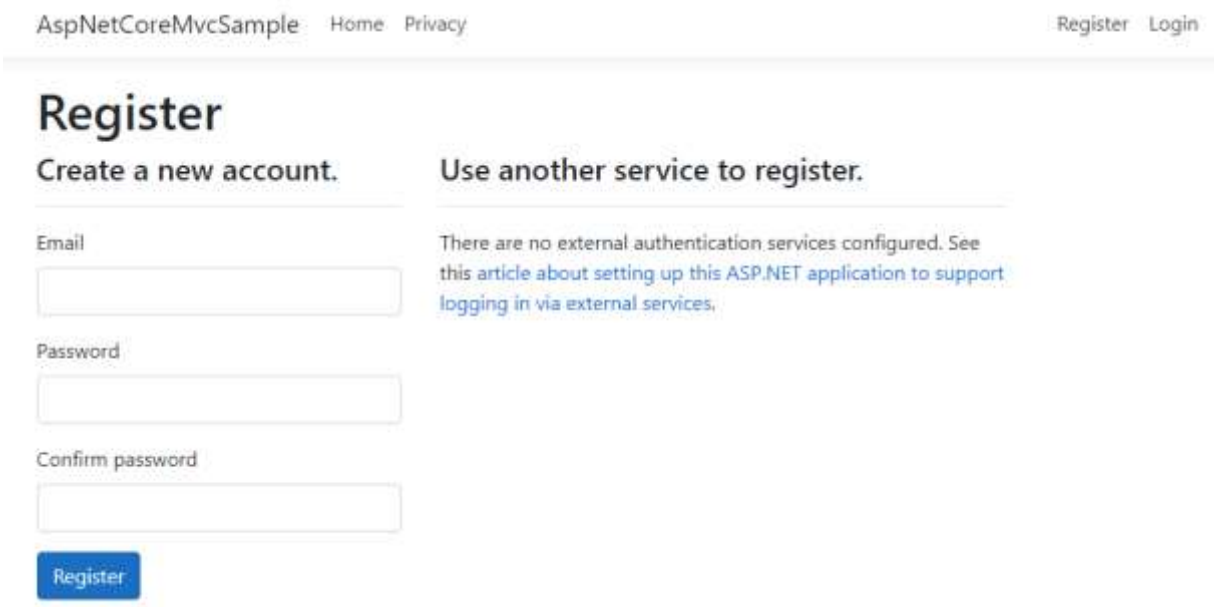




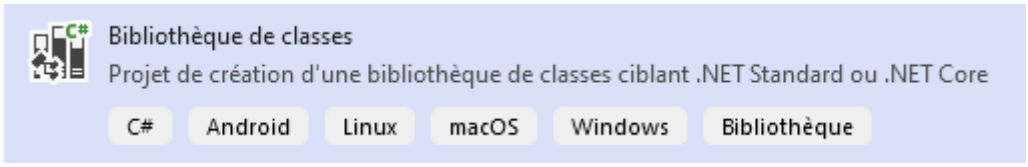
Structure du projet créé



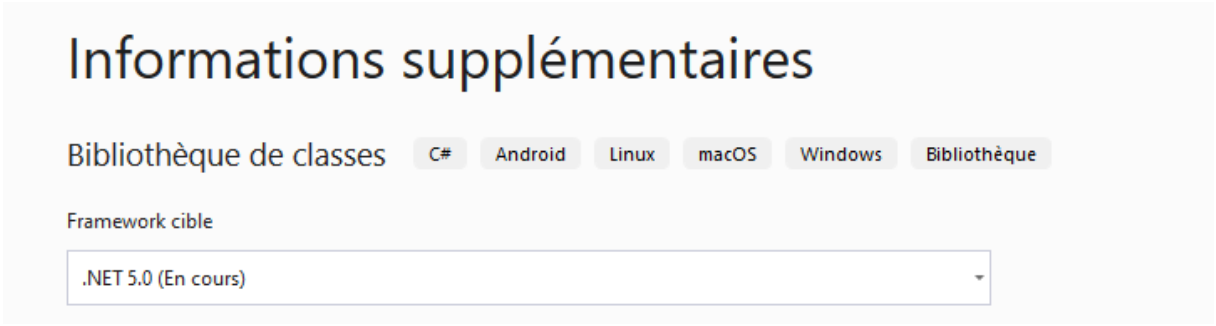
Un IdentityDbContext, la migration et une chaine de connexion (appsettings.json) sont ajoutées de base



Création d'un projet « Data »



Sélection de la version de .NET Core



Déplacer le le dossier « Data » dans le projet « Data »

Ajout des packages références au csproj du projet « Data »

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="5.0.5" />
  <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="5.0.5" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.5" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="5.0.5" />
</ItemGroup>
```

Package « Microsoft.AspNetCore.Identity.EntityFrameworkCore » car on utilise un **IdentityDbContext**

Dans le projet principal

- Ajout d'une référence au projet « Data »
- (La chaine de connexion est déjà présente dans appsettings.json

```
 "ConnectionStrings": {
   "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-
AspNetCoreMvcSample;Trusted_Connection=True;MultipleActiveResultSets=true"
 },
```

- Ainsi que l'enregistrement du DbContext dans Startup.cs)

```
services.AddDbContext<ApplicationDbContext>(options =>
   options.UseSqlServer(
      Configuration.GetConnectionString("DefaultConnection")));

services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
   .AddEntityFrameworkStores<ApplicationDbContext>();
```

IdentityUser

Création d'une classe ApplicationUser avec des champs supplémentaires

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace AspNetCoreMvcSample.Data
{
   public class ApplicationUser : IdentityUser
   {
      [Required]
      public string FirstName { get; set; }
      [Required]
      public string LastName { get; set; }

      public string StreetAddress { get; set; }
      public string City { get; set; }
      public string PostalCode { get; set; }

      [NotMapped]
      public string Role { get; set; }
   }
}
```

Ajout à l'IdentityDbContext

```
public DbSet<ApplicationUser> ApplicationUsers { get; set; }
```

Ajout d'une **migration**

```
Add-Migration AddUserPropertiesTodDb
```

Puis modification de la base de données

```
Update-Database
```

Enregistrement Startup.cs , remplacer

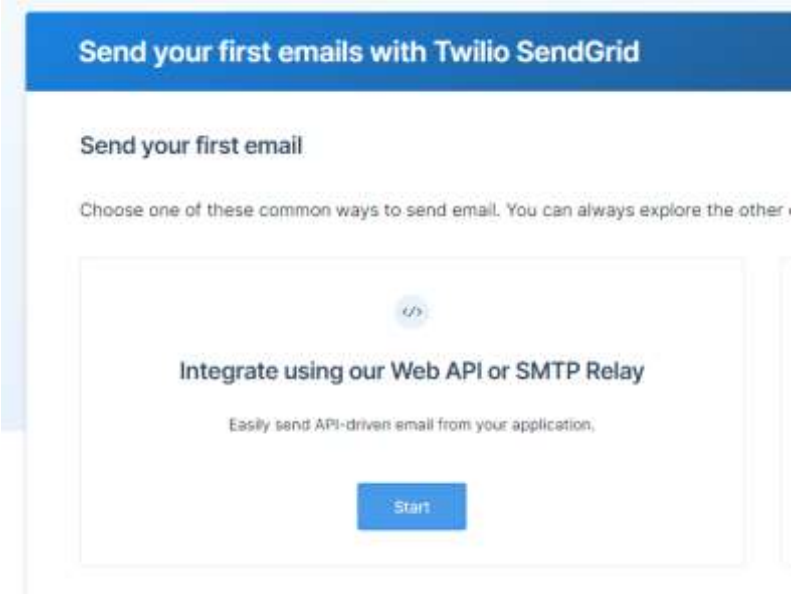
```
        services.AddDbContext<ApplicationDbContext>(options =>
```

```
            options.UseSqlServer(
                Configuration.GetConnectionString("DefaultConnection")));
        services.AddDatabaseDeveloperPageExceptionFilter();

        //services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= true)
        //    .AddEntityFrameworkStores<ApplicationDbContext>();

        services.AddIdentity<IdentityUser, IdentityRole>().AddDefaultTokenProviders()
          .AddEntityFrameworkStores<ApplicationDbContext>();
```

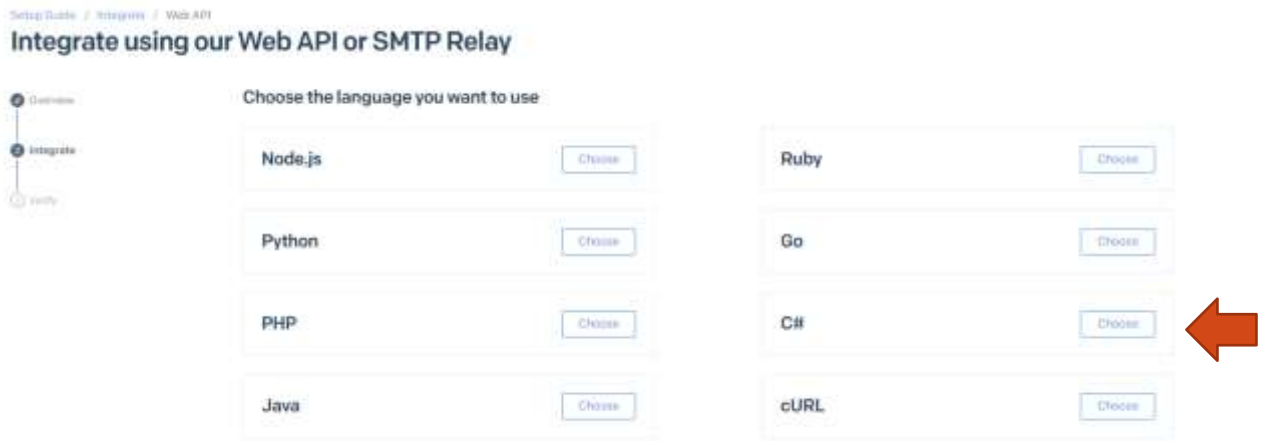
Configurer EmailSender

Avec SendGrid : https://sendgrid.com/

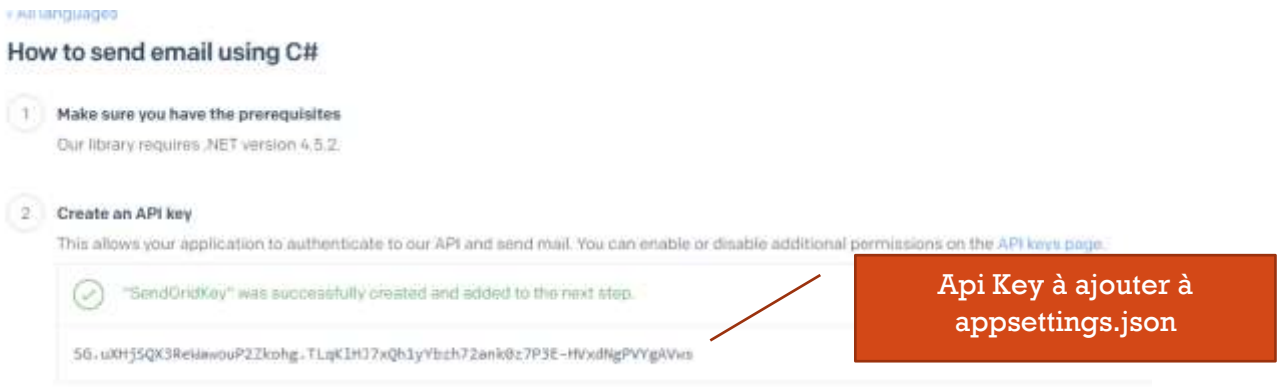
Créer un compte (on peut « startup for free »)



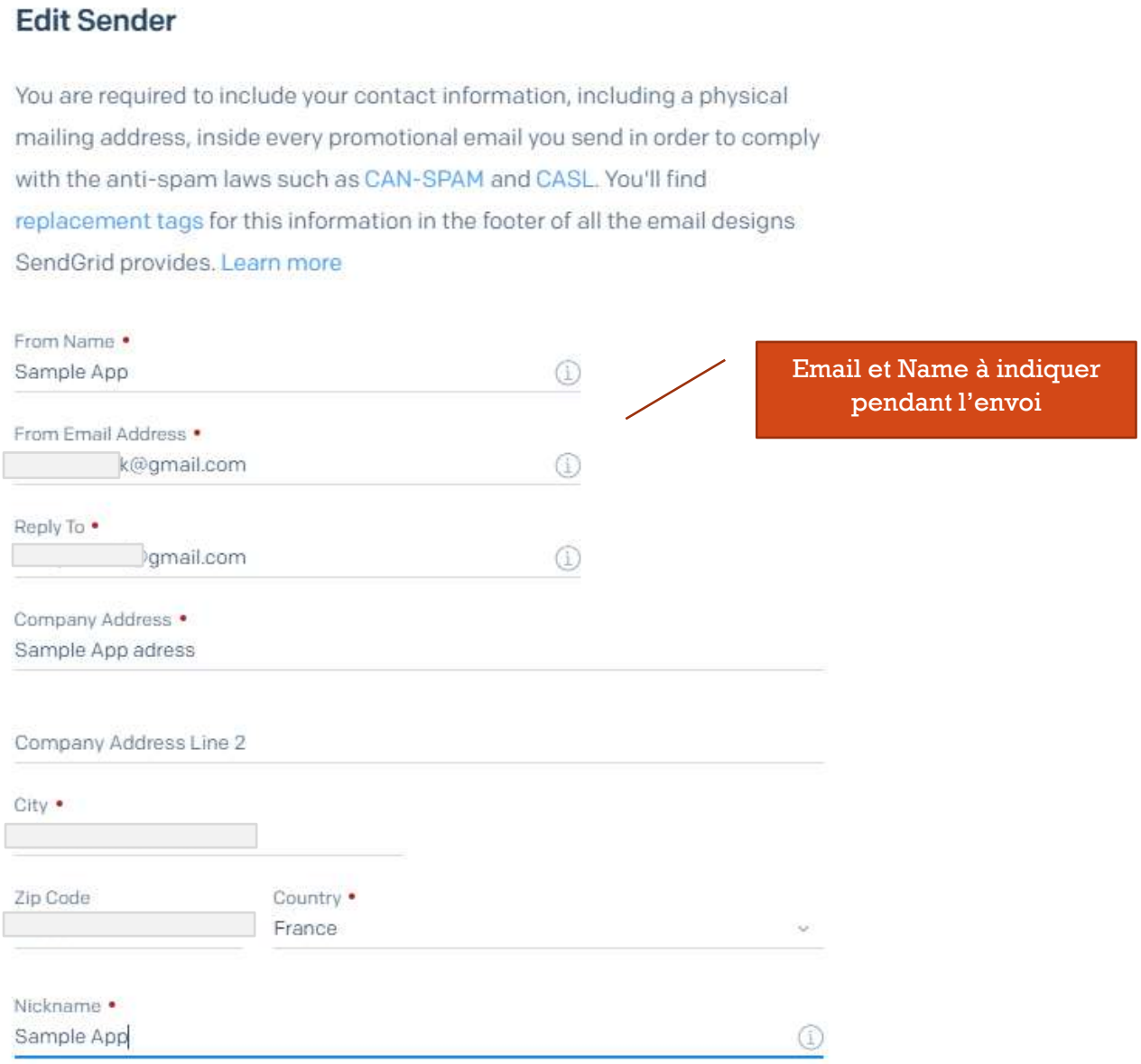
… Sélectionner C#



La démarche est ensuite expliquée. Donner un nom à la clé à créer « SenGridKey » par exemple et cliquer sur créer key



Create a **sender** (demande une vérification d'email du sender créé)



Installer le package

```
Install-Package SendGrid
```

Ajouter à appsettings.json la clé

```
"SendGridKey": "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"
```

Créer une classe EmailOptions pour le binding

```
public class EmailOptions
{
   public string SendGridKey { get; set; }
}
```

.. et l'enregistrer

```
services.Configure<EmailOptions>(Configuration);
```

Création de la EmailSender qui implémente IEmailSender et utilsant SendGrid

```
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Options;
using SendGrid;
using SendGrid.Helpers.Mail;
using System;
using System.Net;
using System.Threading.Tasks;

namespace AspNetCoreMvcSample.Util
{
  public class EmailOptions
  {
    public string SendGridKey { get; set; }
  }

  public class EmailSender : IEmailSender
  {
    private readonly EmailOptions emailOptions;

    public EmailSender(IOptions<EmailOptions> options)
    {
      emailOptions = options.Value;
    }
    public Task SendEmailAsync(string email, string subject, string htmlMessage)
    {
      return Execute(emailOptions.SendGridKey, subject, htmlMessage, email);
    }
    private Task Execute(string sendGridKEy, string subject, string message, string email)
    {
      var client = new SendGridClient(sendGridKEy);
      var from = new EmailAddress("XXXXXX@gmail.com", "Sample App"); // replace
      var to = new EmailAddress(email, "End User");
      var msg = MailHelper.CreateSingleEmail(from, to, subject, "", message);
      return client.SendEmailAsync(msg);
      // ou pour vérifier en cas d'erreur
      //var response = await client.SendEmailAsync(msg);
      //if (response.StatusCode != HttpStatusCode.OK
      //   && response.StatusCode != HttpStatusCode.Accepted)
      //{
      //   var errorMessage = response.Body.ReadAsStringAsync().Result;
      //   throw new Exception($"Failed to send mail to {to}, status code {response.StatusCode}, {errorMessage}");
      //}
```

Le mail et nom doivent correspondre au Sender

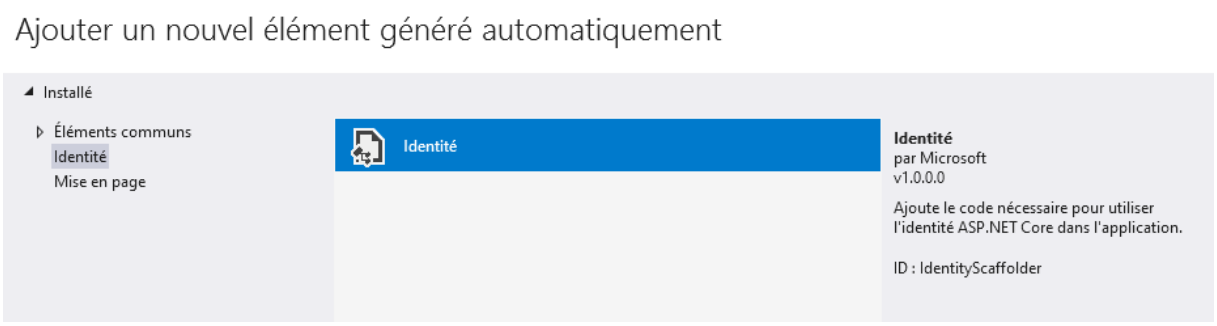
```
    }
  }
}
```

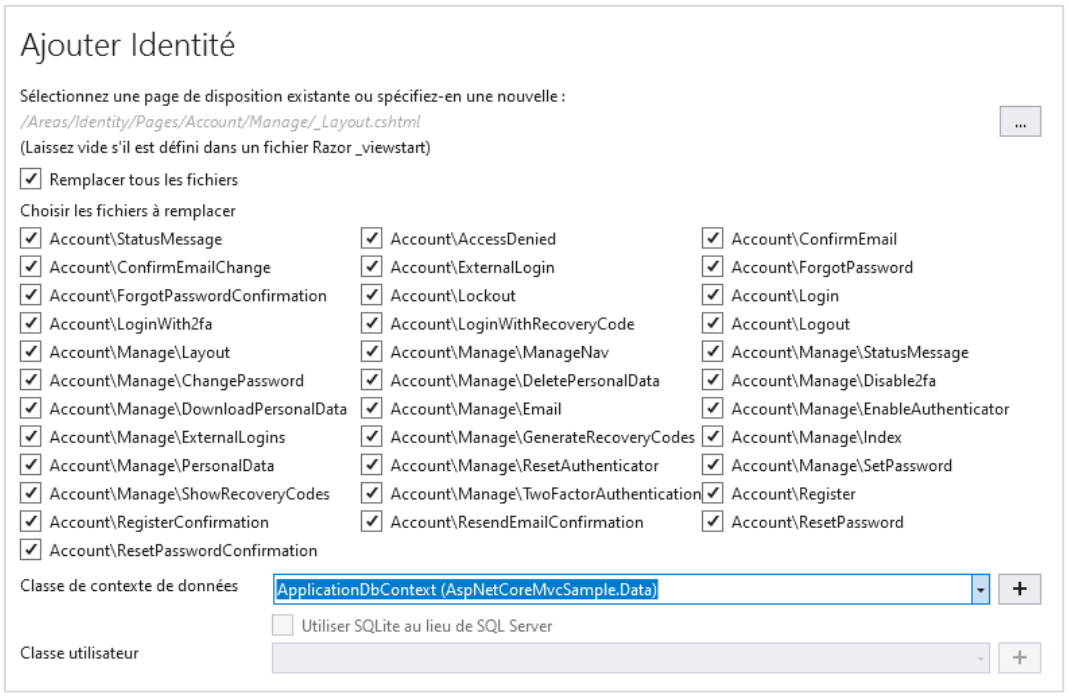
Enregistrer dans Startup.cs

```
services.AddSingleton<IEmailSender, EmailSender>();
```

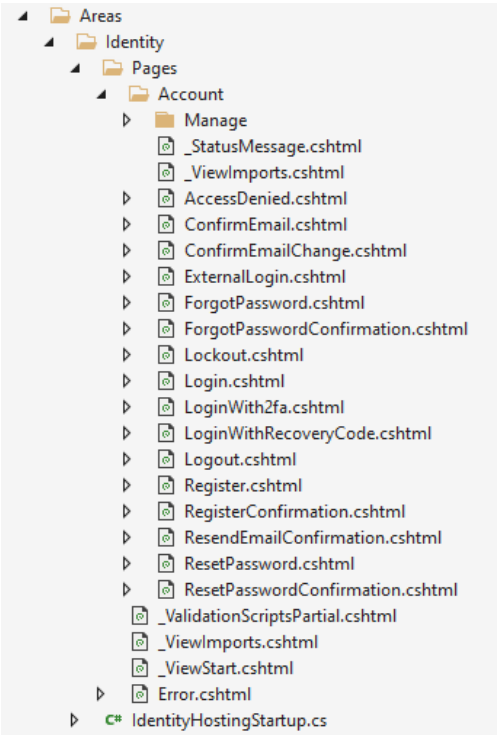
Pour modifier la vue « Register »

Clic droit sur le projet principal … « Ajouter » … « Nouvel élément généré automatiquement » … puis « Identité »





Toutes les razor pages sont générées pour l'area « Identity »



On peut donc ajouter des champs à Register.cshtml pour l'enregistrement de nouvel utilisateur

Copier coller des propriétés de ApplicationUser dans InputModel (code-behing de Register.cshtml)

```
        public class InputModel
        {
            [Required]
            [EmailAddress]
            [Display(Name = "Email")]
            public string Email { get; set; }

            [Required]
            [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1}
characters long.", MinimumLength = 6)]
            [DataType(DataType.Password)]
            [Display(Name = "Password")]
            public string Password { get; set; }

            [DataType(DataType.Password)]
            [Display(Name = "Confirm password")]
            [Compare("Password", ErrorMessage = "The password and confirmation
password do not match.")]
            public string ConfirmPassword { get; set; }

            [Required]
            public string FirstName { get; set; }
            [Required]
            public string LastName { get; set; }
            public string StreetAddress { get; set; }
            public string City { get; set; }
```

On peut en plus ajouter une property PhoneNumber qui n'est pas indiquée dans ApplicationUser

```
        public string PostalCode { get; set; }
        public string Role { get; set; }

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

La classe IdentityUser de base

```
public class IdentityUser<TKey> where TKey : IEquatable<TKey>
{
    public IdentityUser();
    public IdentityUser(string userName);

    public virtual DateTimeOffset? LockoutEnd { get; set; }
    public virtual bool TwoFactorEnabled { get; set; }
    public virtual bool PhoneNumberConfirmed { get; set; }
    public virtual string PhoneNumber { get; set; }
    public virtual string ConcurrencyStamp { get; set; }
    public virtual string SecurityStamp { get; set; }
    public virtual string PasswordHash { get; set; }
    public virtual bool EmailConfirmed { get; set; }
    public virtual string NormalizedEmail { get; set; }
    public virtual string Email { get; set; }
    public virtual string NormalizedUserName { get; set; }
    public virtual string UserName { get; set; }
    public virtual TKey Id { get; set; }
    public virtual bool LockoutEnabled { get; set; }
    public virtual int AccessFailedCount { get; set; }

    public override string ToString();
}
```

Exemple de form

```
<form asp-route-returnUrl="@Model.ReturnUrl" method="post">
  <h4>Create a new account.</h4>
  <hr />
  <div asp-validation-summary="All" class="text-danger"></div>
  <div class="form-row py-2">
    <div class="col">
      <input asp-for="Input.FirstName" placeholder="First Name" class="form-control" />
      <span asp-validation-for="Input.FirstName" class="text-danger"></span>
    </div>
    <div class="col">
      <input asp-for="Input.LastName" placeholder="Last Name" class="form-control" />
      <span asp-validation-for="Input.LastName" class="text-danger"></span>
    </div>
  </div>
  <div class="py-2">
    <input asp-for="Input.Email" type="email" placeholder="Email" class="form-control" />
    <span asp-validation-for="Input.Email" class="text-danger"></span>
  </div>
  <div class="py-2 input-group">
    <div class="input-group-prepend">
      <span class="input-group-text"> +1</span>
    </div>
    <input asp-for="Input.PhoneNumber" type="text" placeholder="Phone Number" class="form-control"
/>
```

```
      <span asp-validation-for="Input.PhoneNumber" class="text-danger"></span>
    </div>
    <div class="py-2">
      <input asp-for="Input.StreetAddress" placeholder="Street Address" class="form-control" />
      <span asp-validation-for="Input.StreetAddress" class="text-danger"></span>
    </div>
    <div class="form-row py-2">
      <div class="col">
        <input asp-for="Input.PostalCode" placeholder="Postal Code" class="form-control" />
        <span asp-validation-for="Input.PostalCode" class="text-danger"></span>
      </div>
      <div class="col">
        <input asp-for="Input.City" placeholder="City" class="form-control" />
        <span asp-validation-for="Input.City" class="text-danger"></span>
      </div>
    </div>
    <div class="form-row py-2">
      <div class="col">
        <input asp-for="Input.Password" type="password" placeholder="Password" class="form-control"
/>
        <span asp-validation-for="Input.Password" class="text-danger"></span>
      </div>
      <div class="col">
        <input asp-for="Input.ConfirmPassword" type="password" placeholder="Confirm Password"
class="form-control" />
        <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
      </div>
    </div>
    <button type="submit" class="btn btn-primary">Register</button>
</form>
```

Dans RegisterModel sont injectés les services utiles pour l'enregistrement. EmailSender doit être configuré (avec SendGrid par exemple). On pourrait en plus injecter RoleManager pour récupérer la liste des rôles par exemple.

```
    [AllowAnonymous]
    public class RegisterModel : PageModel
    {
        private readonly SignInManager<IdentityUser> _signInManager;
        private readonly UserManager<IdentityUser> _userManager;
        private readonly ILogger<RegisterModel> _logger;
        private readonly IEmailSender _emailSender;

        public RegisterModel(
            UserManager<IdentityUser> userManager,
            SignInManager<IdentityUser> signInManager,
            ILogger<RegisterModel> logger,
            IEmailSender emailSender)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _logger = logger;
            _emailSender = emailSender;
        }
```

Enregistrement de nouveau user

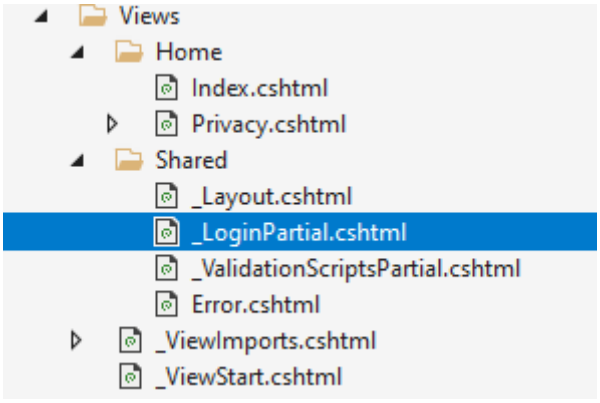
```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
   returnUrl ??= Url.Content("~/");
   ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
   if (ModelState.IsValid)
   {
      // replace
      // var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
      var user = new ApplicationUser
      {
         UserName = Input.Email,
         FirstName = Input.FirstName,
         LastName = Input.LastName,
         Email = Input.Email,
         StreetAddress = Input.StreetAddress,
         City = Input.City,
         PostalCode = Input.PostalCode,
         PhoneNumber = Input.PhoneNumber,
         Role = Input.Role ?? "Customer"
      };

      var result = await _userManager.CreateAsync(user, Input.Password);
      if (result.Succeeded)
      {
         _logger.LogInformation("User created a new account with password.");

         // ajout au role
         await _userManager.AddToRoleAsync(user, user.Role);

         var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
         code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
         var callbackUrl = Url.Page(
            "/Account/ConfirmEmail",
            pageHandler: null,
            values: new { area = "Identity", userId = user.Id, code = code, returnUrl = returnUrl },
            protocol: Request.Scheme);

         await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
            $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking
here</a>.");

         if (_userManager.Options.SignIn.RequireConfirmedAccount)
         {
            return RedirectToPage("RegisterConfirmation", new { email = Input.Email, returnUrl = returnUrl
});
         }
         else
         {
            await _signInManager.SignInAsync(user, isPersistent: false);
            return LocalRedirect(returnUrl);
         }
      }
      foreach (var error in result.Errors)
      {
         ModelState.AddModelError(string.Empty, error.Description);
      }
   }

   // If we got this far, something failed, redisplay form
   return Page();
}
```

On remplace IdentityUser par ApplicationUser pour définir les champs supplémentaires

On ajoute le user au role

Boutons Register et Login du menu

C'est _LoginPartial qui permet d'afficher dans le menu les boutons Register et Login

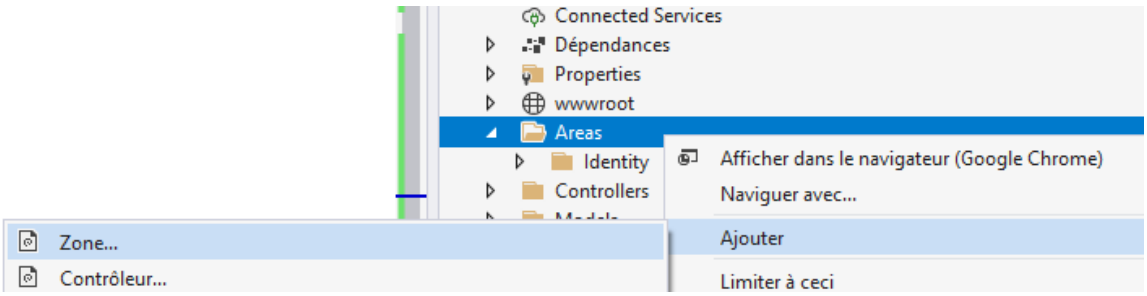


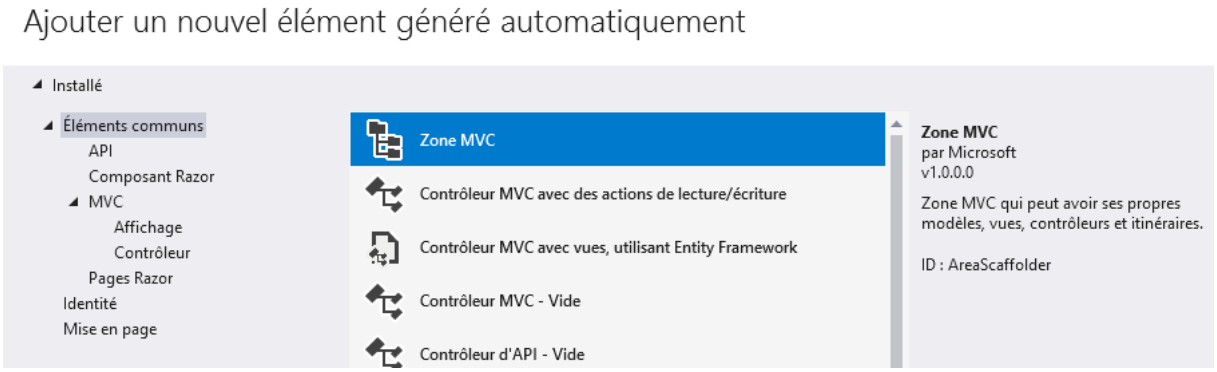
Elle est insérée dans le code de _Layout.cshtml

```
<partial name="_LoginPartial" />
```

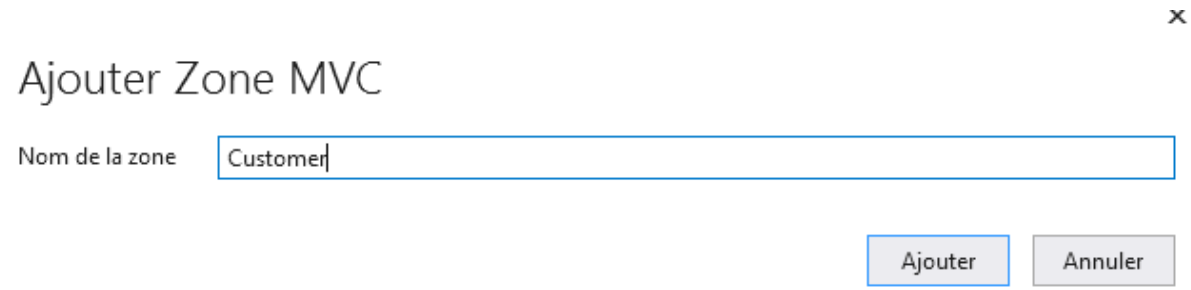
Création d'Area

Si le dosser « Area » … « Ajouter » … « Zone »

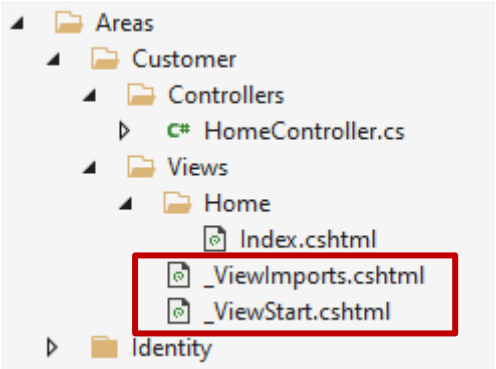




Donner le nom de la zone. Exemple « Customer » (pour le ShoppingCart)



Copier-coller les views _ViewImports et _ViewStart



Faire pointer _ViewStart vers le _Layout

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

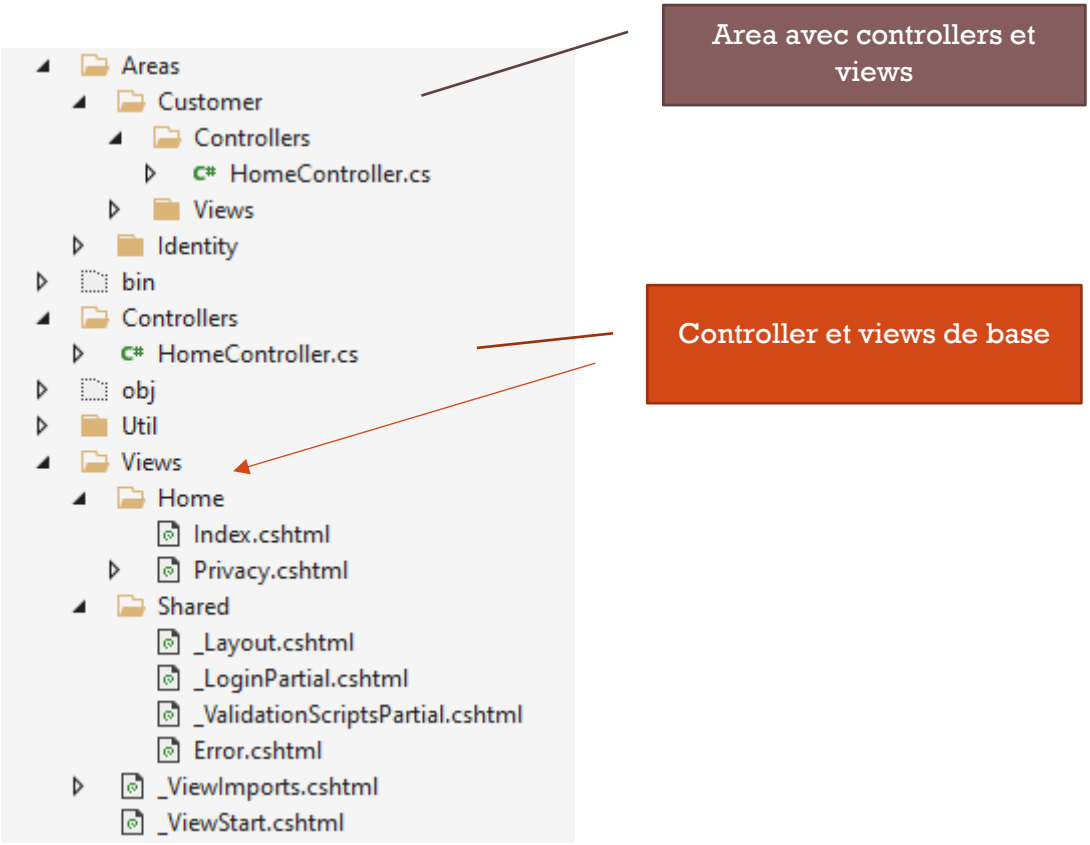
Par défaut elle pointe directement _Layout

```
@{
    Layout = "_Layout";
}
```

Areas avec Routes principales

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(name: "areas", pattern:
"{area:exists}/{controller=Home}/{action=Index}/{id?}");

    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");

    endpoints.MapRazorPages();
});
```

Area en premier



Area « Admin »

On peut créer une area admin pour tout ce qu'il n'est accessible qu'aux users avec le role Admin.

```
[Area("Admin")]
[Authorize(Roles = "Admin")]
public class CategoryController : Controller
```

Création d'un projet Models avec les models et ViewModels

Cela peut être utile pour centraliser les models et viewmodels plutôt qu'ils soient perdus entre les différentes Areas, etc.

Repository et IUnitOfWork

Classe de base

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace AspNetCoreMvcSample.Data.Repository
{
   public interface IRepository<T> where T : class
   {
      void Add(T entity);
      T Get(int id);
      IEnumerable<T> GetAll(Expression<Func<T, bool>> filter = null, Func<IQueryable<T>, IOrderedQueryable<T>>
orderBy = null, string includeProperties = null);
      T GetFirstOrDefault(Expression<Func<T, bool>> filter = null, string includeProperties = null);
      void Remove(int id);
      void Remove(T entity);
```

```
    void RemoveRange(IEnumerable<T> entity);
  }

  public class Repository<T> : IRepository<T> where T : class
  {

    private readonly ApplicationDbContext context;
    internal DbSet<T> dbSet;

    public Repository(ApplicationDbContext context)
    {
      this.context = context;
      this.dbSet = this.context.Set<T>();
    }

    public void Add(T entity)
    {
      dbSet.Add(entity);
    }

    public T Get(int id)
    {
      return dbSet.Find(id);
    }

    public IEnumerable<T> GetAll(Expression<Func<T, bool>> filter = null, Func<IQueryable<T>,
IOrderedQueryable<T>> orderBy = null, string includeProperties = null)
    {
      IQueryable<T> query = dbSet;
      if (filter != null)
        query = query.Where(filter);

      if (includeProperties != null)
      {
        foreach (var includeProp in includeProperties.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))
        {
          query = query.Include(includeProp);
        }
      }

      if (orderBy != null)
        return orderBy(query).ToList();

      return query.ToList();
    }

    public T GetFirstOrDefault(Expression<Func<T, bool>> filter = null, string includeProperties = null)
    {
      IQueryable<T> query = dbSet;
      if (filter != null)
        query = query.Where(filter);

      if (includeProperties != null)
      {
        foreach (var includeProp in includeProperties.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))
        {
          query = query.Include(includeProp);
        }
      }

      return query.FirstOrDefault();
    }

    public void Remove(int id)
    {
      T entity = dbSet.Find(id);
      Remove(entity);
    }

    public void Remove(T entity)
    {
      dbSet.Remove(entity);
    }
```

```
        public void RemoveRange(IEnumerable<T> entity)
        {
            dbSet.RemoveRange(entity);
        }
    }
}
```

Version Async

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;

namespace AspNetCoreMvcSample.Data.Repository
{
    public interface IRepositoryAsync<T> where T : class
    {
        Task AddAsync(T entity);
        Task<IEnumerable<T>> GetAllAsync(Expression<Func<T, bool>> filter = null, Func<IQueryable<T>,
IOrderedQueryable<T>> orderBy = null, string includeProperties = null);
        Task<T> GetAsync(int id);
        Task<T> GetFirstOrDefaultAsync(Expression<Func<T, bool>> filter = null, string includeProperties = null);
        Task RemoveAsync(int id);
        Task RemoveAsync(T entity);
        Task RemoveRangeAsync(IEnumerable<T> entity);
    }

    public class RepositoryAsync<T> : IRepositoryAsync<T> where T : class
    {

        private readonly ApplicationDbContext _db;
        internal DbSet<T> dbSet;

        public RepositoryAsync(ApplicationDbContext db)
        {
            _db = db;
            this.dbSet = _db.Set<T>();
        }

        public async Task AddAsync(T entity)
        {
            await dbSet.AddAsync(entity);
        }

        public async Task<T> GetAsync(int id)
        {
            return await dbSet.FindAsync(id);
        }

        public async Task<IEnumerable<T>> GetAllAsync(Expression<Func<T, bool>> filter = null, Func<IQueryable<T>,
IOrderedQueryable<T>> orderBy = null, string includeProperties = null)
        {
            IQueryable<T> query = dbSet;

            if (filter != null)
                query = query.Where(filter);

            if (includeProperties != null)
            {
                foreach (var includeProp in includeProperties.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))
                {
                    query = query.Include(includeProp);
                }
            }

            if (orderBy != null)
                return await orderBy(query).ToListAsync();

            return await query.ToListAsync();
```

```
    }

    public async Task<T> GetFirstOrDefaultAsync(Expression<Func<T, bool>> filter = null, string includeProperties =
null)
    {
      IQueryable<T> query = dbSet;
      if (filter != null)
        query = query.Where(filter);

      if (includeProperties != null)
      {
        foreach (var includeProp in includeProperties.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))
        {
          query = query.Include(includeProp);
        }
      }

      return await query.FirstOrDefaultAsync();
    }

    public async Task RemoveAsync(int id)
    {
      T entity = await dbSet.FindAsync(id);
      await RemoveAsync(entity);
    }

    public async Task RemoveAsync(T entity)
    {
      dbSet.Remove(entity);
    }

    public async Task RemoveRangeAsync(IEnumerable<T> entity)
    {
      dbSet.RemoveRange(entity);
    }
  }
}
```

Création de Reposiotry « typé »

Exemple avec CategoryRepository :

- Hérite de Repository
- Implémente une interface sans générique pour facilement l'enregistrer en dépendance

```
public interface ICategoryRepository: IRepositoryAsync<Category>
{
  void Update(Category category);
}

public class CategoryRepository : RepositoryAsync<Category>, ICategoryRepository
{
  private readonly ApplicationDbContext context;

  public CategoryRepository(ApplicationDbContext context) : base(context)
  {
    this.context = context;
  }

  public void Update(Category category)
  {
    var categoryToUpdate = context.Categories.FirstOrDefault(x => x.Id ==
category.Id);
```

```
        if (categoryToUpdate != null)
        {
            categoryToUpdate.Name = category.Name;
        }
    }
}
```

UnitOfWork

A :

- Une liste de tous les repositories ne properties
- Une méthode Save pour valider tous les changements apportés d'un coup

```
public interface IUnitOfWork : IDisposable
{
    ICategoryRepository Category { get; }
    IProductRepository Product { get; }

    void Save();
}

public class UnitOfWork : IUnitOfWork
{
    private readonly ApplicationDbContext context;

    public UnitOfWork(ApplicationDbContext context)
    {
        this.context = context;
        Category = new CategoryRepository(this.context);
        Product = new ProductRepository(this.context);
    }

    public ICategoryRepository Category { get; private set; }
    public IProductRepository Product { get; private set; }

    public void Save()
    {
        context.SaveChanges();
    }

    public void Dispose()
    {
        context.Dispose();
    }
}
```

On injectera que IUnitOfWork dans les controllers donc on n'a qu'a enregistrer celui-ci en dépendance (pas besoin d'enregistrer les repositories) dans Startup.cs

```
services.AddScoped<IUnitOfWork, UnitOfWork>();
```

Custom TagHelpers

Exemple pour le paging

```
using AspNetCoreMvcSample.Models.ViewModels;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace AspNetCoreMvcSample.TagHelpers
{
   [HtmlTargetElement("div", Attributes = "page-model")]
   public class PageLinkTagHelper : TagHelper
   {
      [ViewContext]
      [HtmlAttributeNotBound]
      public ViewContext ViewContext { get; set; }

      public PagingInfo PageModel { get; set; }

      public string PageAction { get; set; }
      public bool PageClassesEnabled { get; set; }
      public string PageClass { get; set; }
      public string PageClassNormal { get; set; }
      public string PageClassSelected { get; set; }

      public override void Process(TagHelperContext context, TagHelperOutput output)
      {
         TagBuilder result = new TagBuilder("div");

         for (int i = 1; i <= PageModel.TotalPage; i++)
         {
            TagBuilder tag = new TagBuilder("a");
            string url = PageModel.urlParam.Replace(":", i.ToString());
            tag.Attributes["href"] = url;
            if (PageClassesEnabled)
            {
               tag.AddCssClass(PageClass);
               tag.AddCssClass(i == PageModel.CurrentPage ? PageClassSelected :
PageClassNormal);
            }
            tag.InnerHtml.Append(i.ToString());
            result.InnerHtml.AppendHtml(tag);
         }

         output.Content.AppendHtml(result.InnerHtml);
      }
   }
}
```

Dans ViewImports

```
@using AspNetCoreMvcSample
@using AspNetCoreMvcSample.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper AspNetCoreMvcSample.TagHelpers.*, AspNetCoreMvcSample
```

Shopping Cart

Session

Enregistrement Startup.cs

```
services.AddSession(options =>
{
   options.IdleTimeout = TimeSpan.FromMinutes(30);
   options.Cookie.HttpOnly = true;
   options.Cookie.IsEssential = true;
});
```

Configure

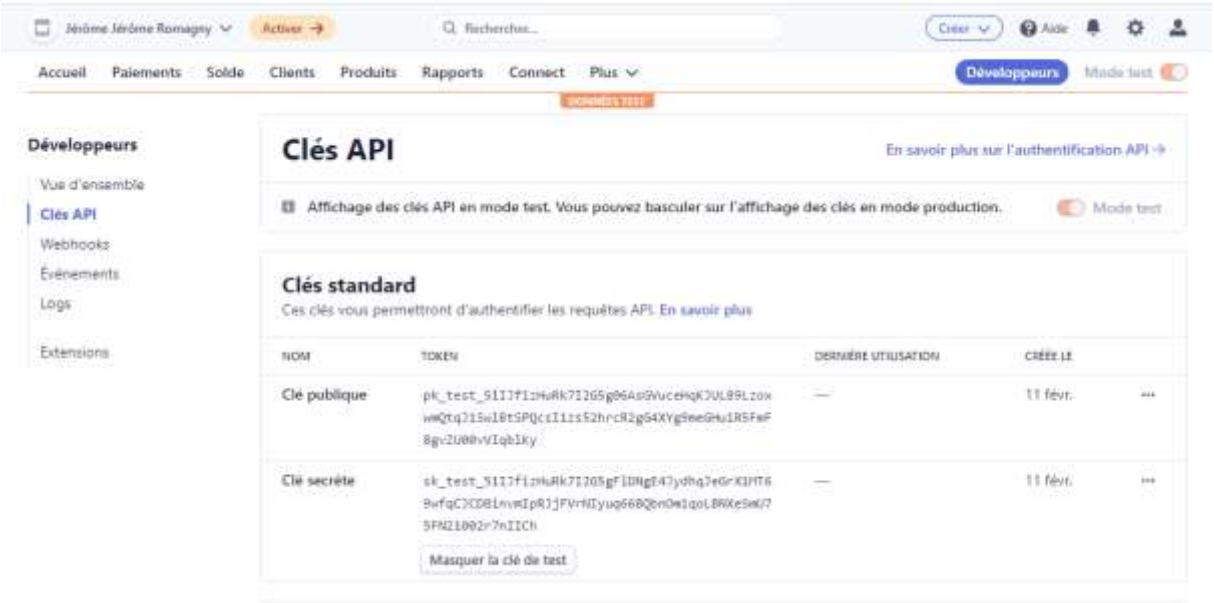
```
app.UseSession();
```

Extension

```
using Microsoft.AspNetCore.Http;
using Newtonsoft.Json;

namespace AspNetCoreMvcSample.Util
{
   public static class SessionExtension
   {
      public static void SetObject(this ISession session, string key, object value)
      {
         session.SetString(key, JsonConvert.SerializeObject(value));
      }
      public static T GetObject<T>(this ISession session, string key)
      {
         var value = session.GetString(key);
         return value == null ? default(T) : JsonConvert.DeserializeObject<T>(value);
      }
   }
}
```

Stripe

Création de compte , puis dans l'onglet developpeurs, récupérer les clés publiques et secrètes



Ajout à appsettings.json

```
  "Stripe": {
   "SecretKey":
"sk_test_51IJfizHuRk7I2G5gFlDNgE4JydhqJeGrX1MT55wfqCJCD8invmIpRJjFVrNIyuq66
BQbnOm1qoLBNXeSmU75FN21002r7nIICh",
   "PublicKey":
"pk_test_51IJfizHuRk7I2G5g06AsGVuceHqKJUL89LzuyiwmQtqJ1SwlBtSPQcsI1zs52hrcR2
gG4XYg9meGHu1R5FmF8gvZU00vVIqblKy"
  }
```

Création d'une classe StripeSettings

```
   public class StripeSettings
   {
      public string SecretKey { get; set; }
      public string PublicKey { get; set; }
   }
```

Et enregistrement

```
services.Configure<StripeSettings>(Configuration.GetSection("Stripe"));
```

Installation du package

```
Install-Package Stripe.net
```

Ou packageReference

```
<PackageReference Include="Stripe.net" Version="39.76.0" />
```