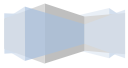
.Net 8 JSON Web Authentication

Step by step



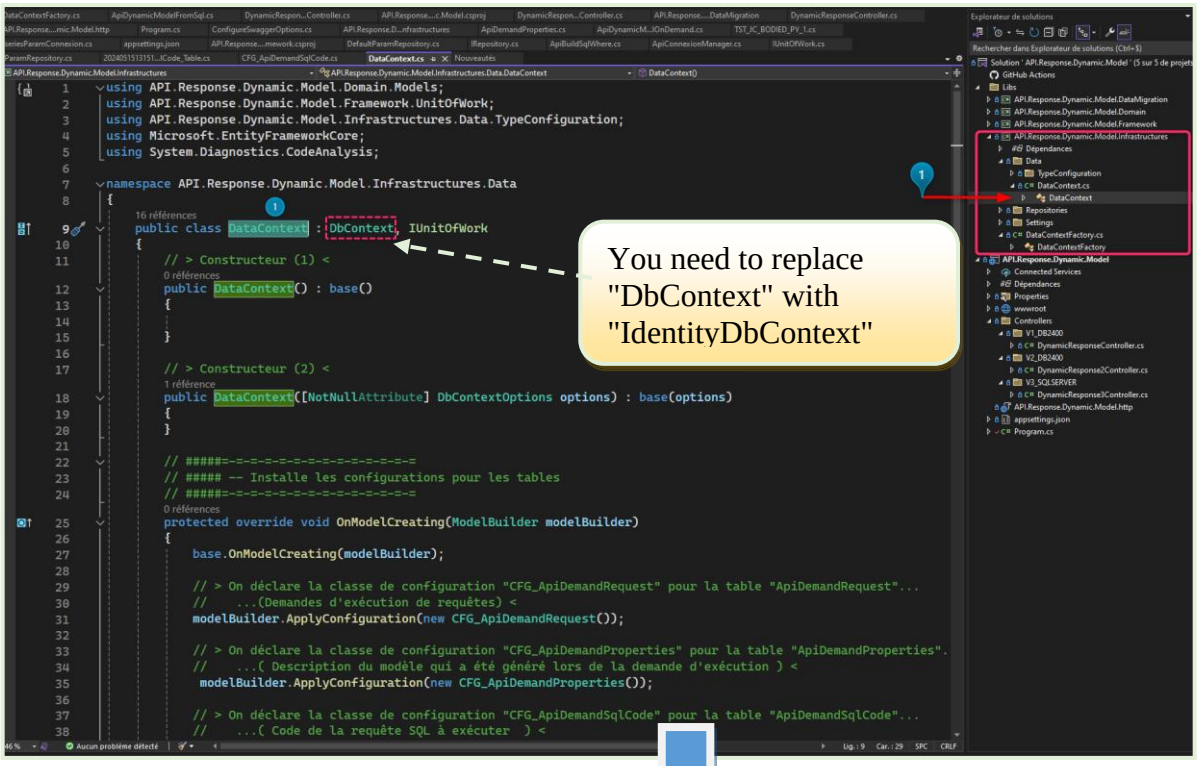
Summary

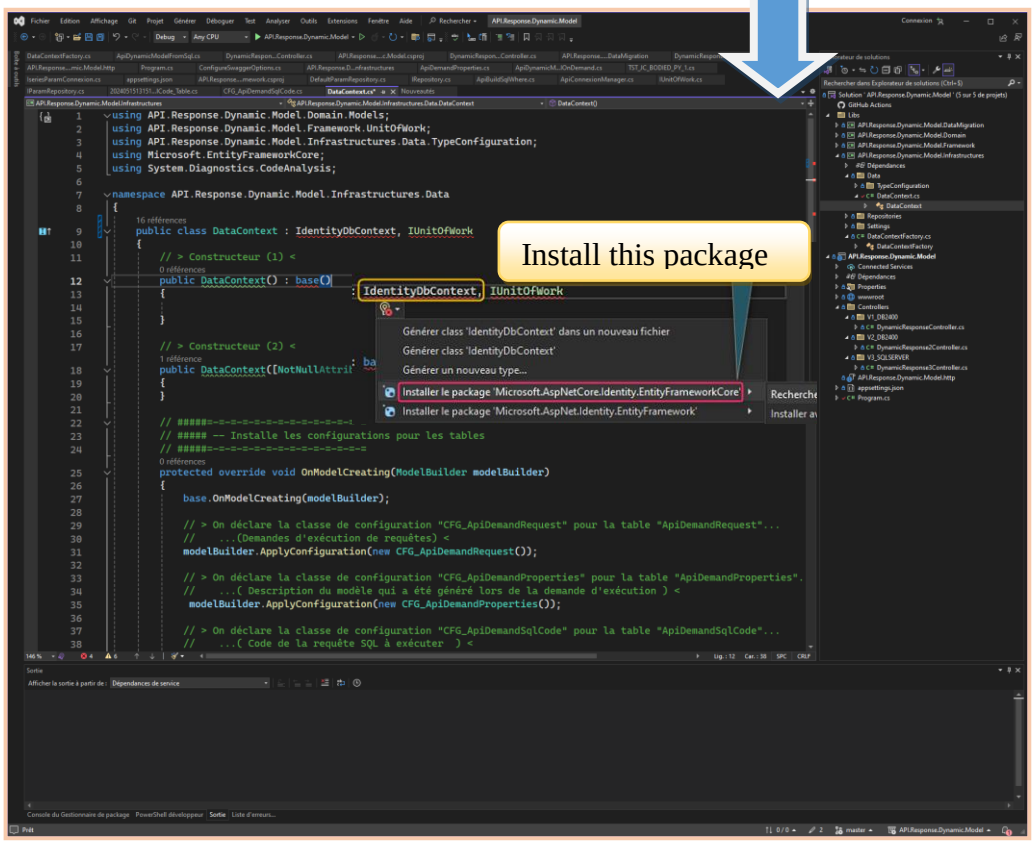
I – Introduction

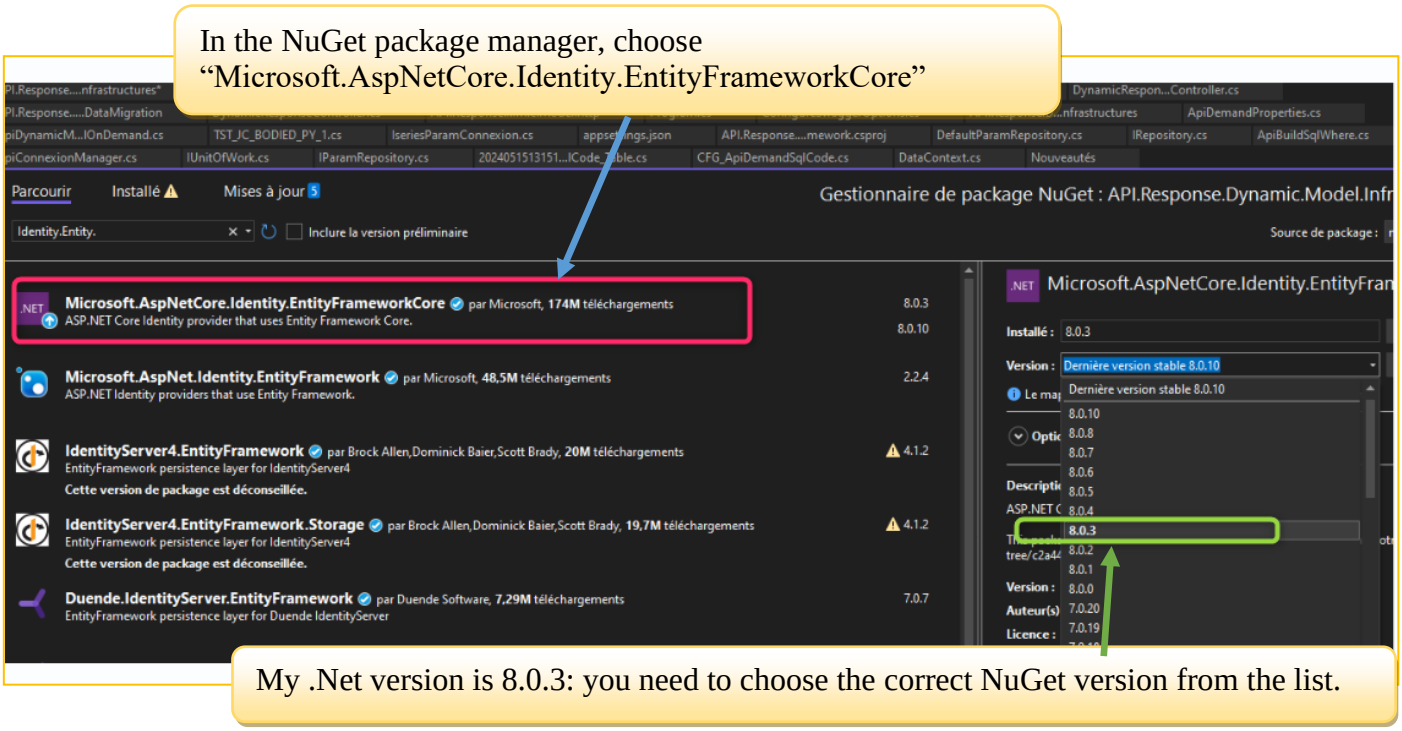
- Summary of JWT Authentication implementation

II – Update the Context: replace “DbContext” with “IdentityDbContext”

II-A Installing the NuGet “Microsoft.AspNetCore.Identity.EntityFrameworkCore”

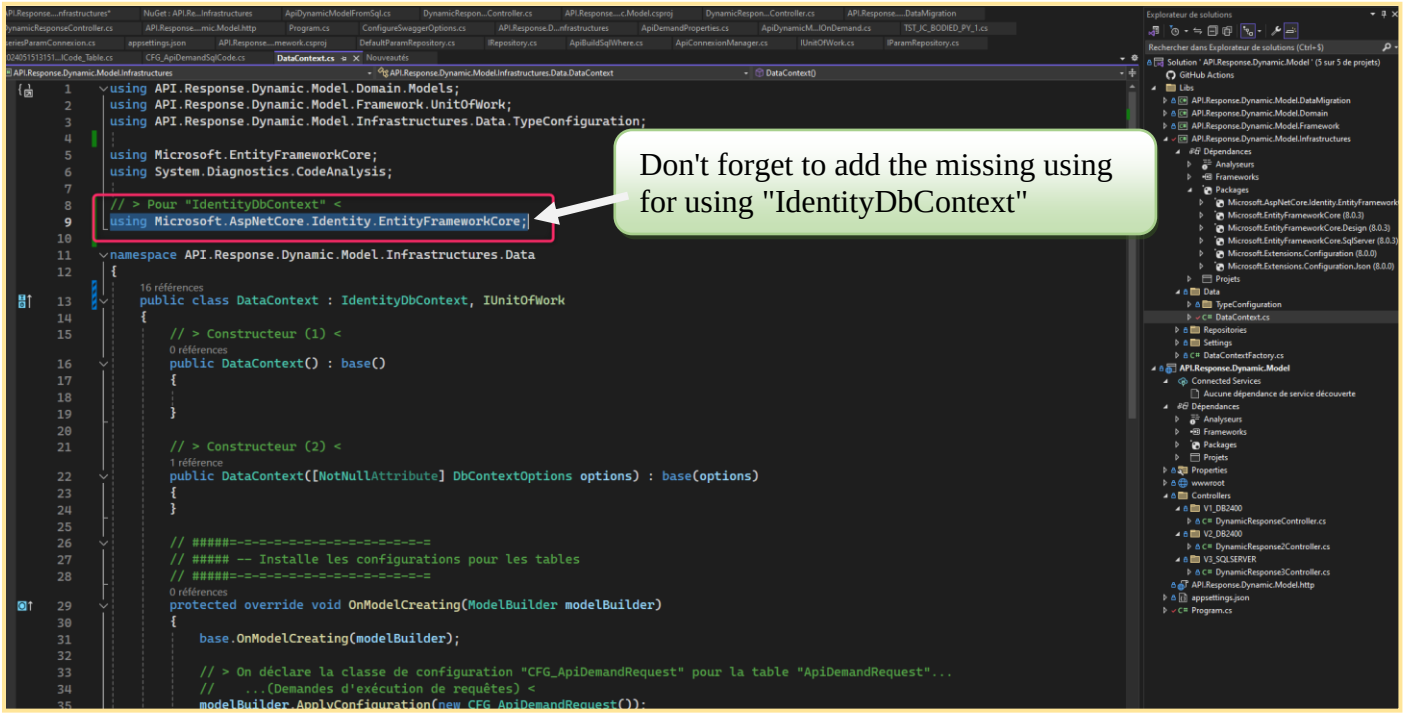




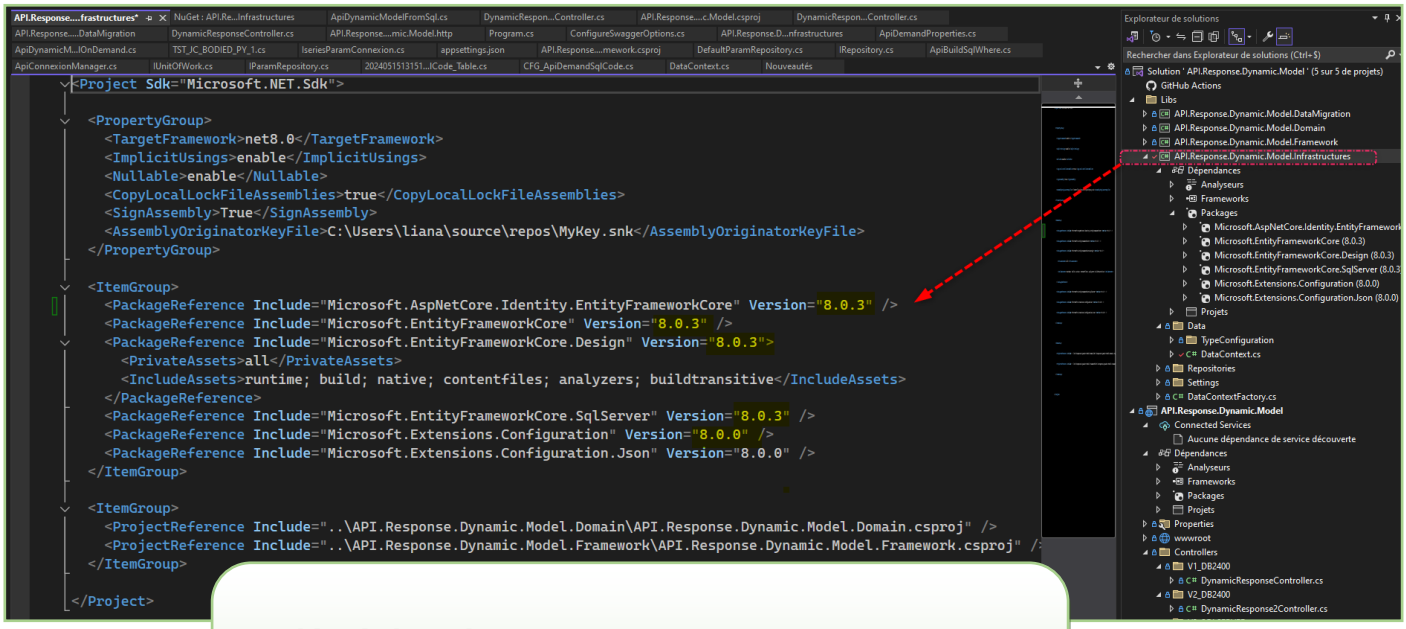


II-A-1 Added “using Microsoft.AspNetCore.Identity.EntityFrameworkCore”

- In the “**DataContext** ” class, add the missing “using ”.



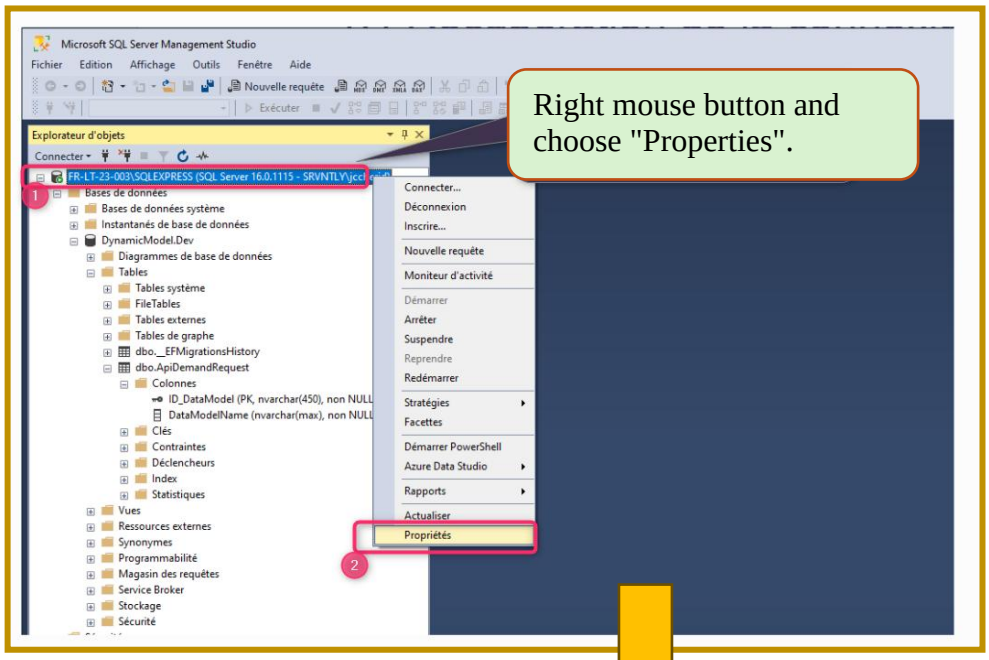
II-B Reminder: How to find out the .Net version of the solution

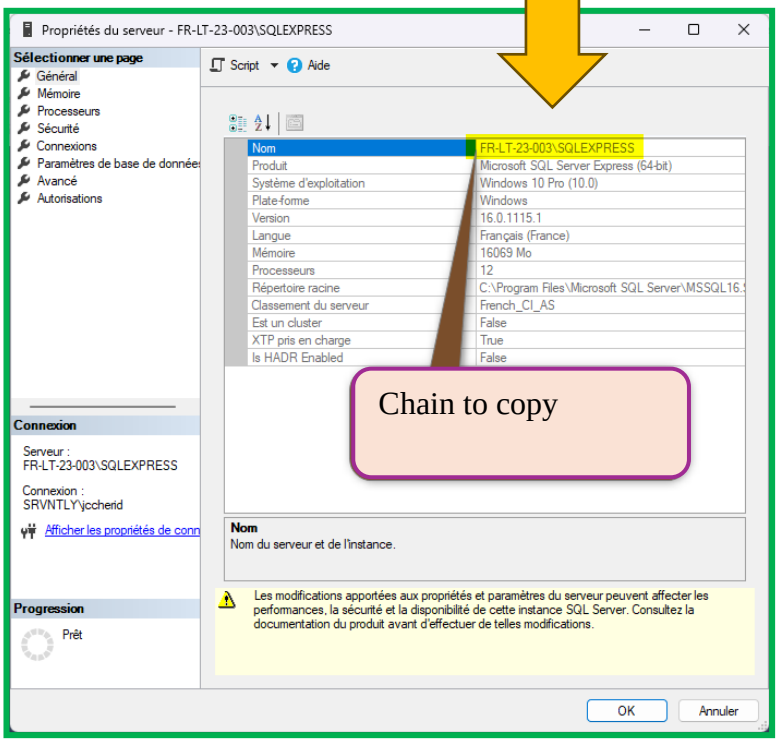


Double-click on the "API.Response.Dynamic.Model.Infrastructures" project: you will see the **current version** of **.Net** used by the project.

III Start creating the database

III – A SQL Server - Retrieve connection information (connection string)

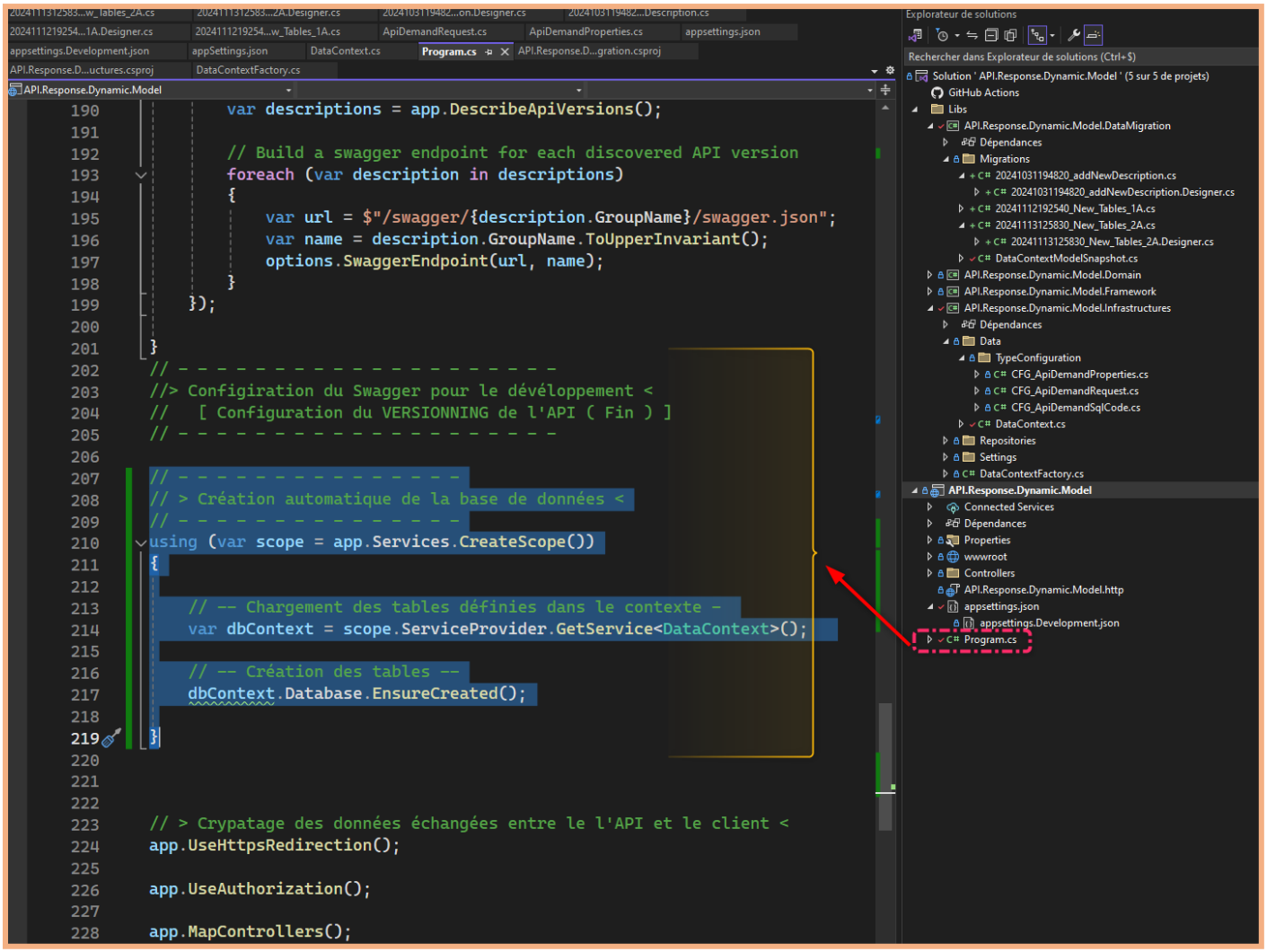




III – B SQL SERVER - Program.cs ➔ Start database creation

✸ In the "**Program.cs**" class, the code below will initiate database creation.

Note: The database will only be created if it does NOT already exist.




```
// - - - - - - - - - - - - - - - -
// > Automatic database creation <
// - - - - - - - - - - - - - - - -
using (var scope = app.Services.CreateScope())
{
   // -- Loading tables defined in the context-
   var dbContext = scope.ServiceProvider.GetService<DataContext>();

   // -- Creating tables--
   dbContext.Database.EnsureCreated();
}
```

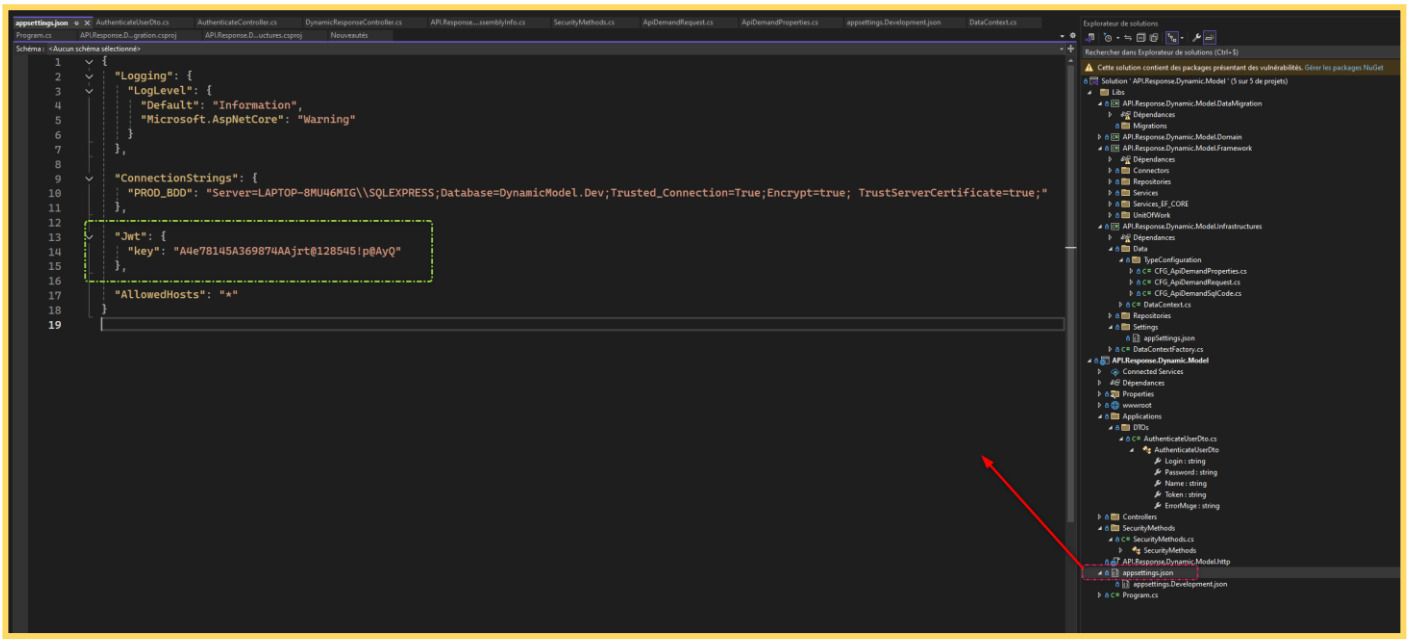
IV Adding JWT rules

IV-A "appsettings.json" - Added key for Token calculation

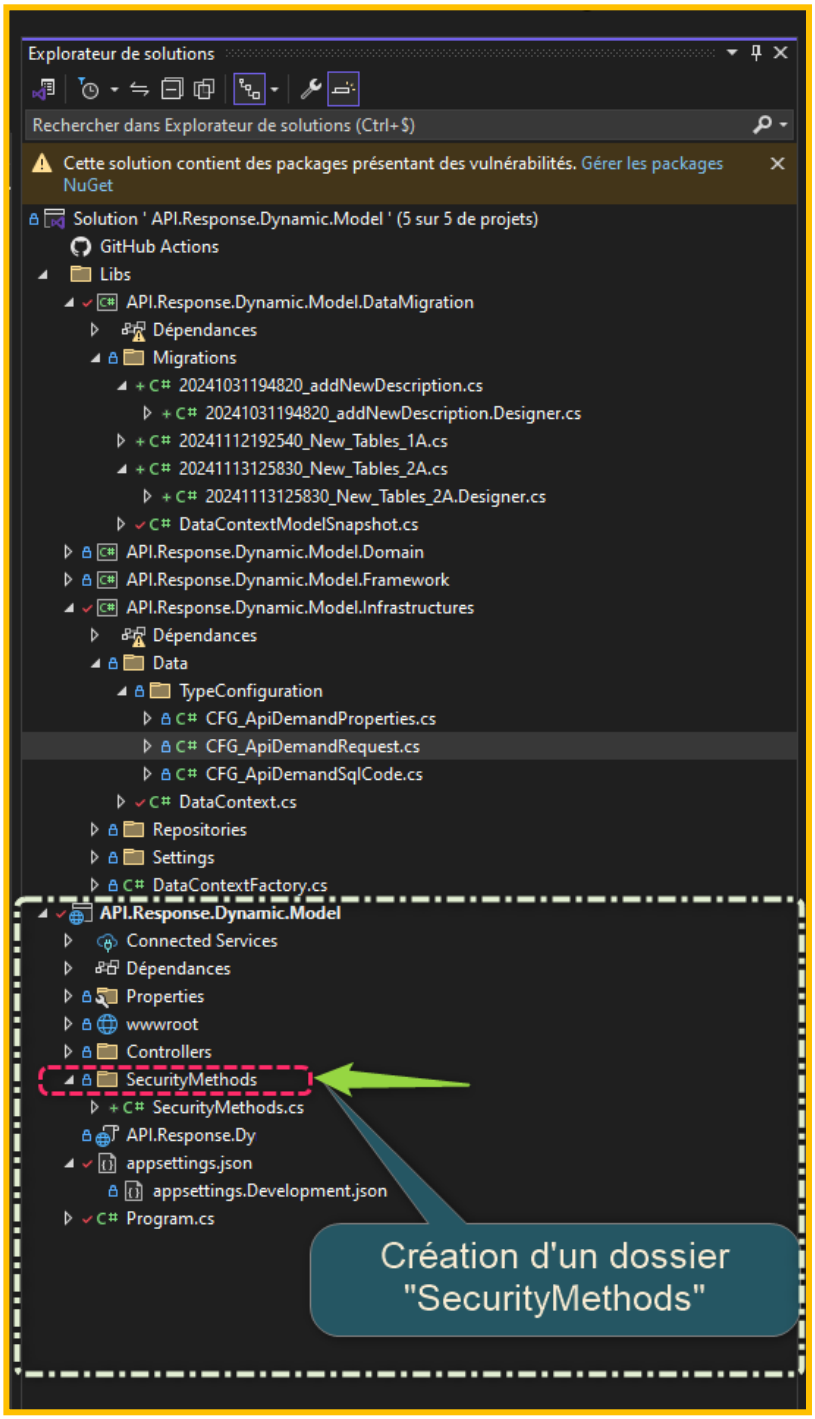
🚀 You must declare the key that will be used for the token calculation algorithm in the appsettings.json file.

👉 Below is an example of a key:

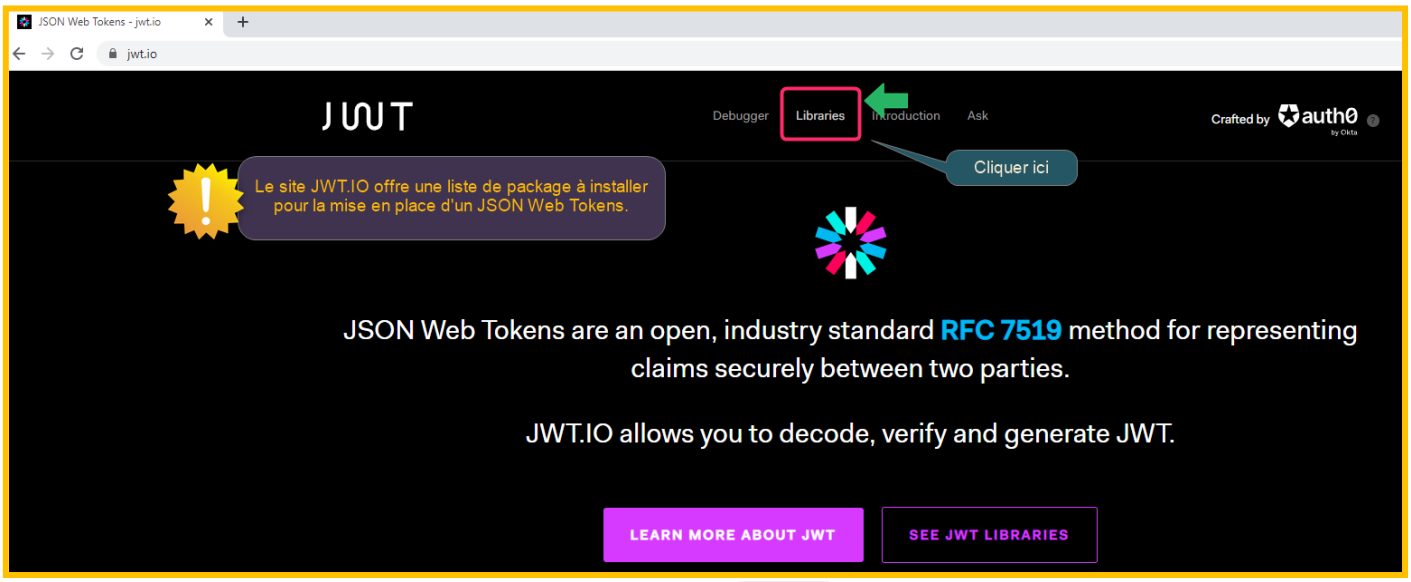
```
"Jwt": {
  "key": "A4e78145A369874AAjrt@128545!p@AyQ"
},
```

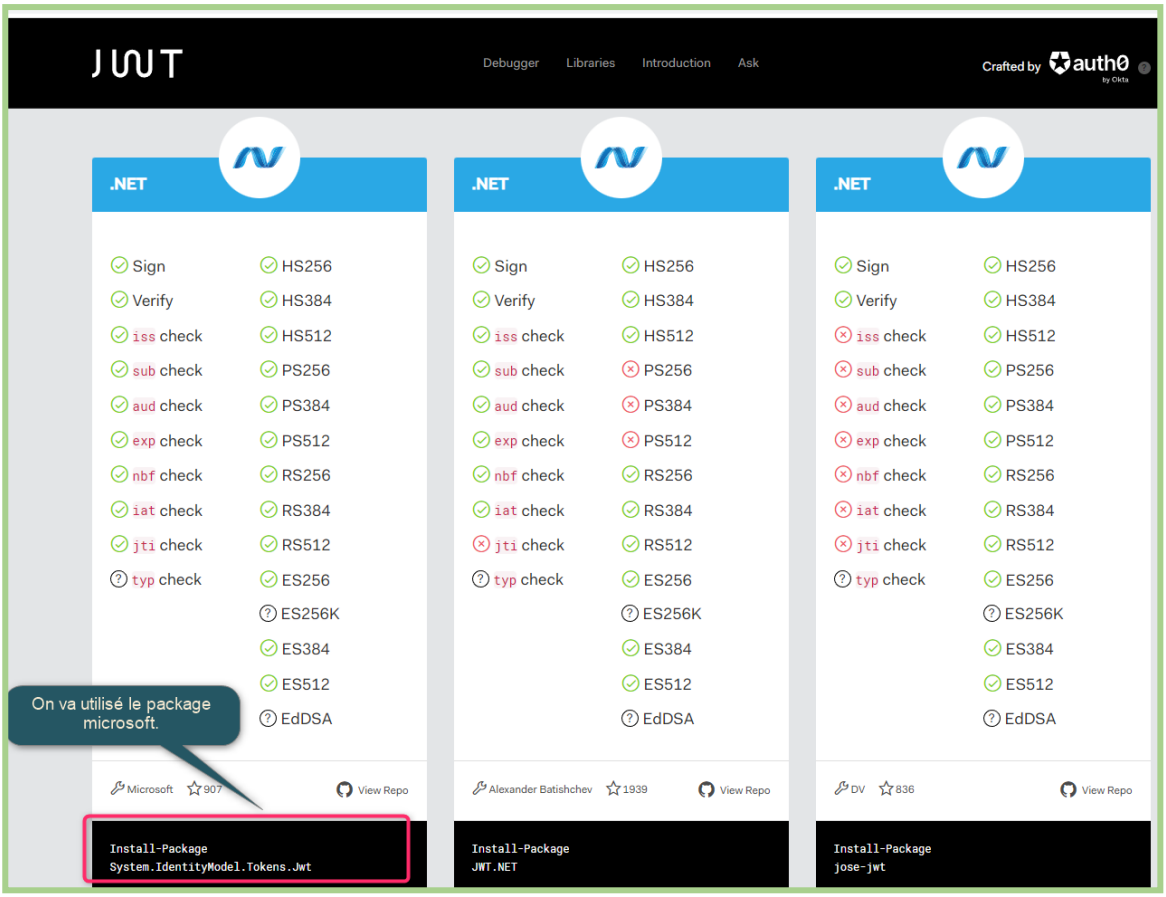


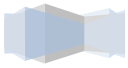
IV-B Create a "SecurityMethods" folder



IV-C Install the "System.IdentityModel.Token.JWT" package

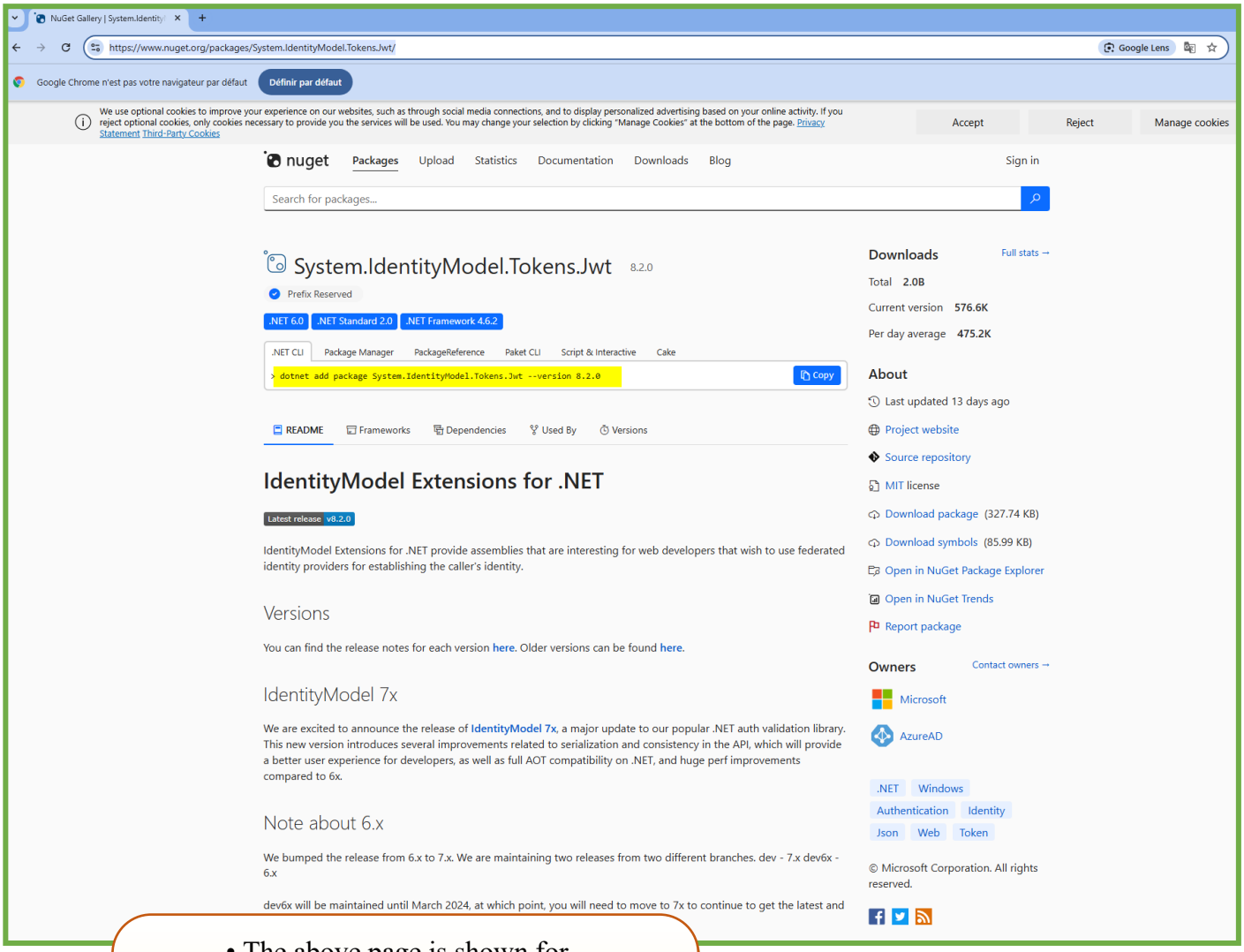




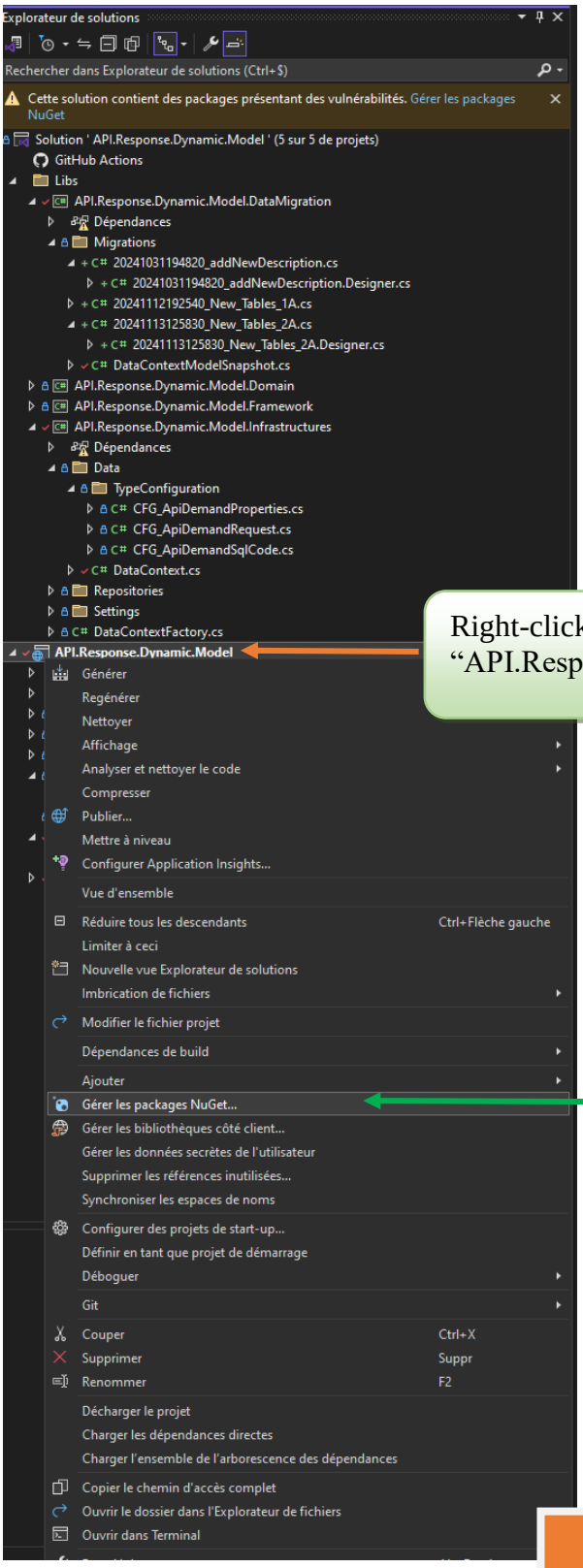


✨When you click on "**Install-Package System.IdentityModel.Tokens.Jwt**", you arrive on the page below:

✅ Link : https://www.nuget.org/packages/System.IdentityModel.Tokens.Jwt/

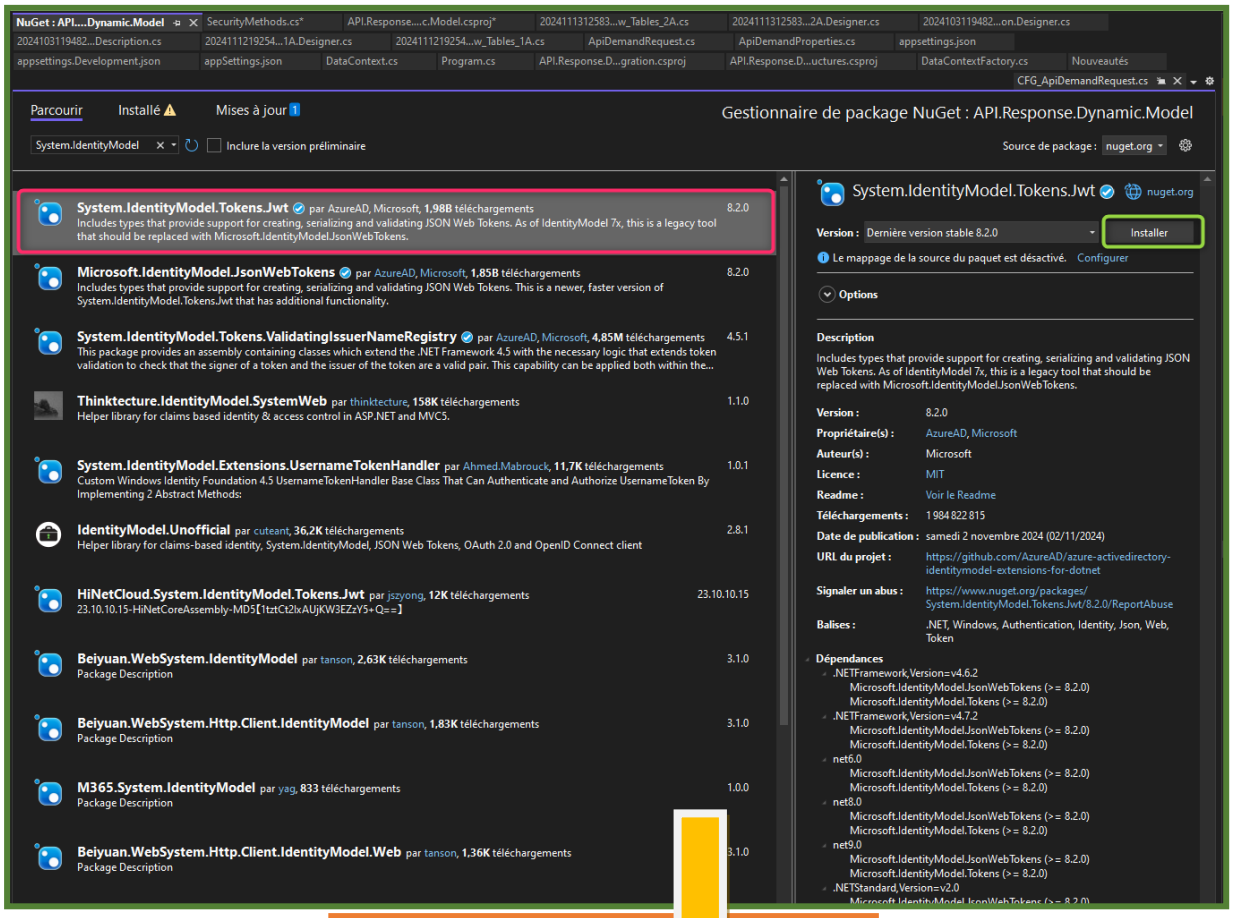


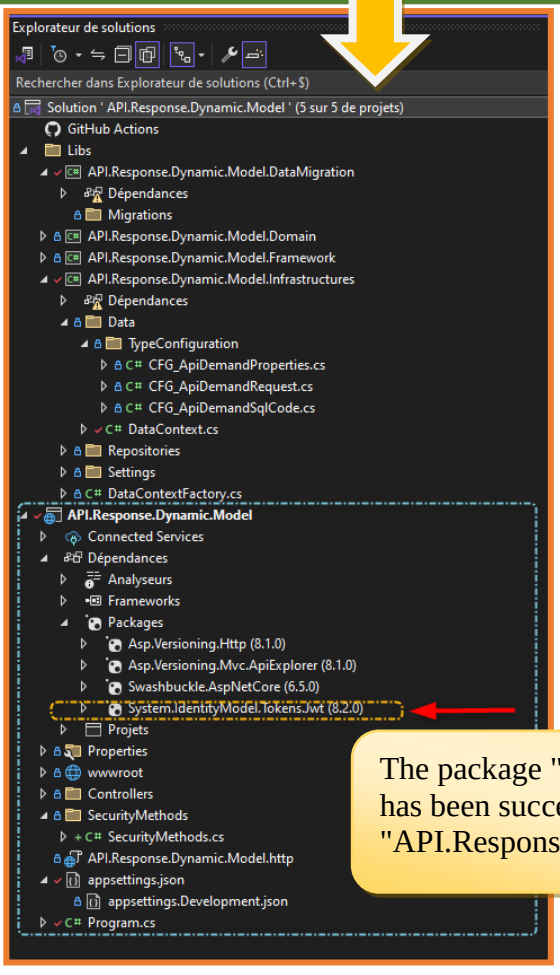
- The above page is shown for informational purposes.
- We will not use this method to install our package.



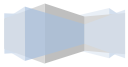
Right-click on the "API.Response.Dynamic.Model" project

Then choose "Manage NuGet packages…"

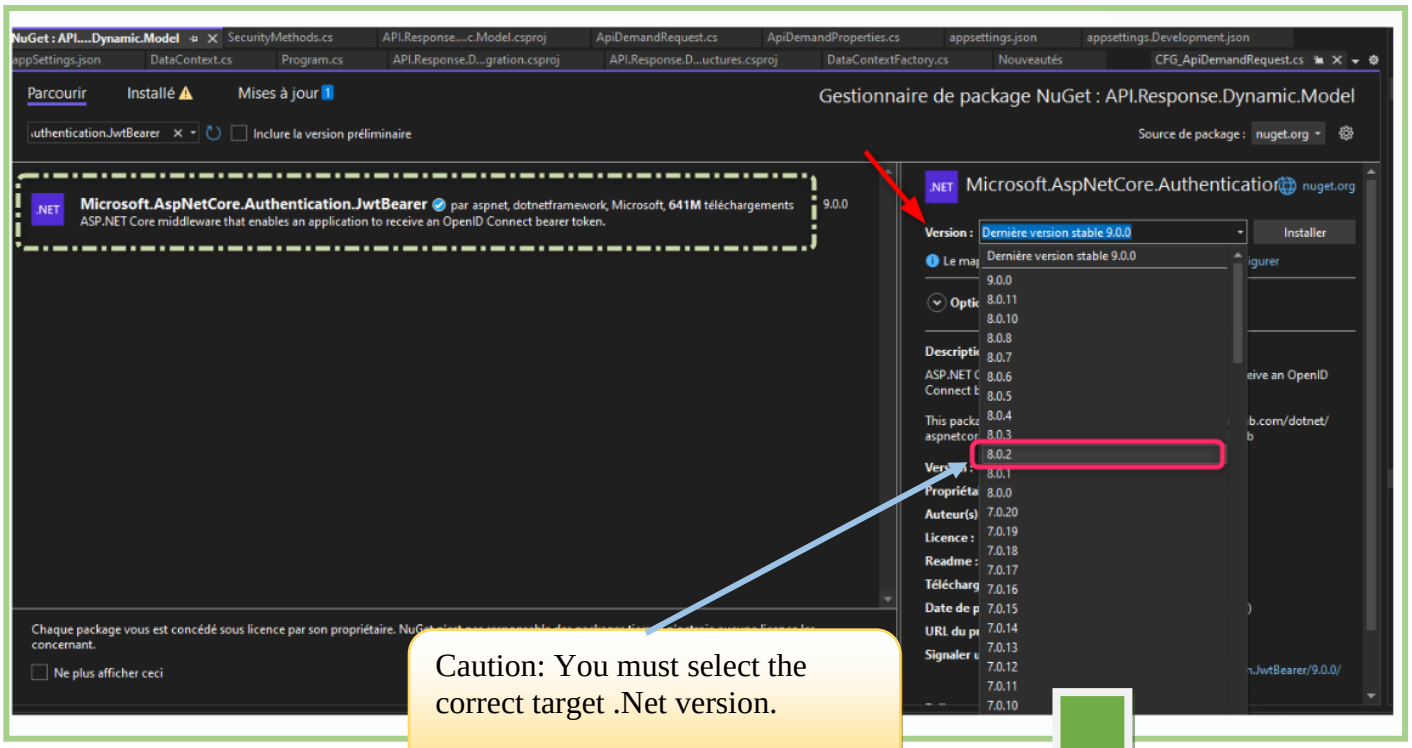


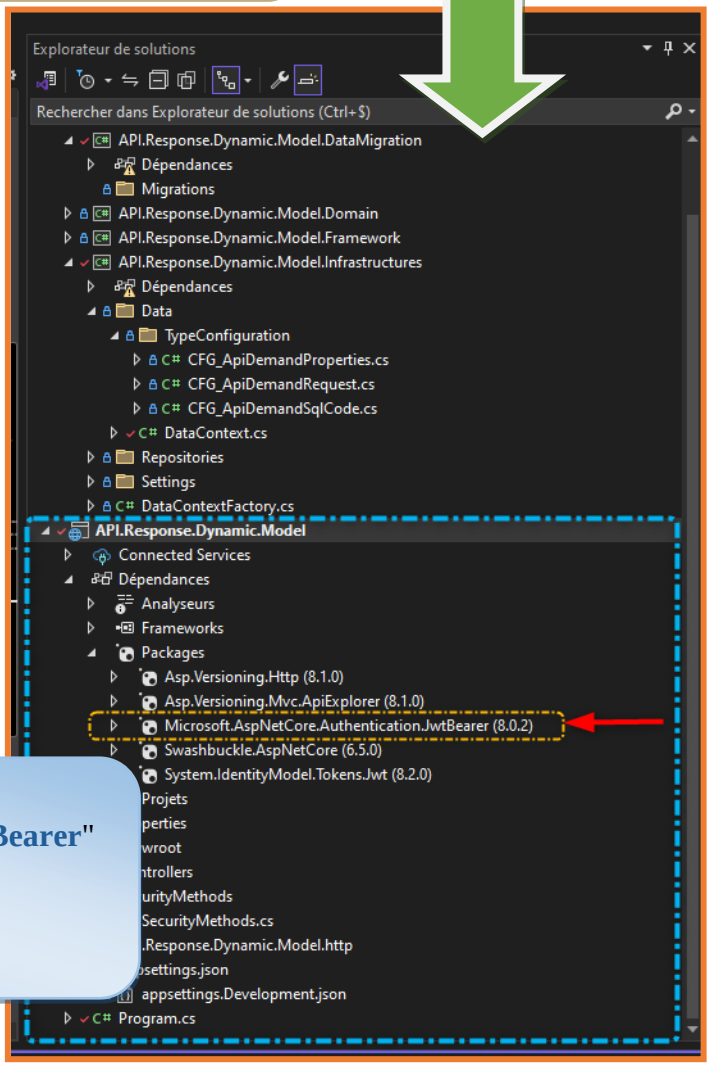


The package "System.IdentityModel.Token.Jwt" has been successfully added to the project "API.Response.Dynamic.Model"



IV-D Install the "Microsoft.AspNetCore.Authentication.JwtBearer" package

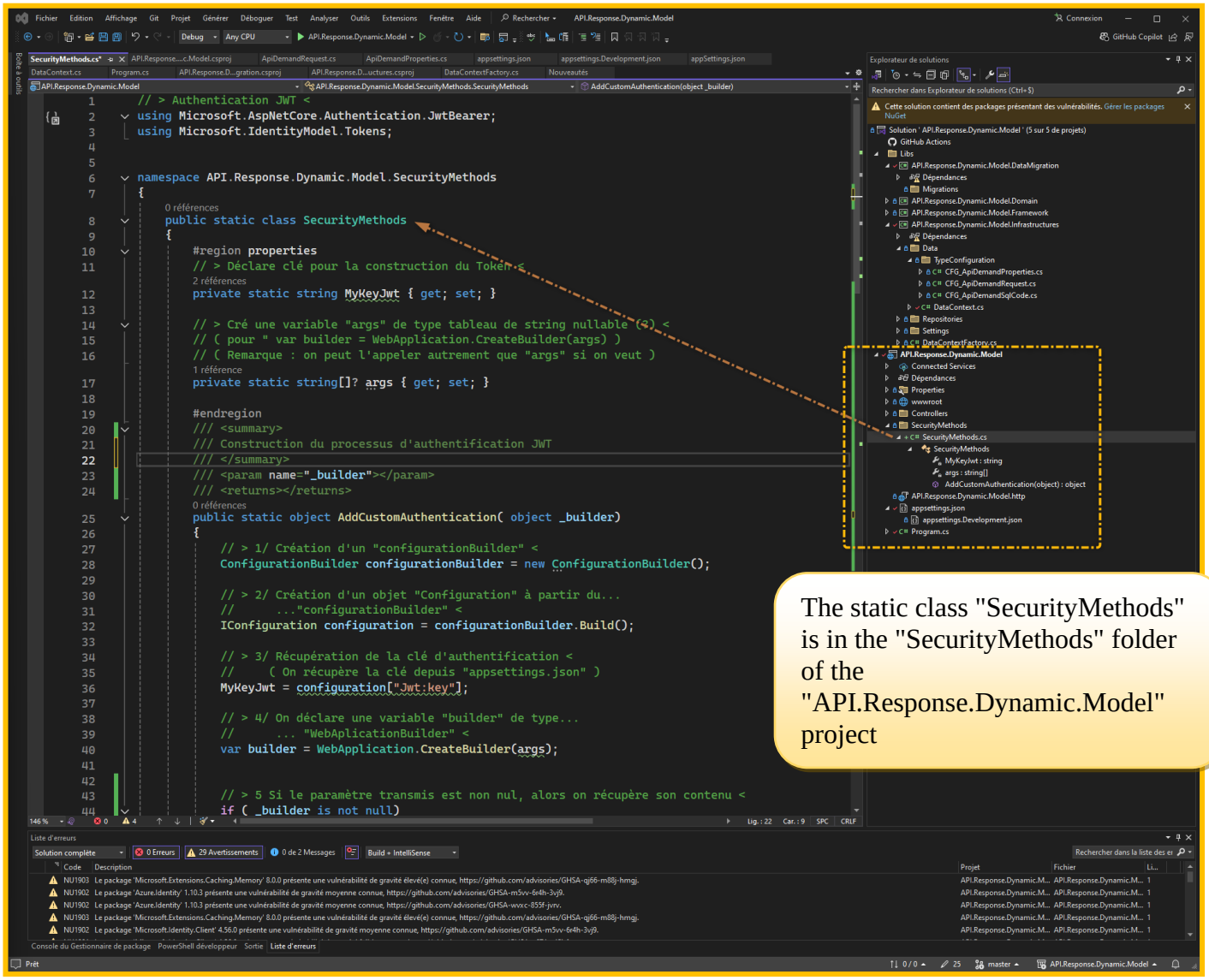




The NuGet "**Microsoft.AspNetCore.Authentication.JwtBearer**" is successfully added to the "API.Response.Dynamic.Model" project

IV-E Creation of the "SecurityMethods" class

IV-E-1 Location of the "SecurityMethods" class



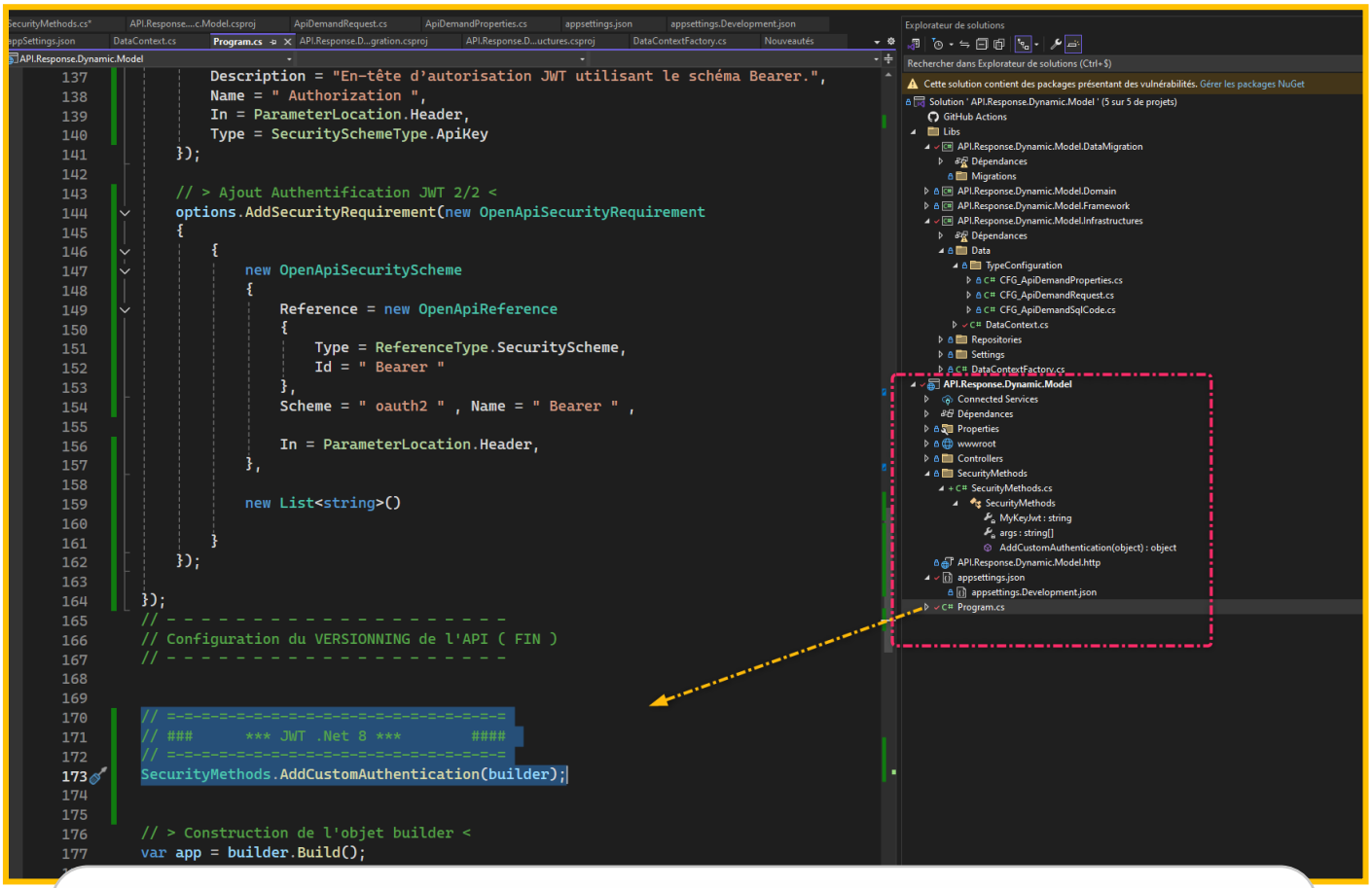
IV-E-2 Code of the “AddCustomAuthentication” method: This is where the key is used to calculate the Token

```
// > JWT Authentication <
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;

namespace API.Response.Dynamic.Model.SecurityMethods
{
   public static class SecurityMethods
   {
      #region properties
      // > Key statement for the construction of the Token <
      private static string MyKeyJwt { get; set; }

      private static string[]? args { get; set; }

      #endregion
      /// <summary>
      ///  Building the JWT Authentication Process
      /// </summary>
      /// <param name="_builder"></param>
      /// <returns></returns>
      public static object AddCustomAuthentication( object _builder)
      {
         // > 1/ Creating a "configurationBuilder" <
         ConfigurationBuilder configurationBuilder = new ConfigurationBuilder();

         // > 2/ Creating a "Configuration" object from the...
         //      ..."configurationBuilder" <
        IConfiguration configuration = configurationBuilder.Build();

         // > 3/ We declare a "builder" variable of type...
          // ... "WebAplicationBuilder" <
         var builder = WebApplication.CreateBuilder(args);

          // > 4 If the parameter passed is not null, then we retrieve its contents <
          if ( _builder is not null)
          {
              // > We parse "_builder" with the "WebApplicationBuilder" class <
              builder = (WebApplicationBuilder) _builder; ;
           }

        // > 5/ Retrieving the authentication key <
        //     ( We retrieve the key from "appsettings.json" )
       MyKeyJwt = string.Empty;
       MyKeyJwt = builder.Configuration.GetSection("Jwt")["key"];
```

```
        // =-=-=-=-=-=-=-=-=-=-=-
        // = = = = START Phase Validation of the <Json Web Token> =-=-=-=
        // =-=-=-=-=-=-=-=-=-=-=-
        //  > On détermine ici quelle est la façon de communiquer des Headers...
        //    ...dans la partie authentification.

        builder.Services.AddAuthentication(options =>
        {
           options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
           options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
           options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

        }).AddJwtBearer(options =>
        {
           // > This is where you configure jwt with all the parameters.<
            {
              // > We save the Token so we can use it later  <
              options.SaveToken  = true;

              // > Token Validation Parameters <
              options.TokenValidationParameters = new TokenValidationParameters()
              {
                 // > On va paramétrer notre Token avec la clé.
                 // > Celle ci  est encodée (=>"SymmetricSecurityKey") <
                 IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(MyKeyJwt)),

                 ValidateAudience = false,

                 ValidateIssuer = false,

                 ValidateActor = false,

                 // > The Token is said to have a limited lifespan.<
                 ValidateLifetime = true

              };

           }
        });

        // =-=-=-=-=-=-=-=-=-=-=-
        // =-=-=- END Phase Validation of the <Json Web Token> =-=-=-=
        // =-=-=-=-=-=-=-=-=-=-=-

        return builder;

     }
  }
}
```

IV-F Calling the "AddCustomAuthentication()" method in the "Program.cs" class

IV-F-1 The "Program.cs" class



✅ The "Program.cs" class belongs to the "API.Response.Dynamic.Model" project.

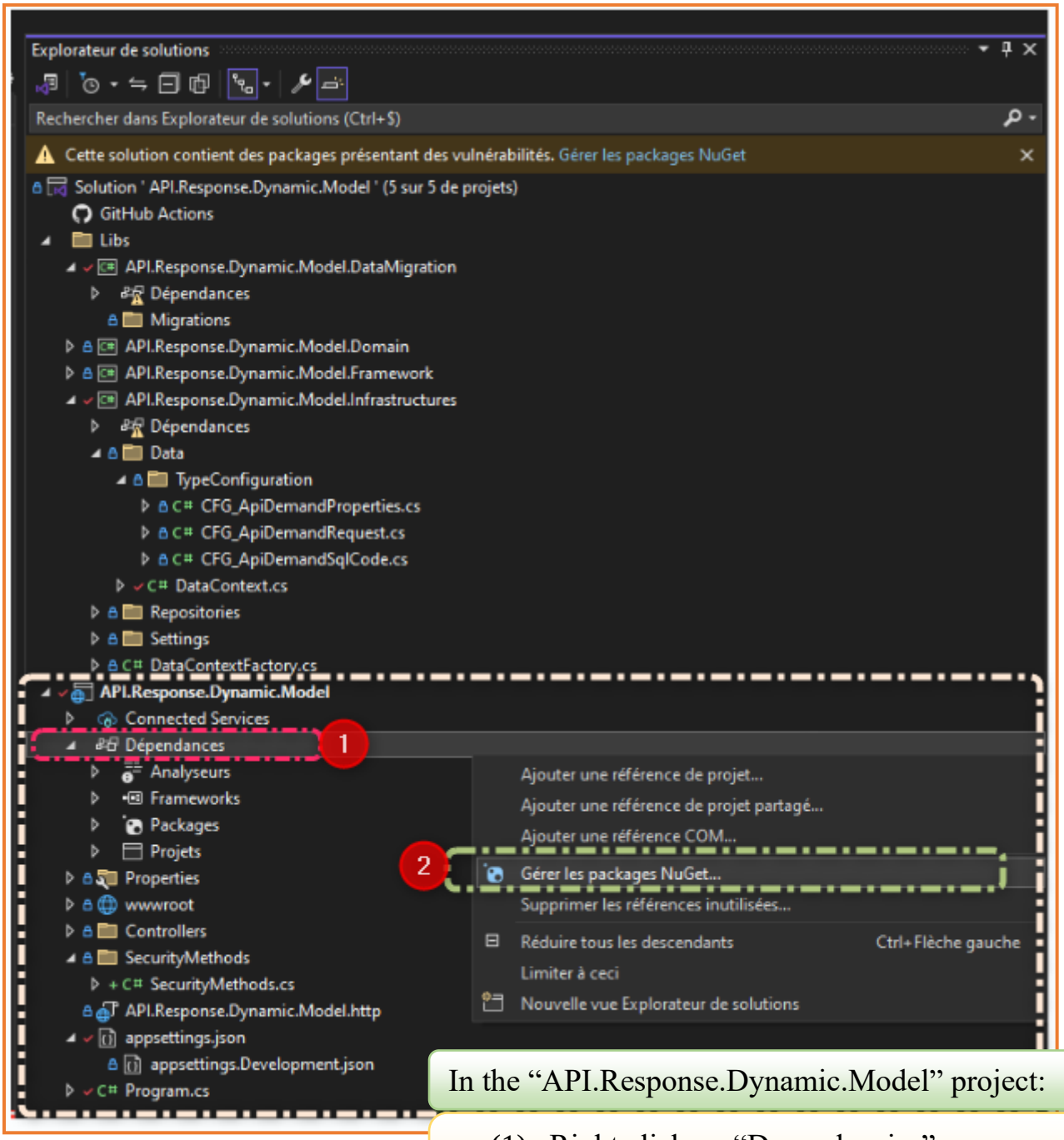
🚨 This is why it is possible to directly call the "AddCustomAuthentication" method of the static "SecuriryMethods" class.

IV-F-2 Line of code

```
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// ###     ***  JWT .Net 8 ***          ####
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
SecurityMethods.AddCustomAuthentication(builder);
```

V Configuring the "Context Identity"

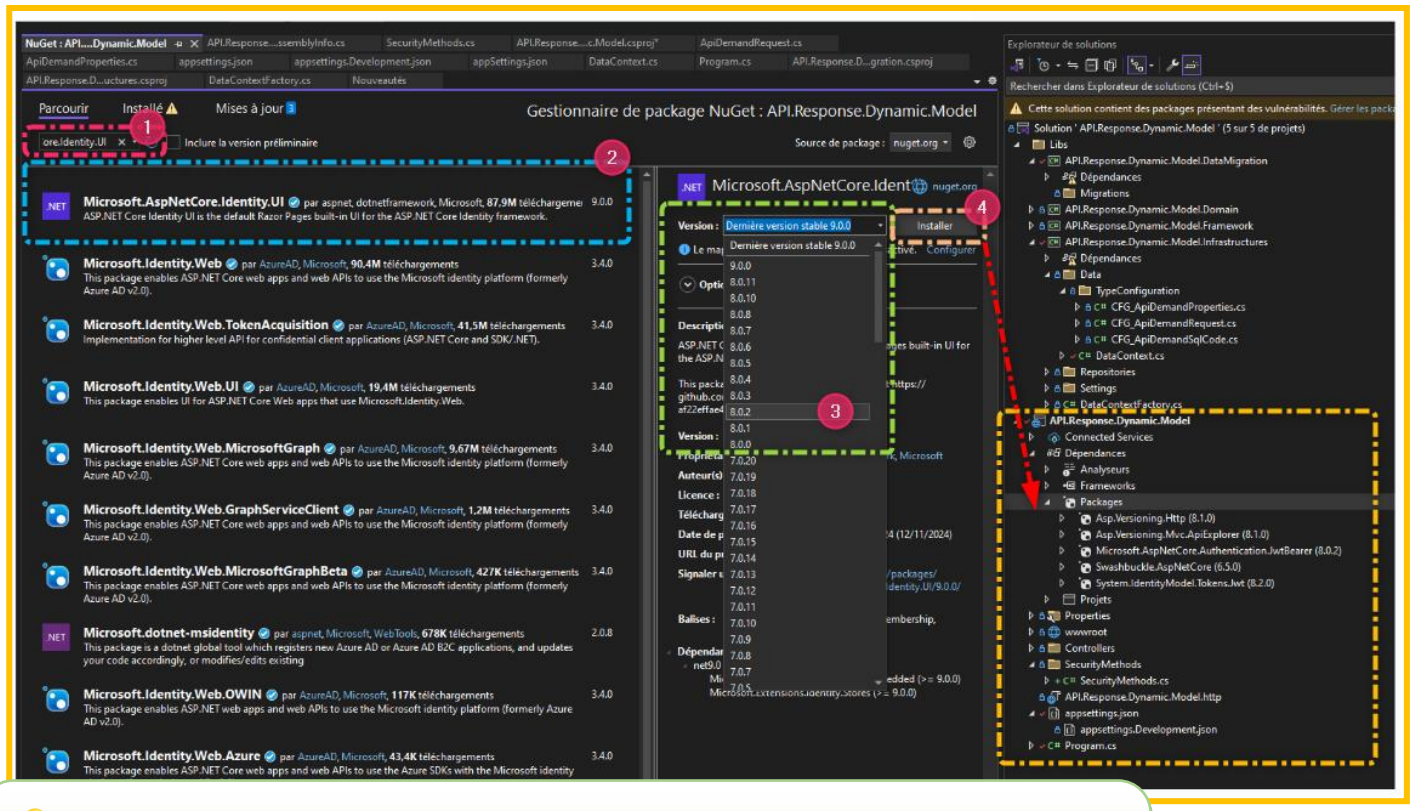
V-A Installing the NuGet "Microsoft.AspNetCore.Identity.UI"



In the "API.Response.Dynamic.Model" project:

(1)– Right click on "Dependencies"
(2)– Select "Manage NuGet Packages"

Step 3 ➔Don’t forget to choose the correct target version of .Net (the solution is in .Net 8.2)



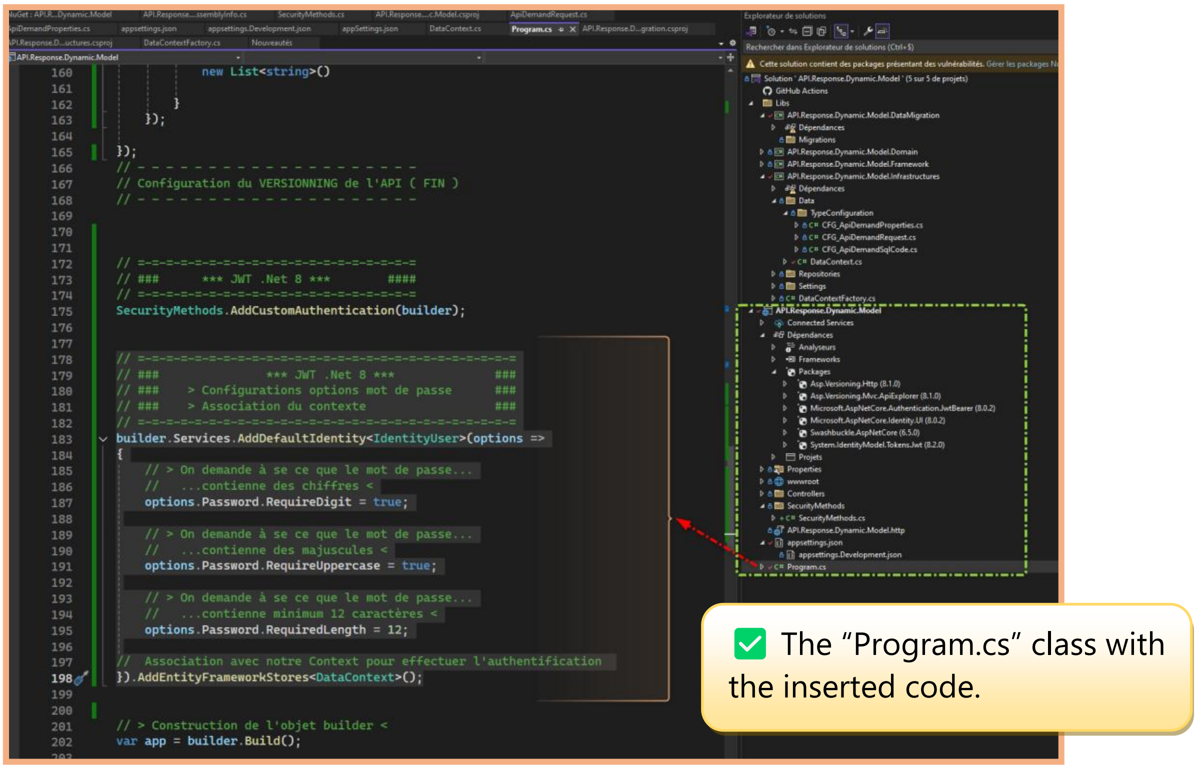
💡 The Nuget “Microsoft.AspNetCore.Identity.UI” is installed in the project “API.Response.Dynamic.Model”

V- B - (.Net 8) Implementation in the “Program.cs” class

V-B-1 Code to insert into the “Program.cs” class

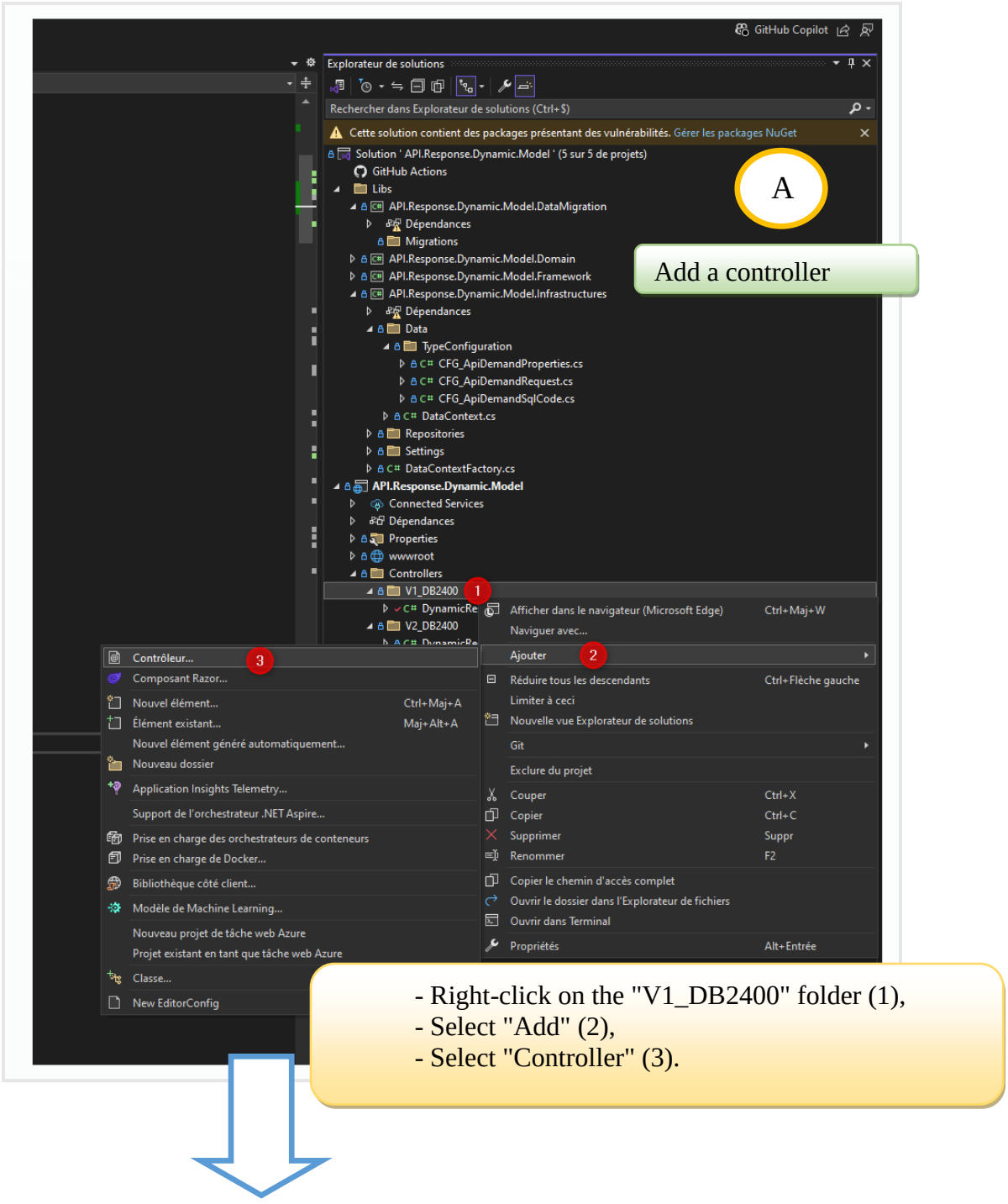
```
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// ###            *** JWT .Net 8 ***                     ###
// ###    > Password options settings                    ###
// ###    > Context association                          ###
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
{
   // > We ask what the password is...
 //    ...contains numbers <
   options.Password.RequireDigit = true;

   // > We ask what the password is...
   //    ...contains capital letters <
   options.Password.RequireUppercase = true;

   // > We ask what the password is...
   //    ..contains at least 12 characters <
   options.Password.RequiredLength = 12;

//  Association with our Context to perform authentication
}).AddEntityFrameworkStores<DataContext>();
```

V-B-2 Presentation of the class “Program.cs” with the code inserted



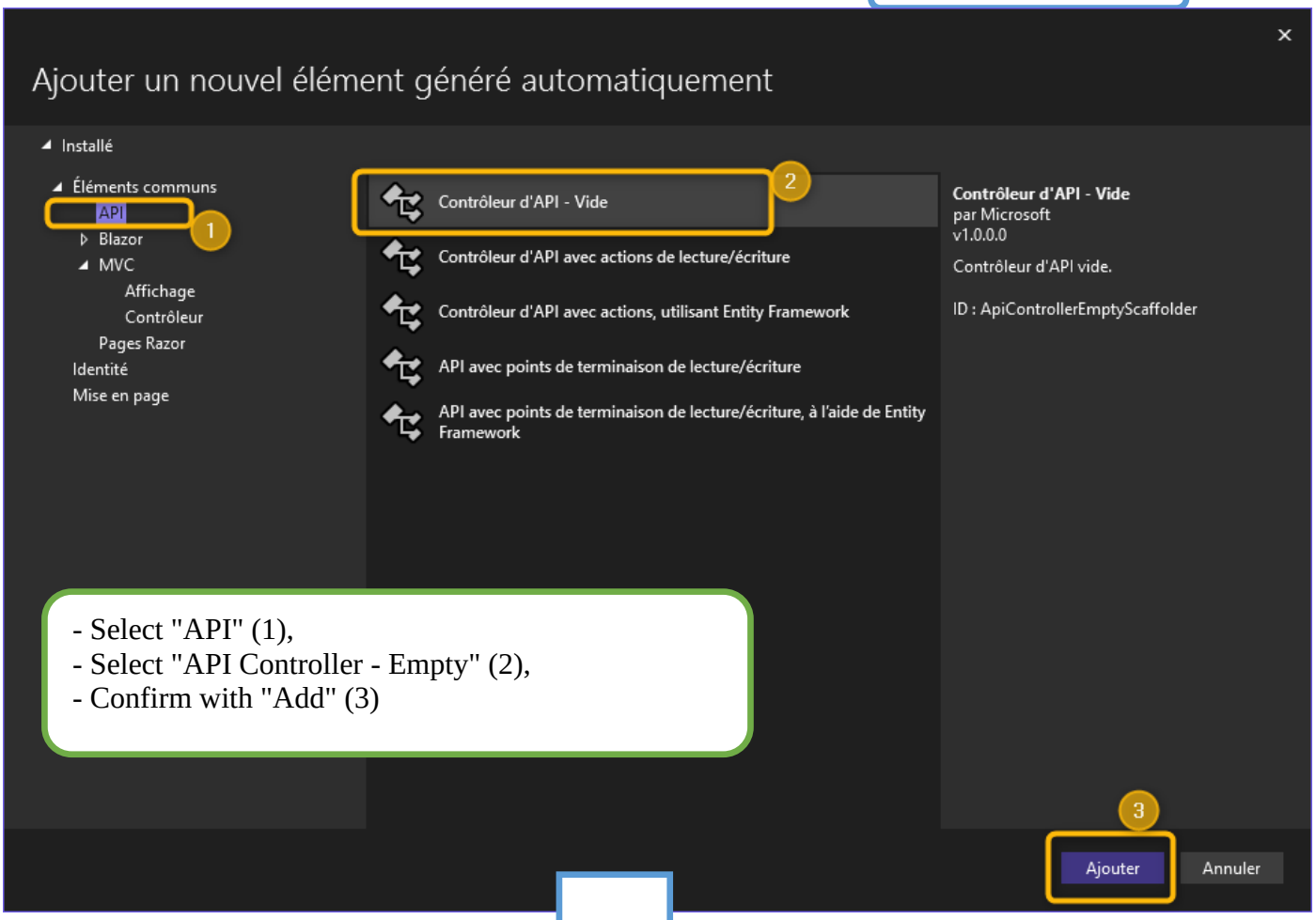
VI Added the "AuthenticateController" controller

VI-A Procedure for adding the controller



B

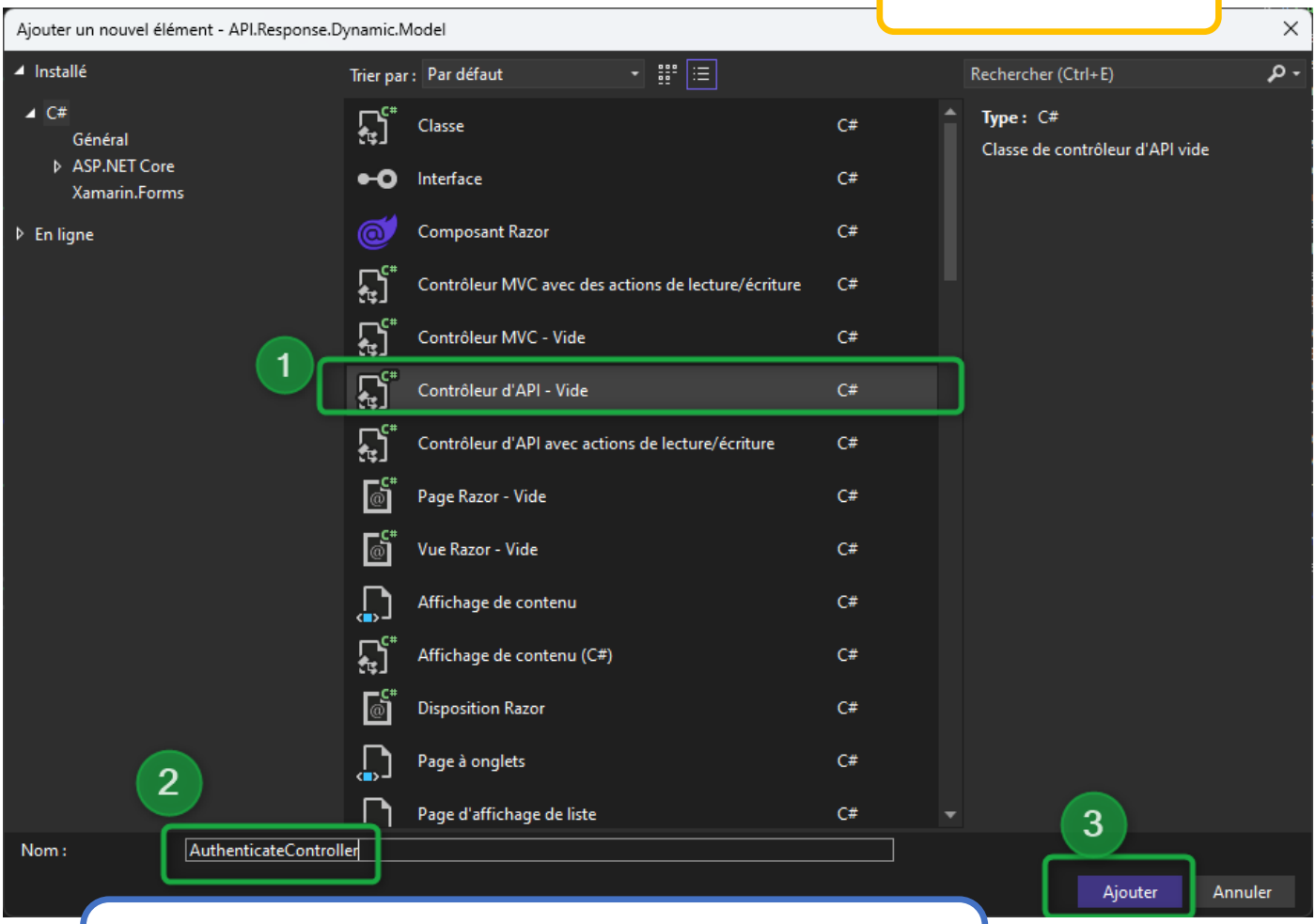
Add a controller



- Select "API" (1),
- Select "API Controller - Empty" (2),
- Confirm with "Add" (3)

C

Add a controller



- Select "API Controller" - empty (1),
- Name the controller "AuthenticateController" (2)
- Validate (3).

VI-B Declaring variables in the "AuthenticateController" controller (with versioning syntax)

⚠️ In this version of the code, I use versioning (\"using Asp.Versioning").
👉See below for the version without versioning

```
using Asp.Versioning;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace API.Response.Dynamic.Model.Controllers.V1_DB2400
{
   [ApiController]
   [ApiVersion("1.0")]
   [Route("api/v{version:apiVersion}/[controller]")]
   public class AuthenticateController : ControllerBase
   {
      #region fields
      // - - - - - - - - - - - -
      // ### Authentication Start ###
      // - - - - - - - - - - - -
      // > Declares a "UserManager" <
      private UserManager<IdentityUser> _userManager = null;

      // > Declare valid to extract the key from "Jwt:Key"  <
      private string ElemKey;

      // > Array of strings used to construct...
        // ...the "host" object <
      private string[] Elems = null;
      // - - - - - - - - - - - -
      // ### Authentication End ###
      // - - - - - - - - - - - -
      #endregion
```

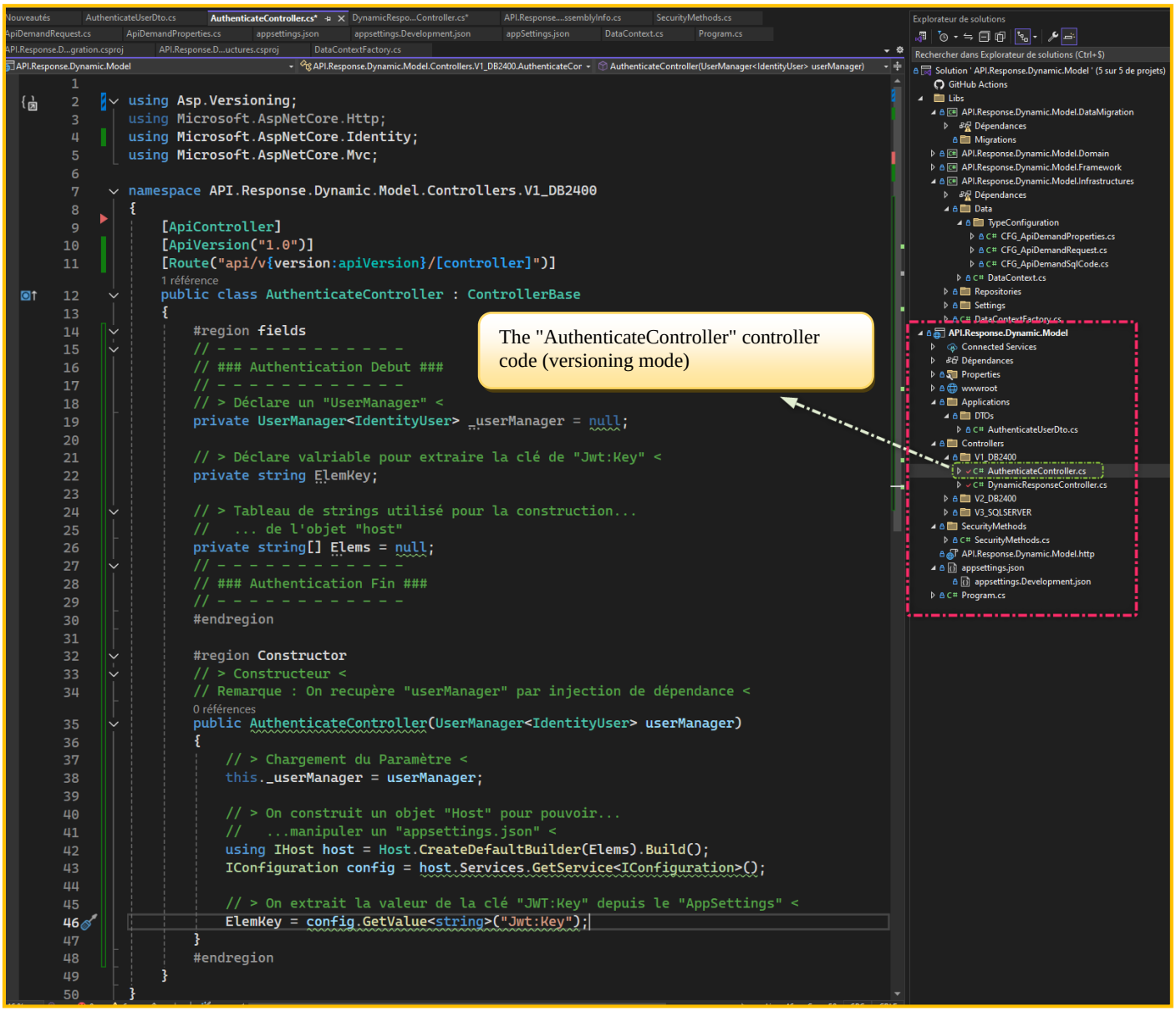
VI-C Declaring variables in the “AuthenticateController” controller (without versioning syntax)

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace API.Response.Dynamic.Model.Controllers.V1_DB2400
{
   [Route("api/[controller]")]
   [ApiController]
   public class AuthenticateController : ControllerBase
   {
      #region fields
      // - - - - - - - - - - - -
      // ### Authentication Start ###
      // - - - - - - - - - - - -
      // > Declares a "UserManager" <
      private UserManager<IdentityUser> _userManager = null;

      // > Declare valid to extract the key from "Jwt:Key" <
      private string ElemKey;

      // >  Array of strings used to construct...
        // ...the "host" object <
       private string[] Elems = null;
      // - - - - - - - - - - - -
      // ### Authentication Fin ###
      // - - - - - - - - - - - -
      #endregion
```

VI-D “AuthenticateController” constructor code

```
      #region Constructor
      // > Constructor <
      // Note: We retrieve "userManager" by dependency injection <

      public AuthenticateController(UserManager<IdentityUser> userManager)
      {
         // > Loading the Parameter<
         this._userManager = userManager;

         // > We construct a "Host" object to be able to...
         // ...manipulate an "appsettings.json" <
         using IHost host = Host.CreateDefaultBuilder(Elems).Build();
         IConfiguration config = host.Services.GetService<IConfiguration>();

         // > We extract the value of the key "JWT:Key" from the "AppSettings"<
         ElemKey = config.GetValue<string>("Jwt:Key");

      }
      #endregion
   }
}
```

VI-E The constructor's code as it should appear



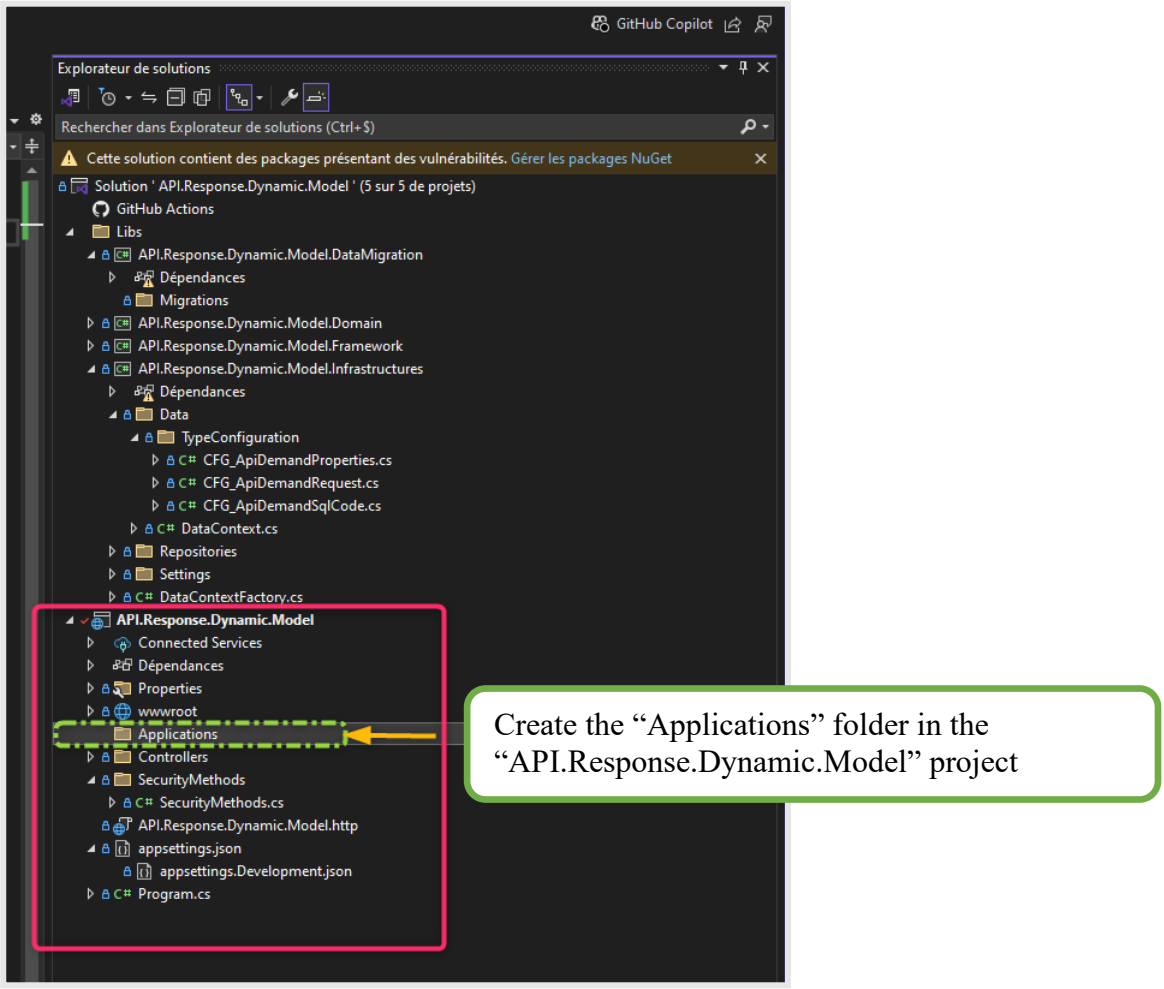

```
using Asp.Versioning;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace API.Response.Dynamic.Model.Controllers.V1_DB2400
{
    [ApiController]
    [ApiVersion("1.0")]
    [Route("api/v{version:apiVersion}/[controller]")]
    1 référence
    public class AuthenticateController : ControllerBase
    {
        #region fields
        // - - - - - - - - - - - - -
        // ### Authentication Debut ###
        // - - - - - - - - - - - - -
        // > Déclare un "UserManager" <
        private UserManager<IdentityUser> _userManager = null;

        // > Déclare valriable pour extraire la clé de "Jwt:Key" <
        private string ElemKey;

        // > Tableau de strings utilisé pour la construction...
        //    ... de l'objet "host"
        private string[] Elems = null;
        // - - - - - - - - - - - - -
        // ### Authentication Fin ###
        // - - - - - - - - - - - - -
        #endregion

        #region Constructor
        // > Constructeur <
        // Remarque : On recupère "userManager" par injection de dépendance <
        0 références
        public AuthenticateController(UserManager<IdentityUser> userManager)
        {
            // > Chargement du Paramètre <
            this._userManager = userManager;

            // > On construit un objet "Host" pour pouvoir...
            //    ...manipuler un "appsettings.json" <
            using IHost host = Host.CreateDefaultBuilder(Elems).Build();
            IConfiguration config = host.Services.GetService<IConfiguration>();

            // > On extrait la valeur de la clé "JWT:Key" depuis le "AppSettings" <
            ElemKey = config.GetValue<string>("Jwt:Key");
        }
        #endregion
    }
}
```

VII Added the Dto (Data Transfer Object) "AuthenticateUserDto

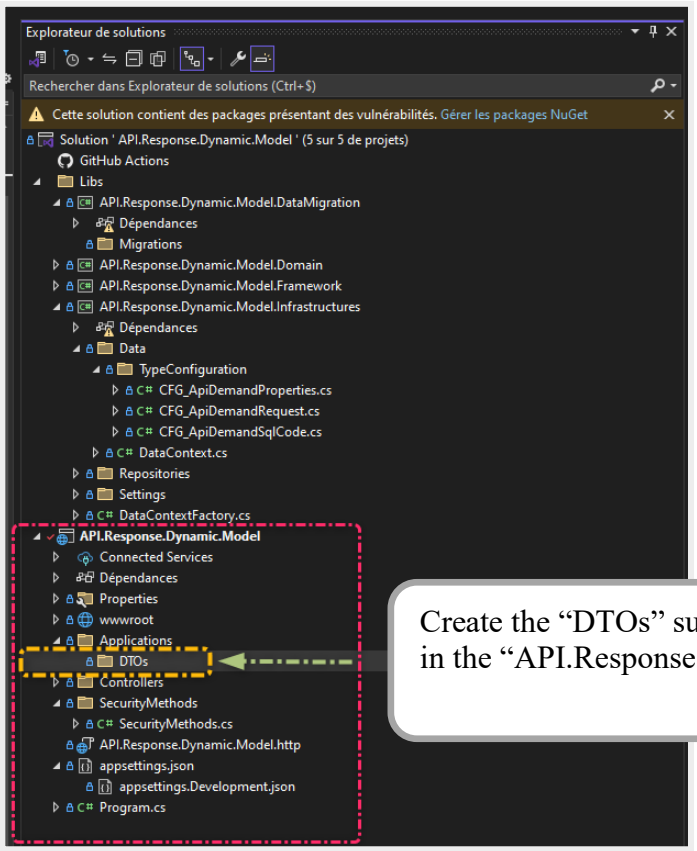
🔥 **Note:** The "**AuthenticateUserDto**" class contains all the login information.

🔥 If the login is successful (see the "Login" method of the "AuthenticateController" controller), the calculated Token is returned (via the "**Token**" property).

VII-A Creating an "Applications" folder

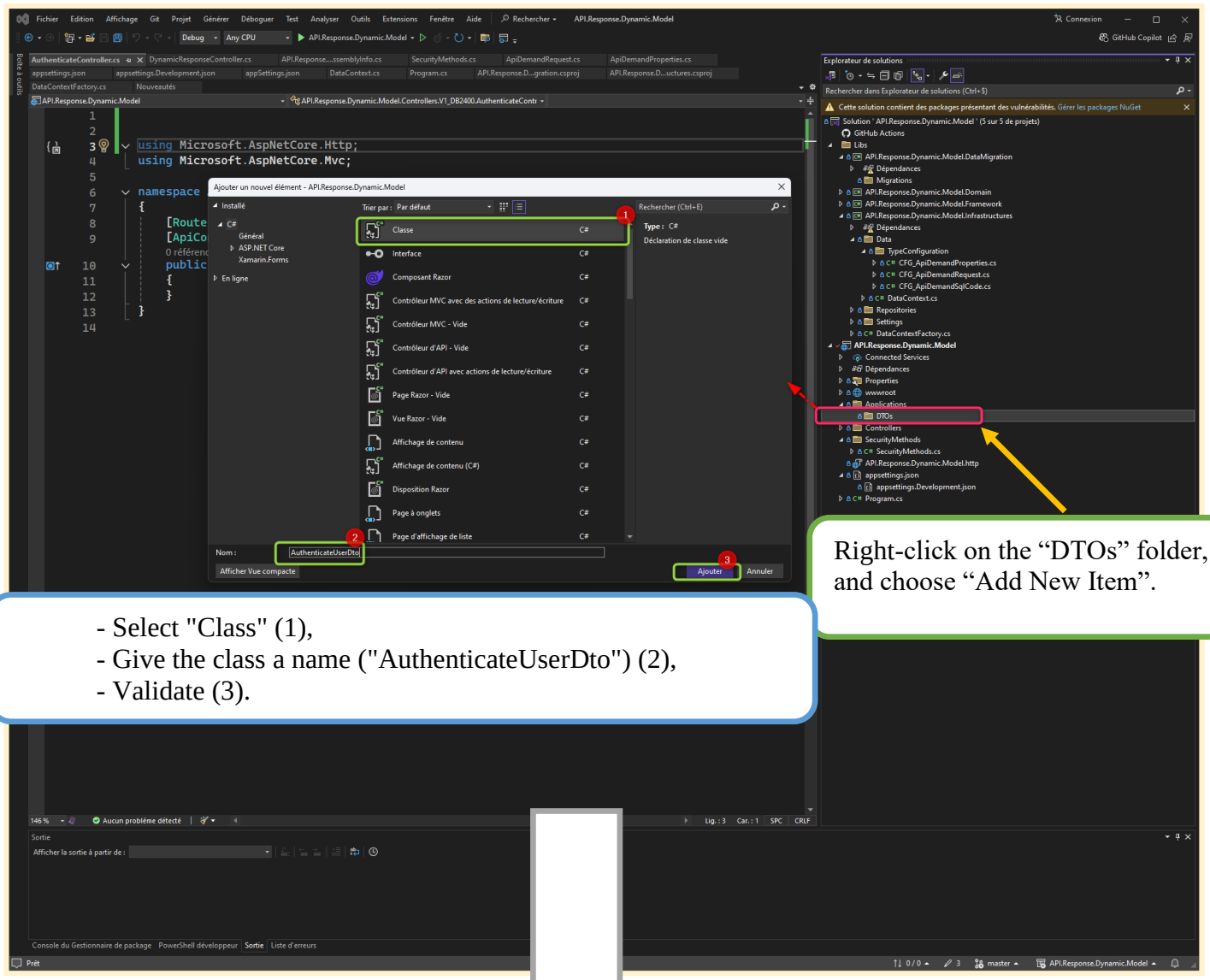


VII-B Creating a "DTOs" folder



Create the "DTOs" subfolder in the "Applications" folder, still in the "API.Response.Dynamic.Model" project.

VII-C Added the “AuthenticateUserDto” DTO



```
namespace API.Response.Dynamic.Model.Applications.DTOs
{
   public class AuthenticateUserDto
   {
      #region properties

      // > The Login will contain the email<
      public string Login { get; set; }

      public string Password { get; set; }

      public string Name { get; set; }

      // > Token calculated and returned <
      public string Token   { get; set; }

      // > Contains possible error message <
      public string ErrorMsge { get; set; }

      #endregion
   }
}
```

VIII Added the “Register” method in the “AuthenticateController” controller

🎯 The “**Register**” method will allow the caller to register (email address + password) in the authentication system.

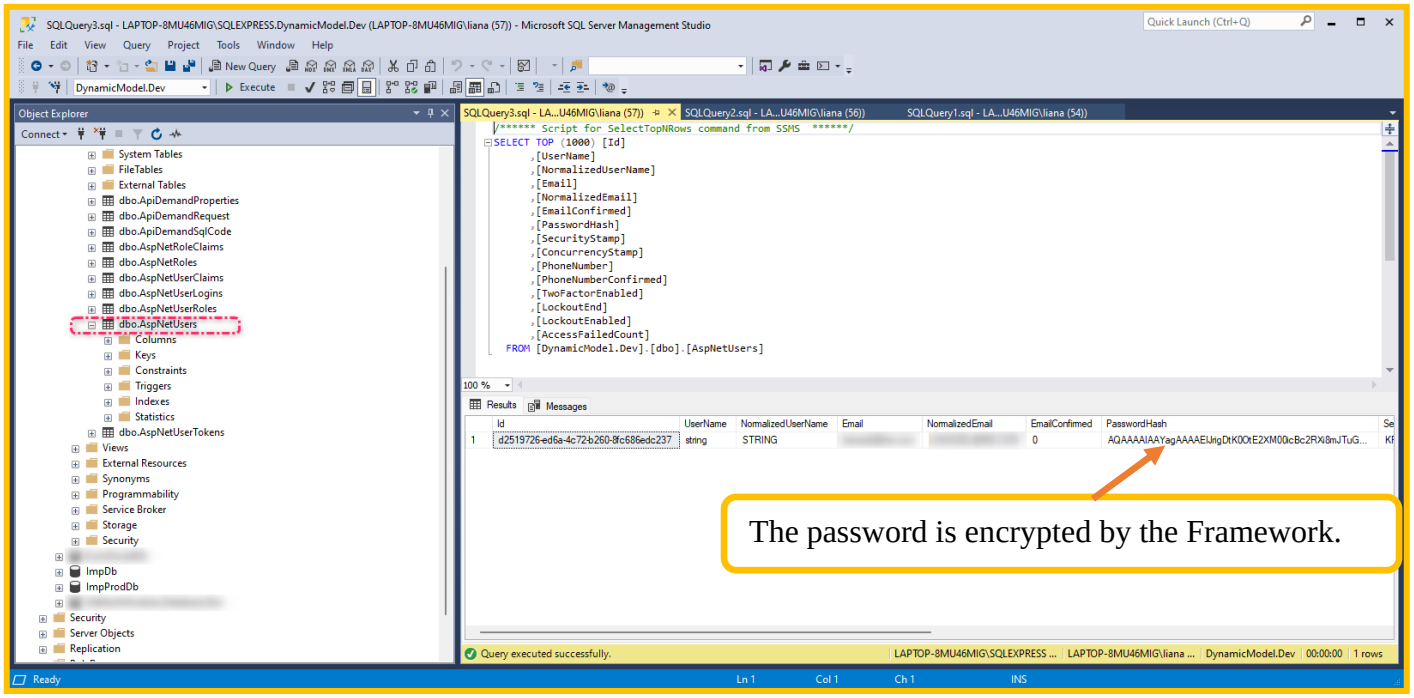
VIII-A The Code of the “Register” Method

```
#region methods

[Route("Register")]
[HttpPost]
public async Task<IActionResult> Register([FromBody] AuthenticateUserDto userDto)
{
    // > By default, pessimistic<
    IActionResult result = this.BadRequest();

   // ----------------------------
   // ## START - Constructs the "user" object ##
   // ----------------------------
   var user = new IdentityUser(userDto.Login);
   user.Email = userDto.Login;
   user.UserName  = userDto.Name;
   // ----------------------------
   // ## END - Constructs the "user" object ##
   // ----------------------------

   // > Attempting to write to the new User Base <
   var success = await this._userManager.CreateAsync(user, userDto.Password);

   if (success.Succeeded)
   {
      try
      {
         // > We retrieve the Token calculated by the "GenerateJwtToken" method <
         userDto.Token = this.GenerateJwtToken(user);

         // > We return the Dto with the calculated Token <
         result = this.Ok(userDto);
      }

      // >  Anomaly detected during token generation ? <
      catch (Exception ex)
      {
          StringBuilder sb = new StringBuilder();

          // > Identification of the message code <
          sb.Append("JWT_ERR_REGISTER_A1 - ");

         // > Retrieving the error message <
         sb.Append(ex.Message);
         userDto.ErrorMsge = sb.ToString();

         // > We return the Dto with the error message <
         result = this.BadRequest(userDto);
      }

      finally
      { }
   }
```

```
    // > Something went wrong <
    // - Incorrect password: does NOT comply with the predefined rules,
    // or
    // - A user with this login already exists...
    else
    {
        StringBuilder sb = new StringBuilder();

        // > Identification of the message code <
        sb.Append("JWT_ERR_REGISTER_A2 - ");

        foreach ( char item in success.Errors.ToString())
        {
            sb.Append(item);
        }

        userDto.ErrorMsge = sb.ToString();
        result = this.BadRequest(userDto);
    }

    return result;
}
```

VIII-B The "[DBO]AspNetUsers" table contains registration information.

.



VIII-C The code of the private method "GenerateJwtToken"

🔥 The "**GenerateJwtToken**" method can be placed in the "**AuthenticateController**" controller (this is the controller that contains the "**Register**" method and the "**Login**" method).

🔥 This method "**GenerateJwtToken**" is responsible for calculating the Token: it is returned in the DTO "**AuthenticateUserDto**".

🔥 Here is the code for "**GenerateJwtToken**" which is called by the "**Login**" method (see below), and therefore also the "**Register**" method.

✒️ Below is the code for the "**GenerateJwtToken**" method:

```
private string GenerateJwtToken( IdentityUser user)
 {
    // We define the Jwt token which will be responsible...
    // ...of the creation of our Token <
    var jwtTokenHandler = new JwtSecurityTokenHandler();

    // > We retrieve our secret code from "app.settings"  <
    // ( Caution: Work with "UTF8" here because in the method...
    // ..."AddCustomAuthentication" of the "SecurityMethods" class...
    // ..in which the TOKEN is described, "UTF8" is declared. )

    // (It is important to keep the same parameters, otherwise the...
     // ...Framework will NOT be able to compare).
    var key = Encoding.UTF8.GetBytes(ElemKey);

   // ------------------------------
   // --- Token Description ----
   // ------------------------------
   // > We need to use "claims"...
   // ...which are properties in our token that provide...
   // ...information about the token belonging to the user...
   // ...and so we have information such as the user's ID,...
   // ...the user's name, their email address...
   // ...The good thing is that this information is generated by...
   // ...our server and our identity framework, which are valid and reliable.

    var tokenDescriptor = new SecurityTokenDescriptor
    {
       Subject = new System.Security.Claims.ClaimsIdentity(new[]
        {
           new Claim("Id",user.Id),
           new Claim(JwtRegisteredClaimNames.Sub,user.Email) ,
           new Claim(JwtRegisteredClaimNames.Email,user.Email),
           // > The JTI is used for our Token <
           new Claim(JwtRegisteredClaimNames.Jti,Guid.NewGuid().ToString())
        }),

        // > The Token will have a duration of three minutes <
       Expires = DateTime.UtcNow.AddMinutes(3),

        // > Here we add information about the encryption algorithm...
        // ...which will be used to decrypt our token <
       SigningCredentials = new SigningCredentials(

       // > Caution: Working with "SymmetricSecurityKey" here because...
       // ...in the "addCustomAuthentication" method of the class...
       // ..."SecurityMethods" in which we describe the "TOKEN", ...
       // ...we declare the "SymmetricSecurityKey". <
       // > It is important to keep the same parameters, otherwise...
       // ...the FrameWork will NOT be able to compare <
       new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };

    var token = jwtTokenHandler.CreateToken(tokenDescriptor);
    var JwtToken = jwtTokenHandler.WriteToken(token);
    return JwtToken;
 }
```

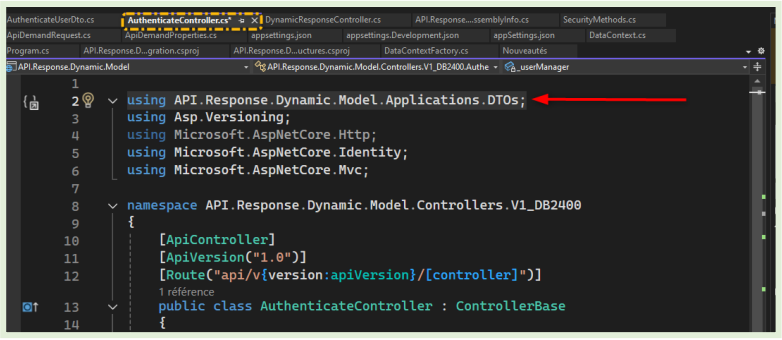
IX Added the “Login” method in the “AuthenticateController” controller

🌟 The “Login” method will allow the caller to identify themselves (email address + password)

💡 **Note:** The "**Login**" method of the "**AuthenticateController**" controller performs a login test with the password contained in the "**AuthenticateUserDto**" **DTO** which will be in the "body" ("[**FromBody**]") of the http request.

IX-A The code for the public method "Login"

💡 **Note:** Remember to add the using " **using API.Response.Dynamic.Model.Applications.DTOs**; ;" in the "AuthenticateController" controller.

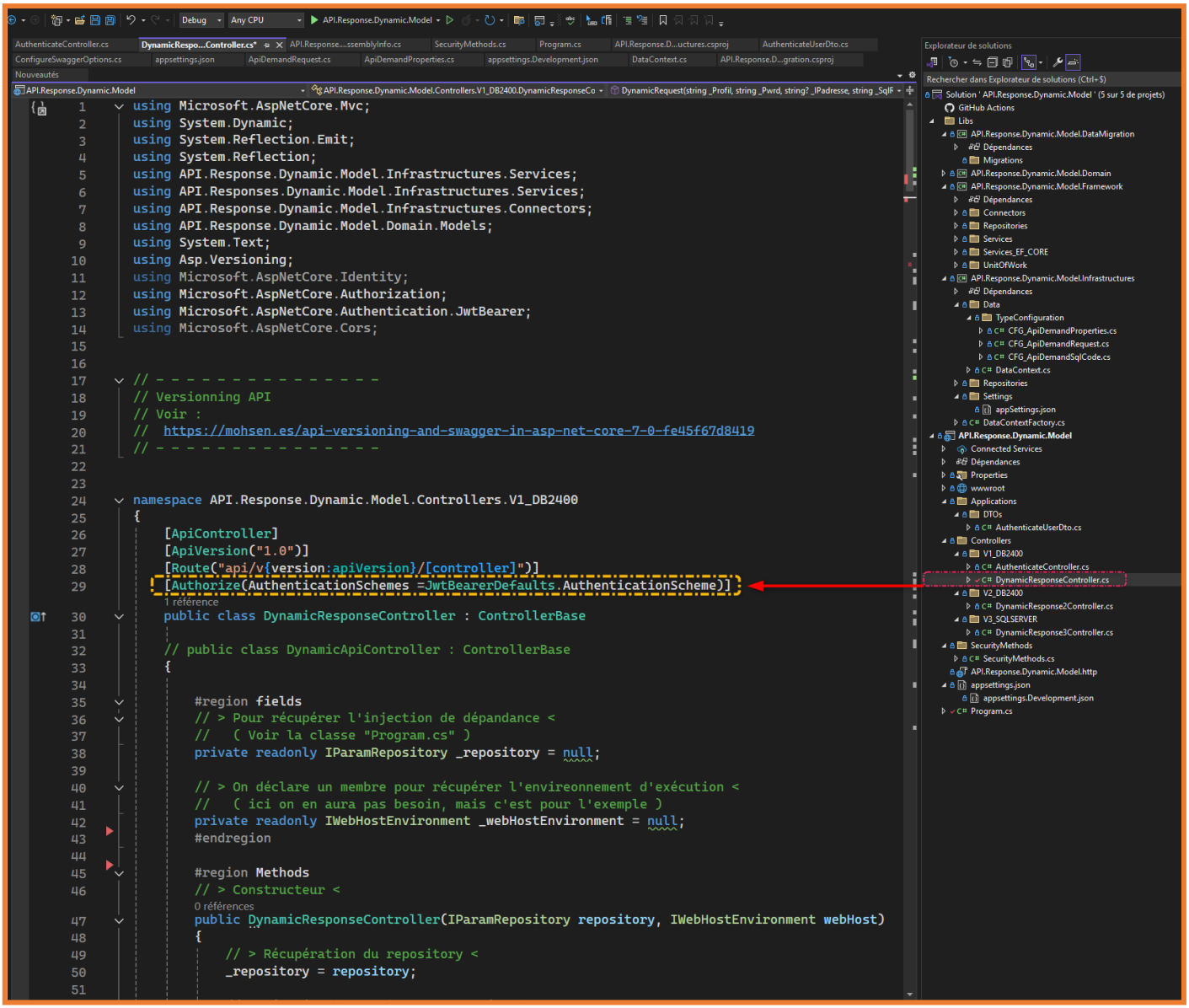



```
[Route("Login")]
[HttpPost]
public async Task<IActionResult> Login([FromBody] AuthenticateUserDto DtoUser)
{
   // > By default, pessimistic <
   IActionResult result = this.BadRequest();

   // > We'll verify the profile using the email address and password <
   // > The idea is to be able to authenticate with the login that...
  // ...IS the email address. <
  // > Since we're searching using the email address (contained in...
  // ...the login, we use the "FindByEmailAsync" method <

   // > Reminder: when our "Login" method is "async",...
   // ...we must do an "await" <
   var user = await this._userManager.FindByEmailAsync(DtoUser.Login);

   if (user != null)
   {
      // > We will check the password <
      var verif = await this._userManager.CheckPasswordAsync(user, DtoUser.Password);
      if (verif)
        {
           // > We load the DTO with the calculation of the token returned by the method...
           // ..."GenerateJwtToken" <
            result = this.Ok(new AuthenticateUserDto()
            {
              Login = user.Email,
              Name = user.UserName,
             Token = this. GenerateJwtToken(user),
            });
        }
```

```
        else
        {
            StringBuilder sb = new StringBuilder();
            // > Identification of the message code <
            sb.Append("JWT_ERR_LOGIN_A1 - Invalid email address or password");
            result = this.BadRequest(new AuthenticateUserDto()
            {
                Login = user.Email,
                Name = user.UserName,
                // > Calls the "GenerateJwtToke" method, which is responsible for...
                // ...calculating a new Token <
                Token = null,
                ErrorMsge = sb.ToString()
            });
        }
    }
    // > We return the result <
    return result;
}
```

X The "DynamicResponseController" controller

X-A Added the "[Authorize]" attribute

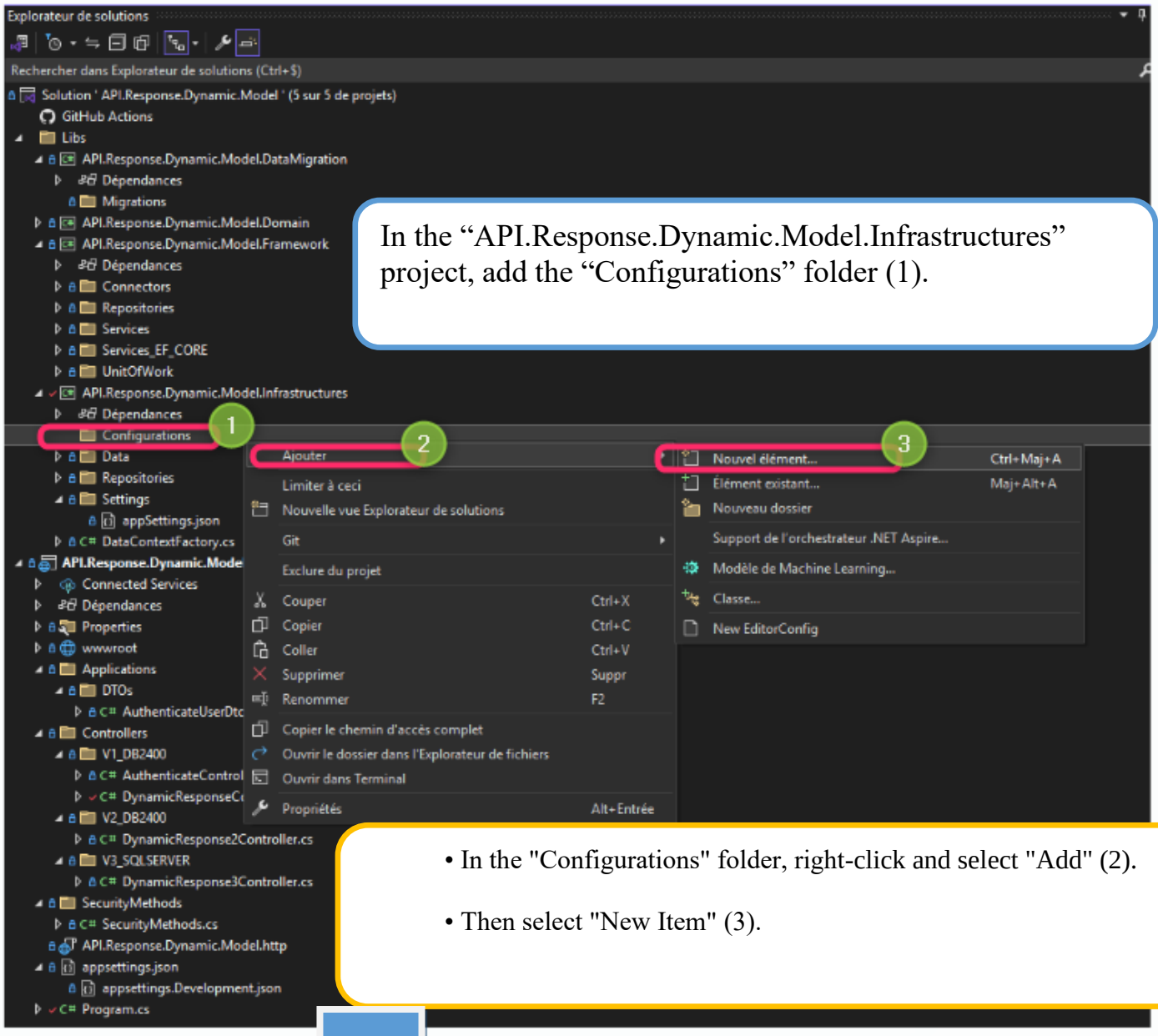


```
using Microsoft.AspNetCore.Mvc;
using System.Dynamic;
using System.Reflection.Emit;
using System.Reflection;
using API.Response.Dynamic.Model.Infrastructures.Services;
using API.Responses.Dynamic.Model.Infrastructures.Services;
using API.Response.Dynamic.Model.Infrastructures.Connectors;
using API.Response.Dynamic.Model.Domain.Models;
using System.Text;
using Asp.Versioning;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Cors;


// - - - - - - - - - - - - - - - -
// Versionning API
// Voir :
//  https://mohsen.es/api-versioning-and-swagger-in-asp-net-core-7-0-fe45f67d8419
// - - - - - - - - - - - - - - - -


namespace API.Response.Dynamic.Model.Controllers.V1_DB2400
{
    [ApiController]
    [ApiVersion("1.0")]
    [Route("api/v{version:apiVersion}/[controller]")]
    [Authorize(AuthenticationSchemes =JwtBearerDefaults.AuthenticationScheme)]
    1 référence
    public class DynamicResponseController : ControllerBase

    // public class DynamicApiController : ControllerBase
    {

        #region fields
        // > Pour récupérer l'injection de dépandance <
        //   ( Voir la classe "Program.cs" )
        private readonly IParamRepository _repository = null;

        // > On déclare un membre pour récupérer l'envireonnement d'exécution <
        //   ( ici on en aura pas besoin, mais c'est pour l'exemple )
        private readonly IWebHostEnvironment _webHostEnvironment = null;
        #endregion

        #region Methods
        // > Constructeur <
        0 références
        public DynamicResponseController(IParamRepository repository, IWebHostEnvironment webHost)
        {
            // > Récupération du repository <
            _repository = repository;
```

The attribute "[Authorize(AuthenticationSchemes =JwtBearerDefaults.AuthenticationScheme)]" will allow the validity of the transmitted Token to be checked.

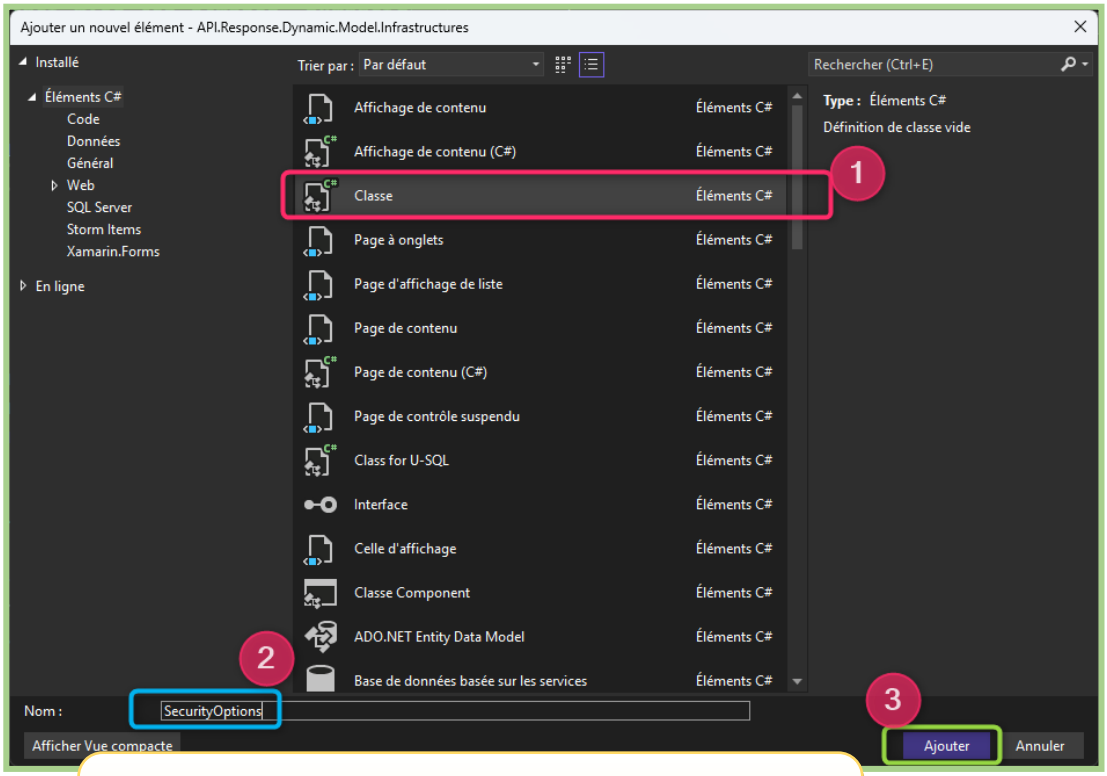
XI Improve the configuration….

XI-A Create a "SecurityOptions" class to store the Token calculation key

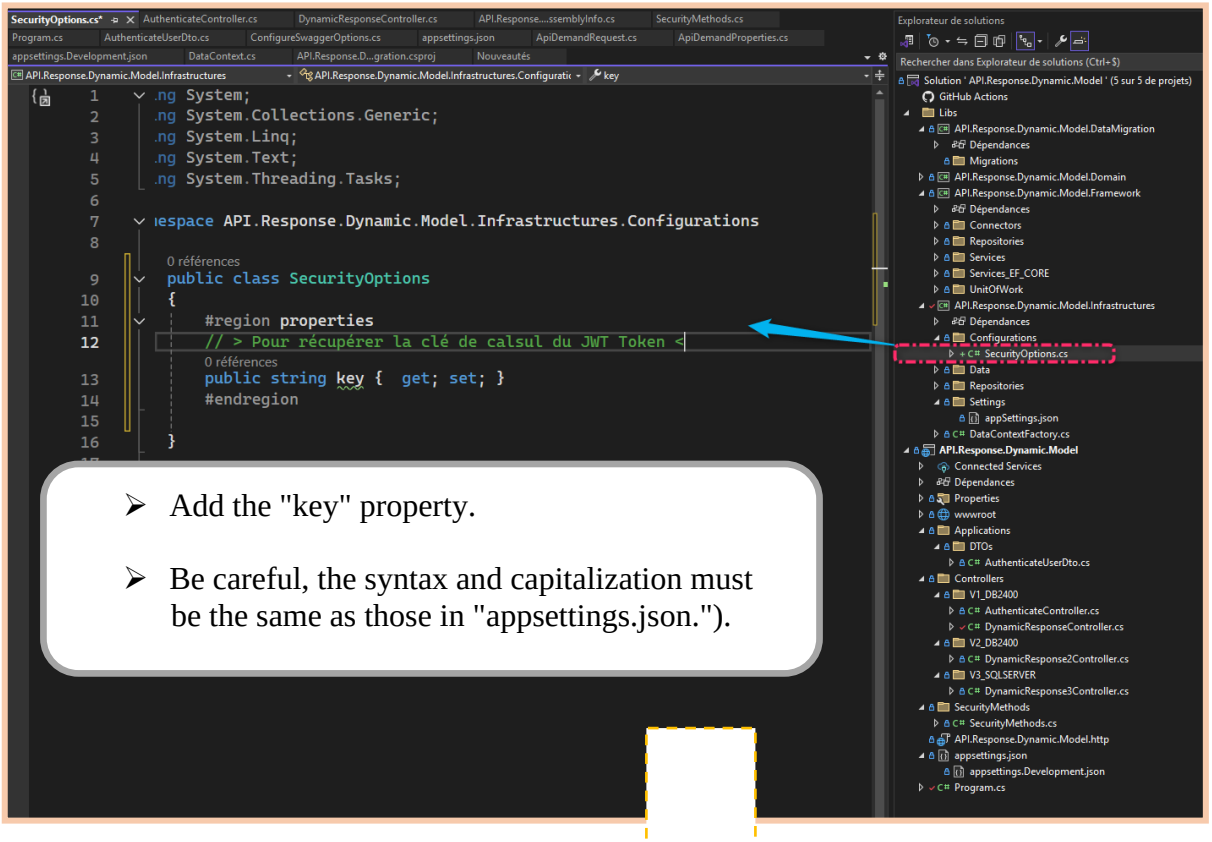


In the "API.Response.Dynamic.Model.Infrastructures" project, add the "Configurations" folder (1).

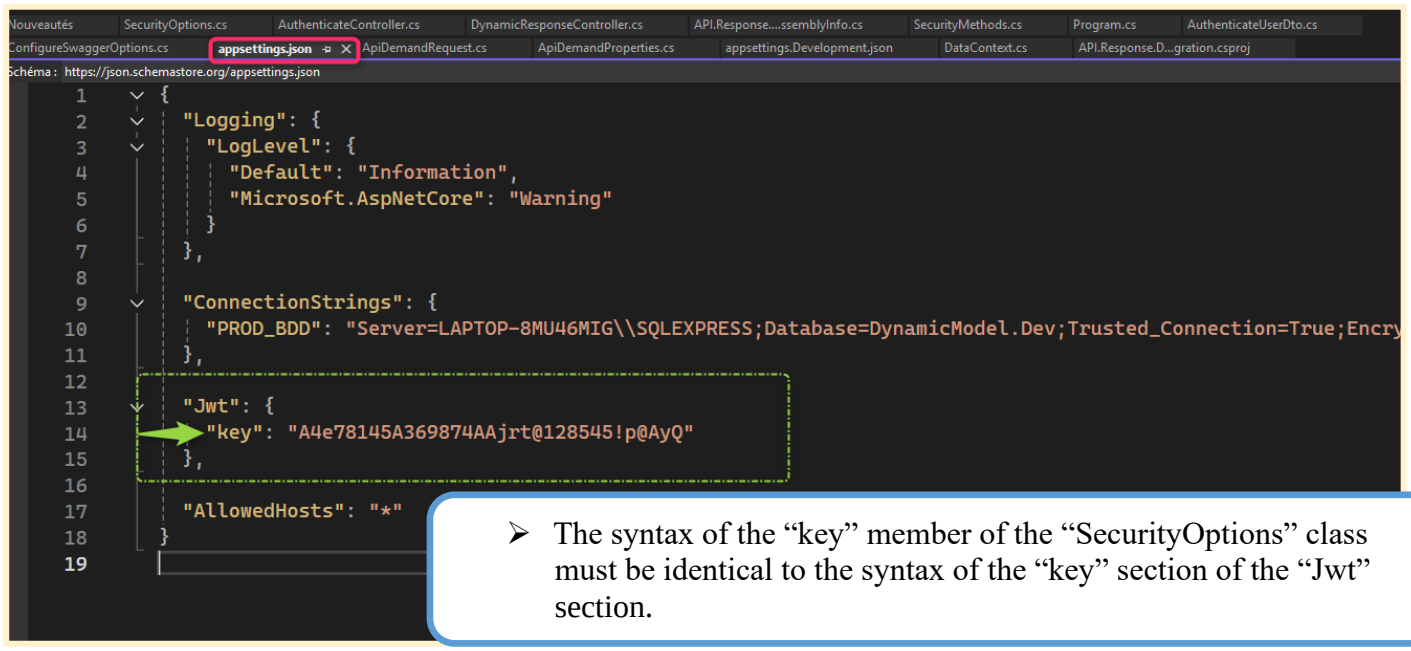
- In the "Configurations" folder, right-click and select "Add" (2).
- Then select "New Item" (3).



- Select "Class" (1),
- Name the class "SecurityOptions" (2),
- Confirm with "Add" (3)

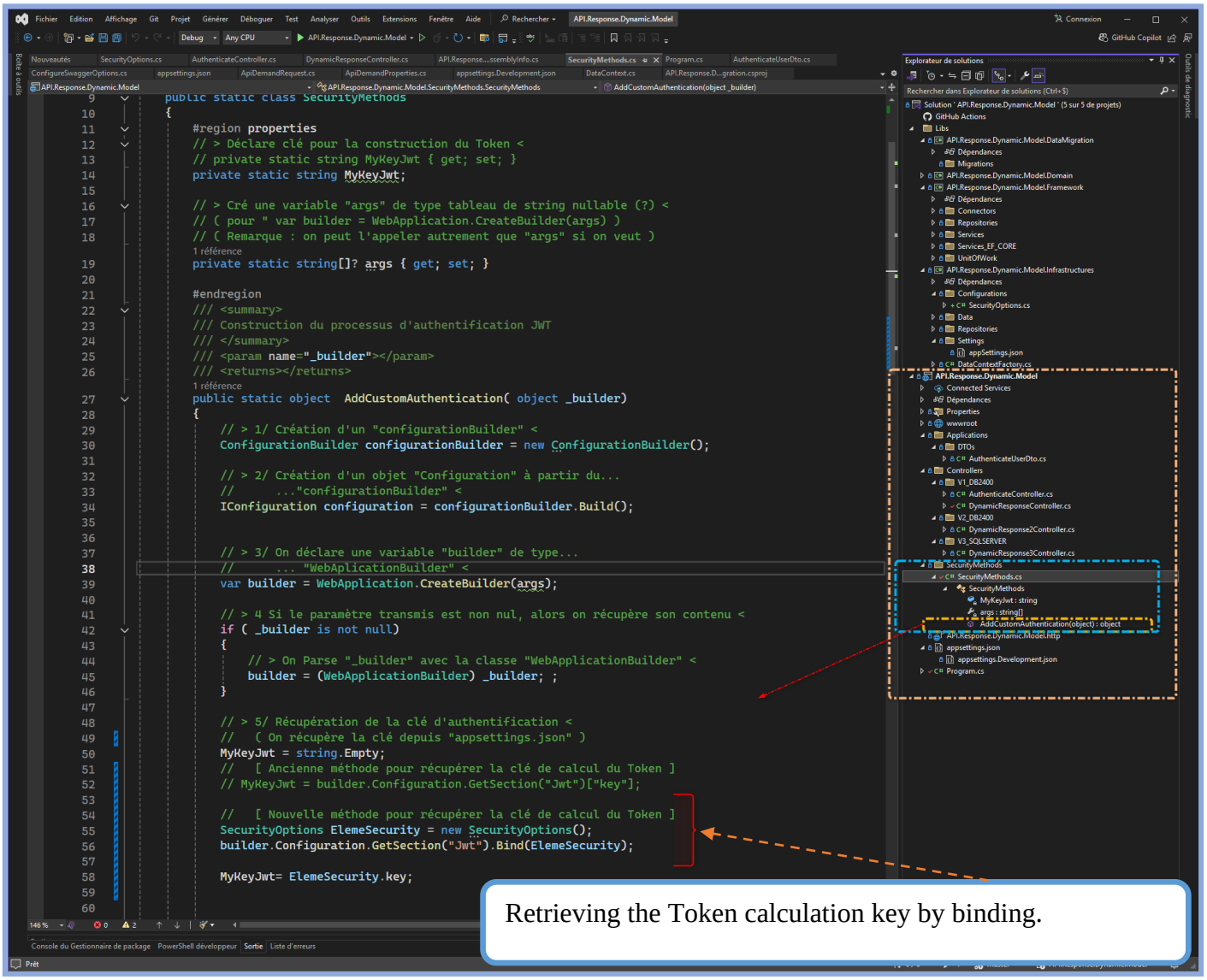


- Add the "key" property.
- Be careful, the syntax and capitalization must be the same as those in "appsettings.json.").



- The syntax of the “key” member of the “SecurityOptions” class must be identical to the syntax of the “key” section of the “Jwt” section.

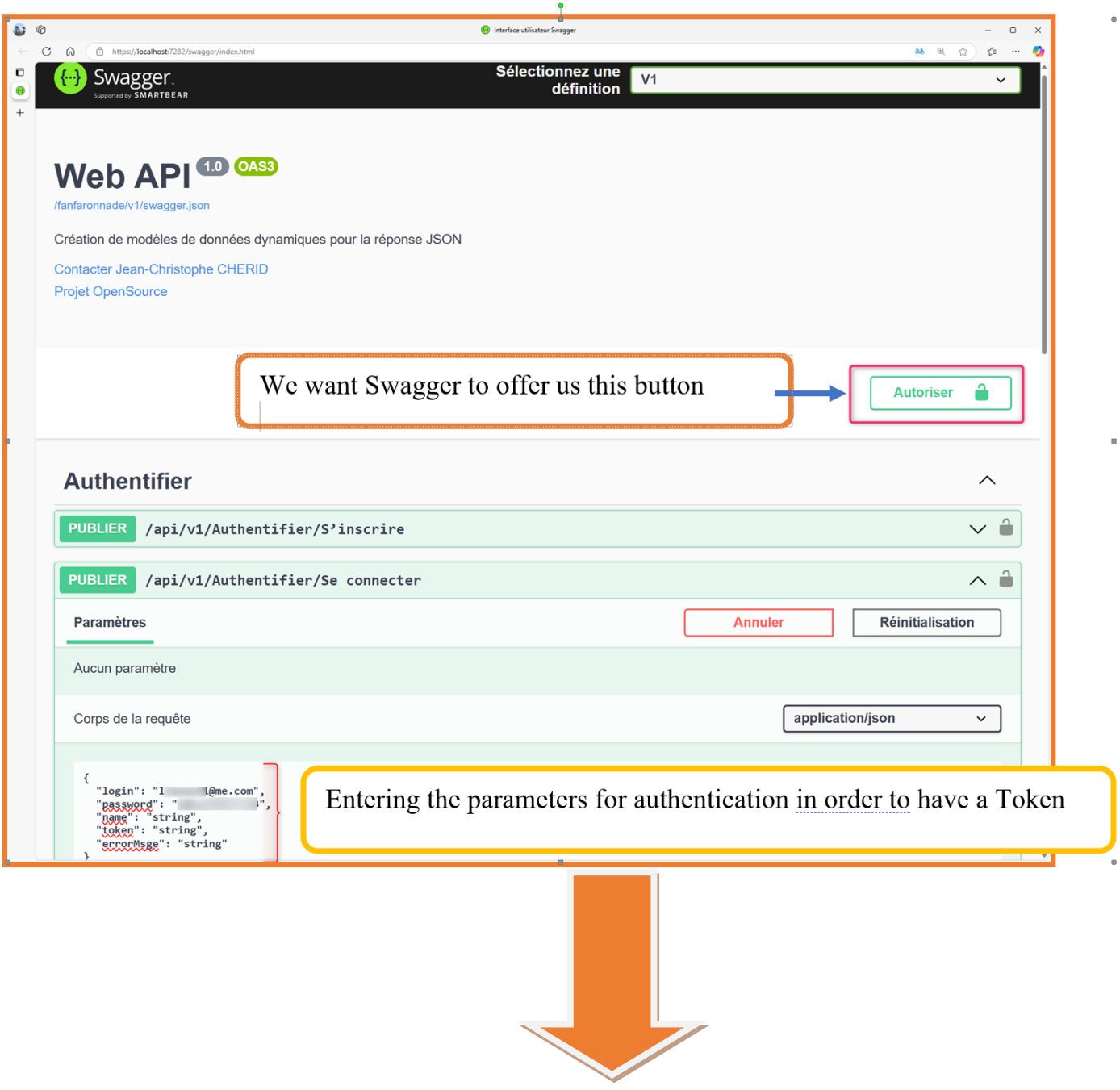
XI-B Modify the "AddCustomAuthentication" method of the "SecurityMethods" class

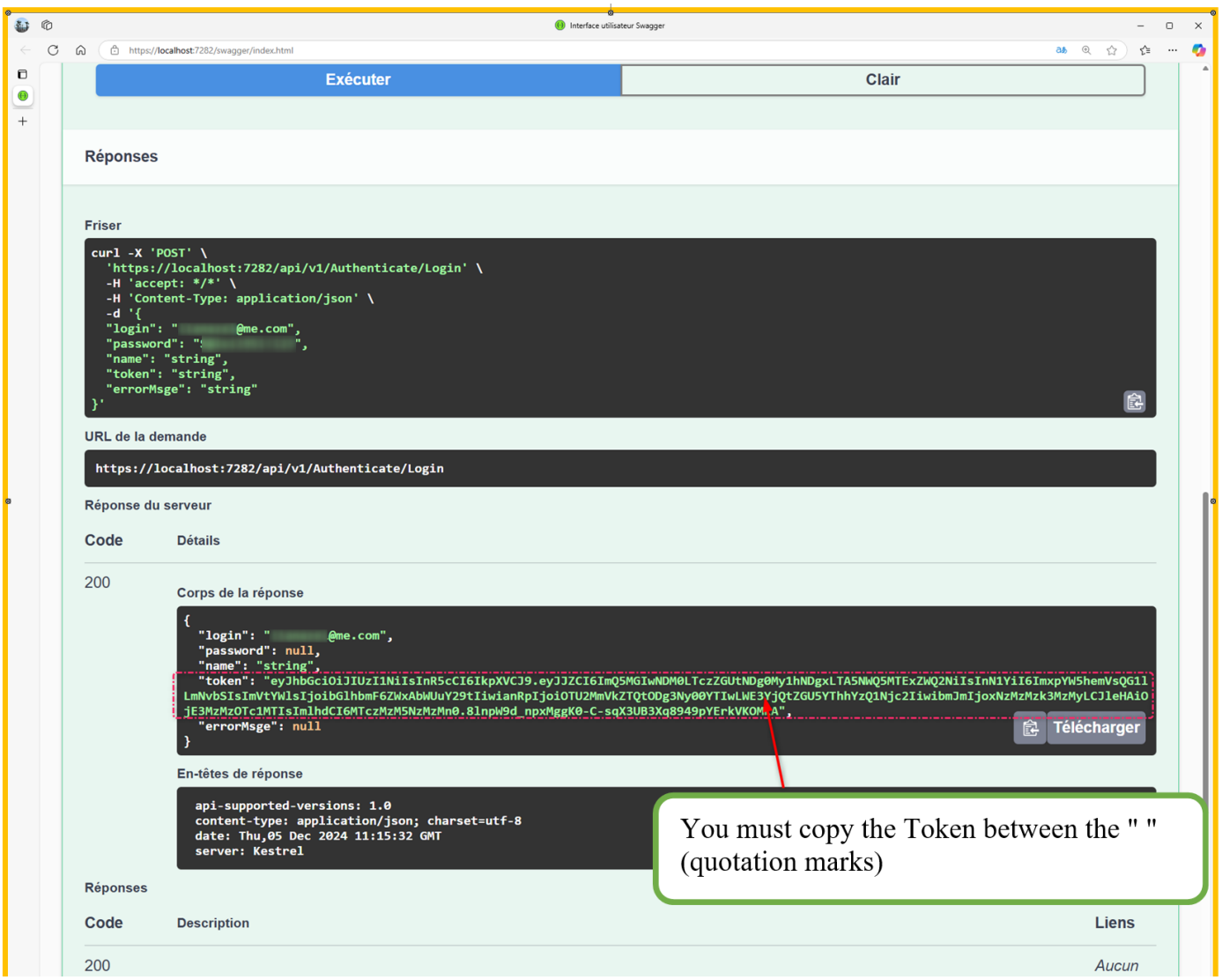


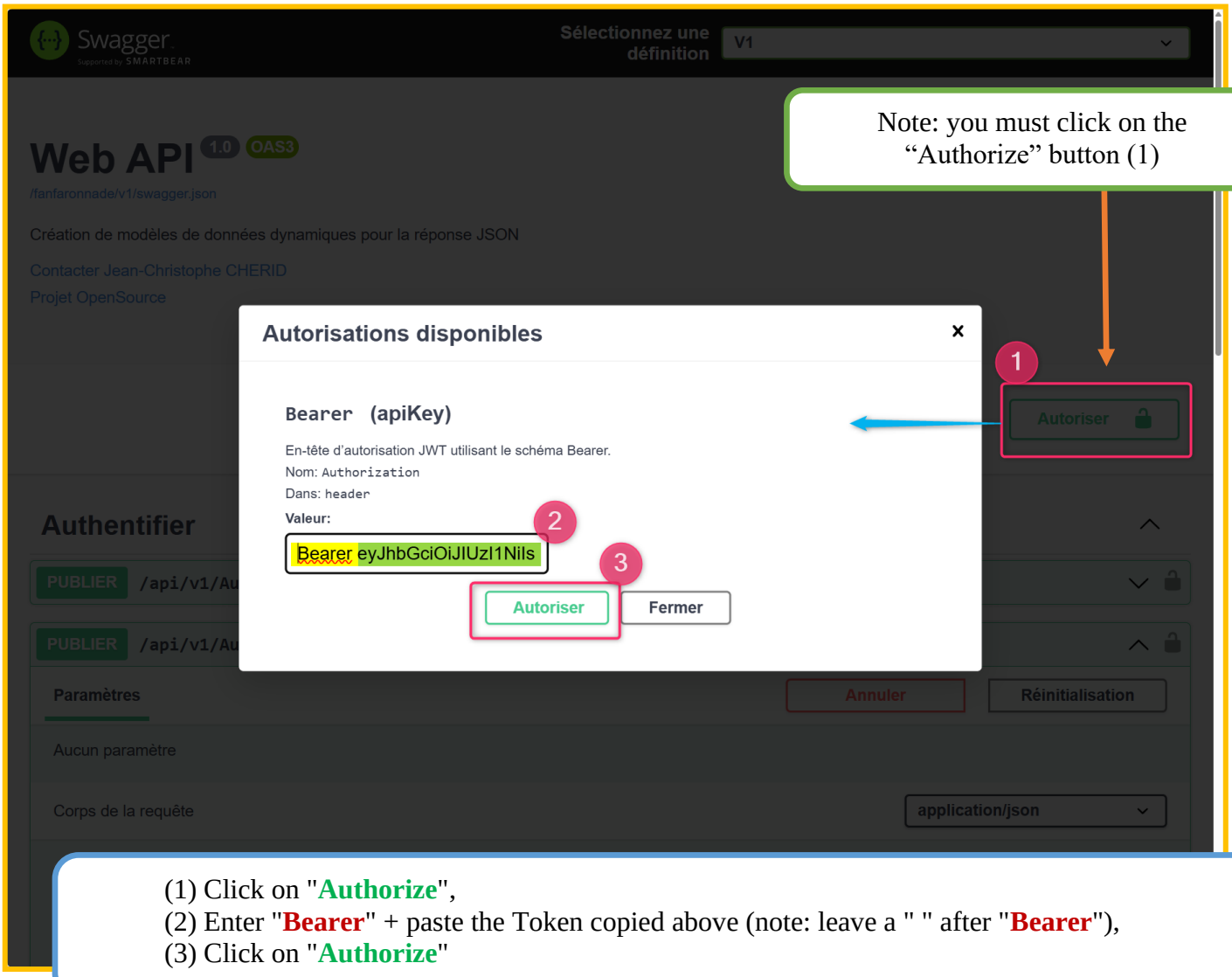
XII Test authentication with swagger

XII-A Configure Swagger in the "Program.cs" class for Token input

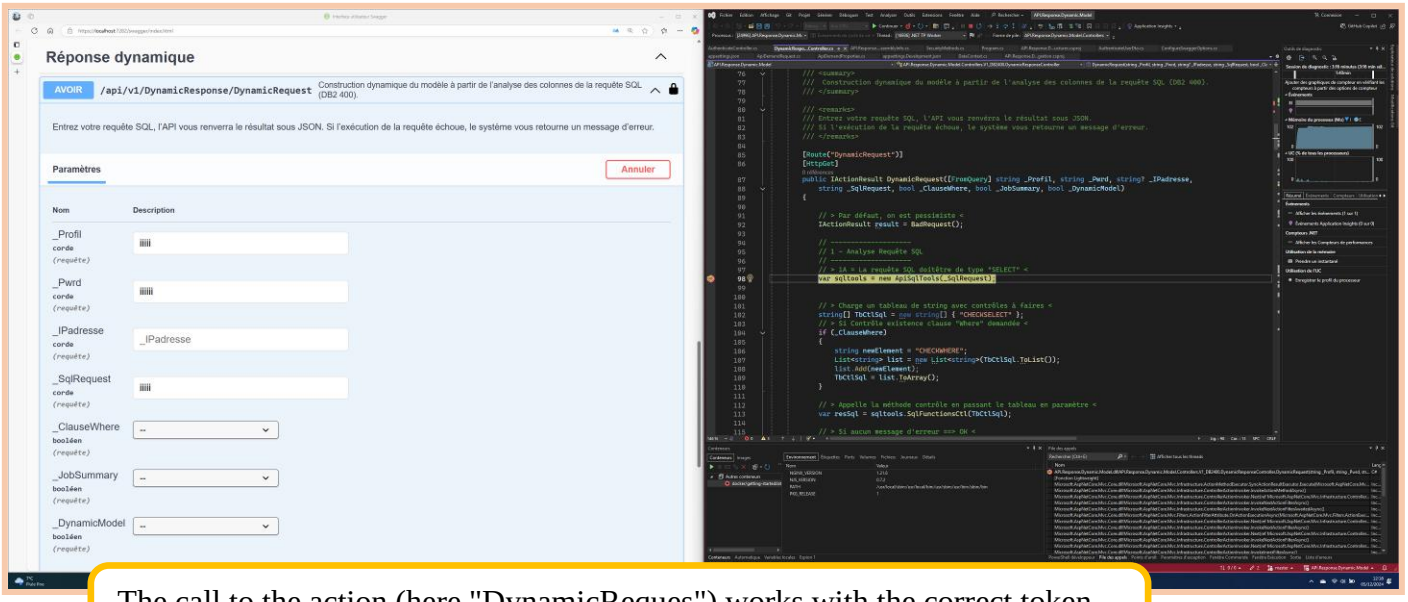
XII-A-1 What we want to achieve




Interface utilisateur Swagger
https://localhost:7282/swagger/index.html
Exécuter
Clair
Réponses
Friser
curl -X 'POST' \
'https://localhost:7282/api/v1/Authenticate/Login' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"login": "@me.com",
"password": "",
"name": "string",
"token": "string",
"errorMsge": "string"
}'
URL de la demande
https://localhost:7282/api/v1/Authenticate/Login
Réponse du serveur
Code
Détails
200
Corps de la réponse
{
"login": "@me.com",
"password": null,
"name": "string",
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJZCI6ImQ5MGIwNDM0LTczZGUtNDg0My1hNDgxLTA5NWQ5MTExZWQ2NiIsInN1YiI6ImxpYW5hemVsQG1lLmNvbSIsImVtYWlsIjoibGlhbmF6ZWxAbWUuY29tIiwianRpIjoiOTU2MmVkZTQtODg3Ny00YTIwLWE3YjQtZGU5YThhYzQ1Njc2IiwibmJmIjoxNzMzMzk3MzMyLCJleHAiOjE3MzMzOTc1MTIsImlhdCI6MTczMzM5NzMzMn0.8lnpW9d_npxMggK0-C-sqX3UB3Xq8949pYErkVKOMcA",
"errorMsge": null
}
Télécharger
En-têtes de réponse
api-supported-versions: 1.0
content-type: application/json; charset=utf-8
date: Thu,05 Dec 2024 11:15:32 GMT
server: Kestrel
You must copy the Token between the " " (quotation marks)
Réponses
Code
Description
Liens
200
Aucun



(1) Click on "**Authorize**",
(2) Enter "**Bearer**" + paste the Token copied above (note: leave a " " after "**Bearer**"),
(3) Click on "**Authorize**"



The call to the action (here "DynamicReques") works with the correct token.
(We're now entering the breakpoint).

XII-A-2 The code to insert into the “Program.cs” class

See also my documentation «TEC_C#_NET_JWT_AUTHENTICATION_ERROR_MESSAGE_FAILED_TO_EXECEUTE.DOC ».

```
builder.Services.AddSwaggerGen(options =>
{
   // > Add a custom operation filter that sets default values <
   options.OperationFilter<SwaggerDefaultValues>();

   var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name} .xml";
   var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
   options.IncludeXmlComments(xmlPath);

   // > Add JWT Authentication 1/2 <
   options.AddSecurityDefinition( "Bearer ", new Microsoft.OpenApi.Models.OpenApiSecurityScheme
   {
      Description = "JWT Authorization header using Bearer schema.",
      Name = "Authorization",
      In = ParameterLocation.Header,
      Type = SecuritySchemeType.ApiKey

   });

   // > Add JWT Authentication 2/2 <
   options.AddSecurityRequirement(new OpenApiSecurityRequirement
   {
      {
         new OpenApiSecurityScheme
         {
            Reference = new OpenApiReference
            {
               Type = ReferenceType.SecurityScheme,
               Id = "Bearer "
            },
            Scheme = "oauth2" , Name = "Bearer " ,

            In = ParameterLocation.Header,
         },

         new List<string>()

      }
   });

});
```