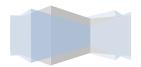


# .Net 8 JSON Web Authentication

Step by step

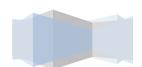


# Summary

<b>I – Introduction .....</b>	<b>4</b>
<b>II – Update the Context: replace “DbContext” with “IdentityDbContext” .....</b>	<b>5</b>
II-A <i>Installing the NuGet “Microsoft.AspNetCore.Identity.EntityFrameworkCore” .....</i>	5
II-A-1 <i>Added “using Microsoft.AspNetCore.Identity.EntityFrameworkCore” .....</i>	6
II-B <i>Reminder: How to find out the .Net version of the solution .....</i>	7
<b>III Start creating the database .....</b>	<b>8</b>
III – A <i>SQL Server - Retrieve connection information (connection string) .....</i>	8
III – B <i>SQL SERVER - Program.cs ➔ Start database creation .....</i>	9
<b>IV Adding JWT rules .....</b>	<b>10</b>
IV-A <i>"appsettings.json" - Added key for Token calculation .....</i>	10
IV-B <i>Create a "SecurityMethods" folder .....</i>	11
IV-C <i>Install the "System.IdentityModel.Tokens.Jwt" package .....</i>	12
IV-D <i>Install the "Microsoft.AspNetCore.Authentication.JwtBearer" package .....</i>	16
IV-E <i>Creation of the "SecurityMethods" class .....</i>	17
IV-E-1 <i>Location of the "SecurityMethods" class .....</i>	17
IV-E-2 <i>Code of the "AddCustomAuthentication" method: This is where the key is used to calculate the Token .....</i>	18
IV-F <i>Calling the "AddCustomAuthentication()" method in the "Program.cs" class .....</i>	20
IV-F-1 <i>The "Program.cs" class .....</i>	20
IV-F-2 <i>Line of code .....</i>	20
<b>V Configuring the “Context Identity” .....</b>	<b>21</b>
V-A <i>Installing the NuGet “Microsoft.AspNetCore.Identity.UI” .....</i>	21
V-B - (.Net 8) <i>Implementation in the "Program.cs" class .....</i>	23
V-B-1 <i>Code to insert into the "Program.cs" class .....</i>	23
V-B-2 <i>Presentation of the class "Program.cs" with the code inserted .....</i>	23
<b>VI Added the “AuthenticateController” controller .....</b>	<b>24</b>
VI-A <i>Procedure for adding the controller .....</i>	24
VI-B <i>Declaring variables in the “AuthenticateController” controller (with versioning syntax) .....</i>	27
VI-C <i>Declaring variables in the “AuthenticateController” controller (without versioning syntax) .....</i>	28
VI-D <i>“AuthenticateController” constructor code .....</i>	28
VI-E <i>The constructor's code as it should appear .....</i>	29
<b>VII Added the Dto (Data Transfer Object) “AuthenticateUserDto” .....</b>	<b>30</b>
VII-A <i>Creating an “Applications” folder .....</i>	30
VII-B <i>Creating a “DTOS” folder .....</i>	31
VII-C <i>Added the “AuthenticateUserDto” DTO .....</i>	32
<b>VIII Added the “Register” method in the “AuthenticateController” controller .....</b>	<b>34</b>
VIII-A <i>The Code of the “Register” Method .....</i>	34
VIII-B <i>The “[dbo]AspNetUsers” table contains registration information .....</i>	35
VIII-C <i>The code of the private method "GenerateJwtToken" .....</i>	36

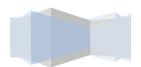


<b>IX Added the “Login” method in the “AuthenticateController” controller.....</b>	<b>37</b>
<i>IX-A The code for the public method "Login".....</i>	<i>37</i>
<b>X The “DynamicResponseController” controller .....</b>	<b>39</b>
<i>X-A Added the “[Authorize]” attribute.....</i>	<i>39</i>
<b>XI Improve the configuration.....</b>	<b>40</b>
<i>XI-A Create a “SecurityOptions” class to store the Token calculation key.....</i>	<i>40</i>
<i>XI-B Modify the "AddCustomAuthentication" method of the "SecurityMethods" class .....</i>	<i>43</i>
<b>XII Test authentication with swagger .....</b>	<b>44</b>
<i>XII-A Configure Swagger in the “Program.cs” class for Token input .....</i>	<i>44</i>
XII-A-1 What we want to achieve.....	44
XII-A-2 The code to insert into the “Program.cs” class .....	48



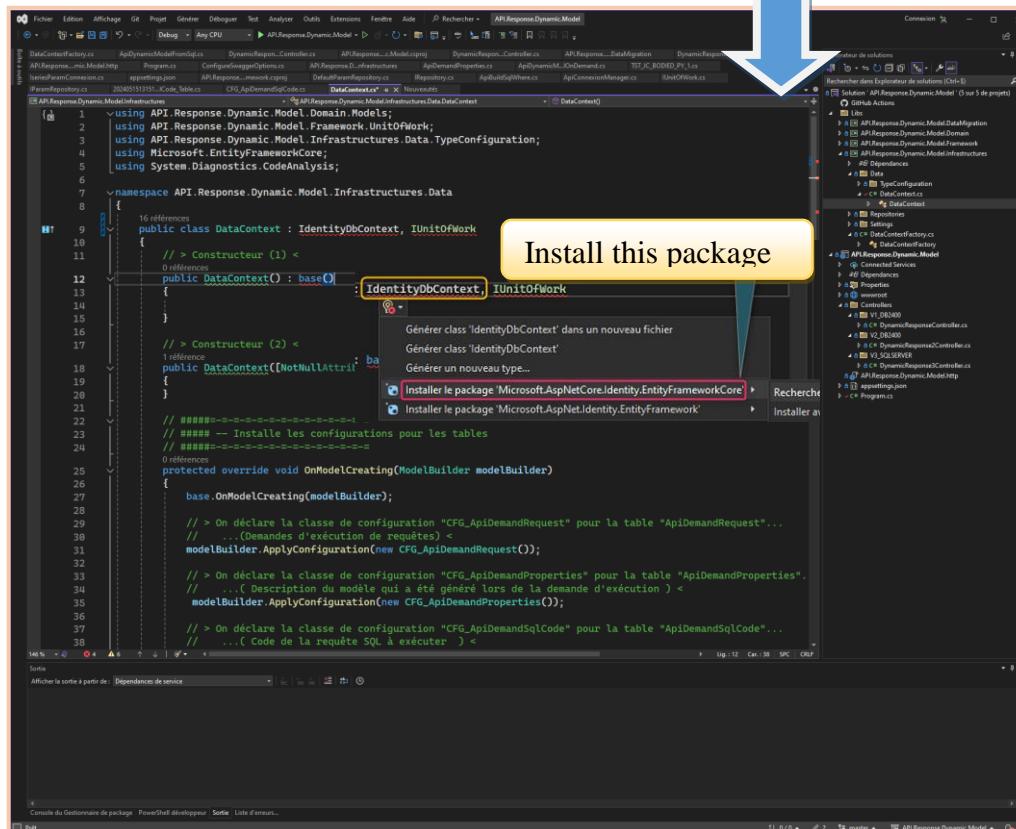
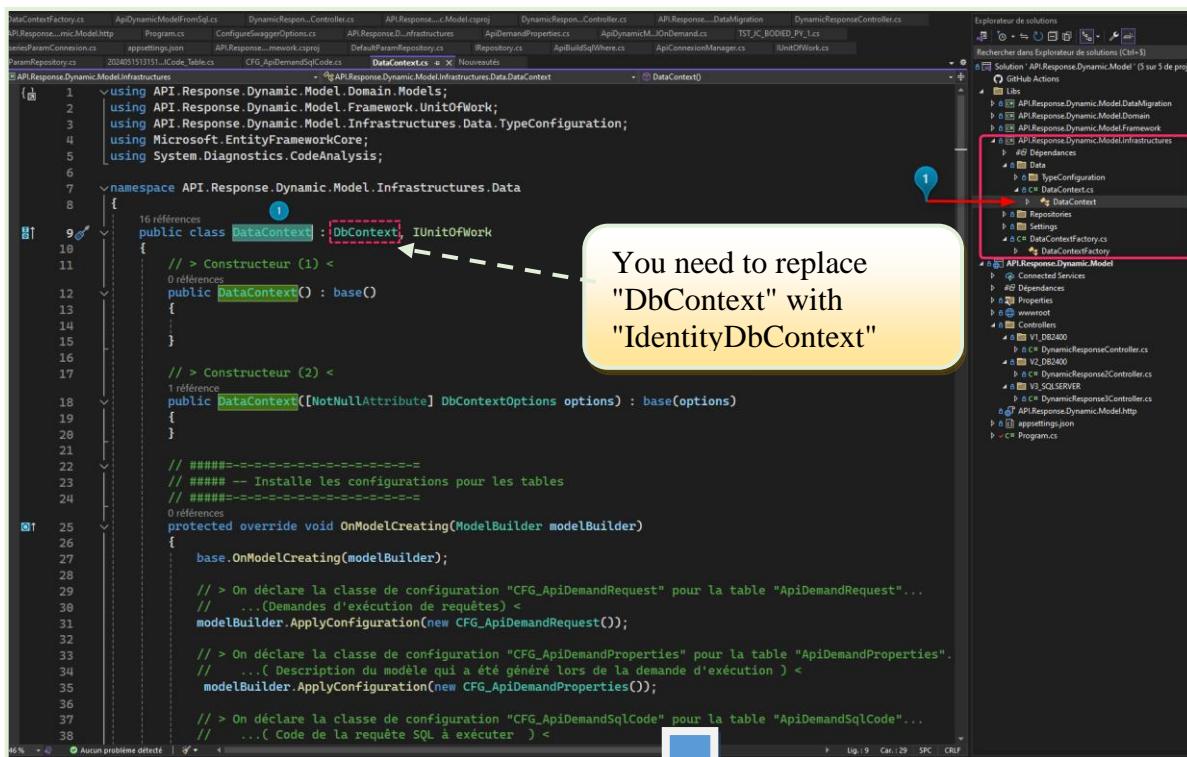
## I – Introduction

- Summary of JWT Authentication implementation

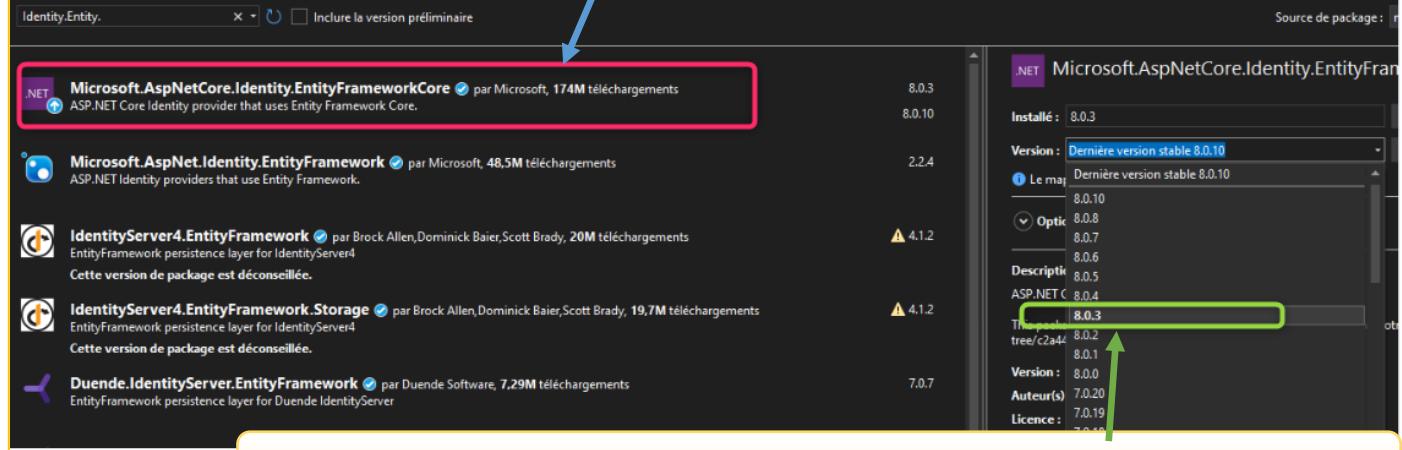


## II – Update the Context: replace “DbContext” with “IdentityDbContext”

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



In the NuGet package manager, choose  
“Microsoft.AspNetCore.Identity.EntityFrameworkCore”

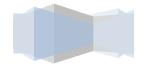
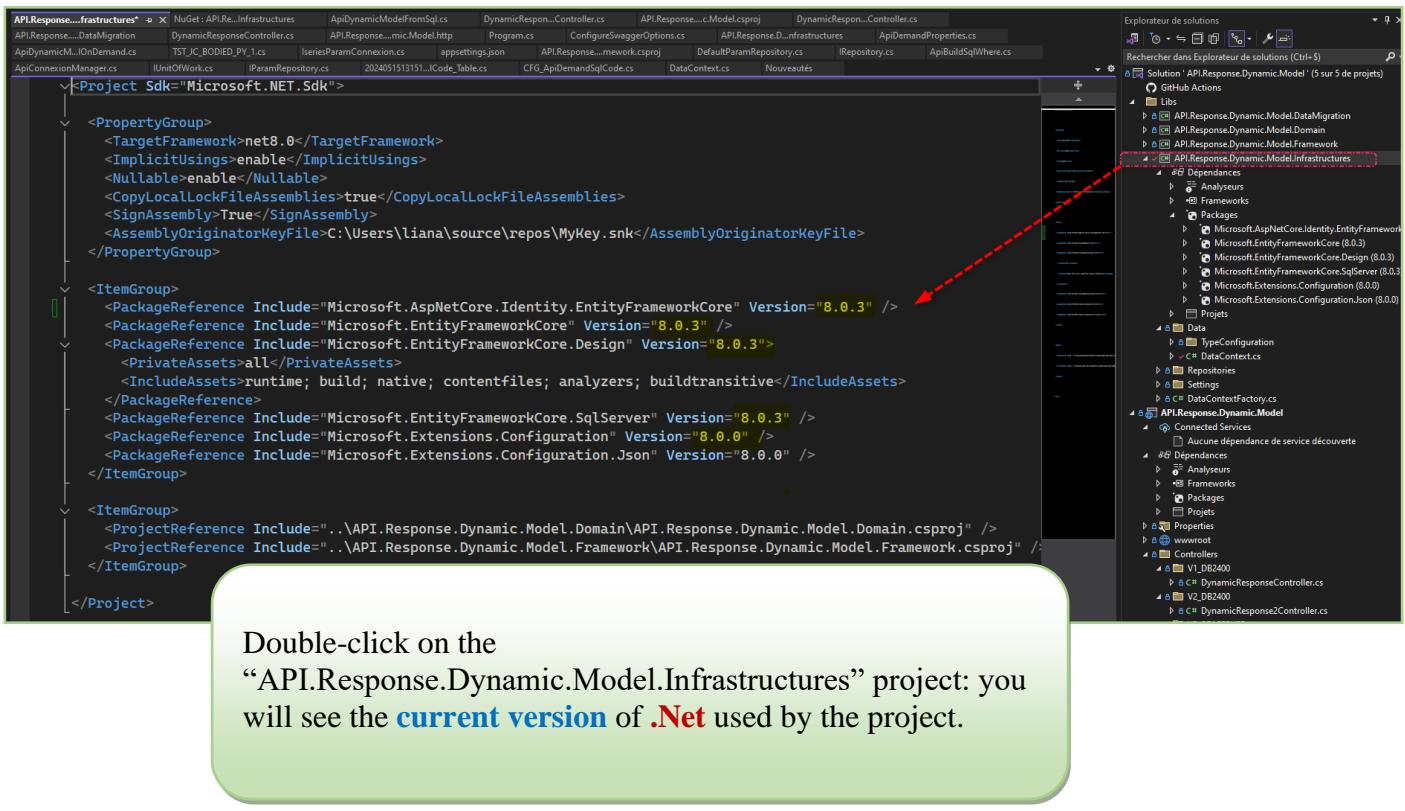


My .Net version is 8.0.3: you need to choose the correct NuGet version from the list.

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

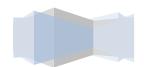
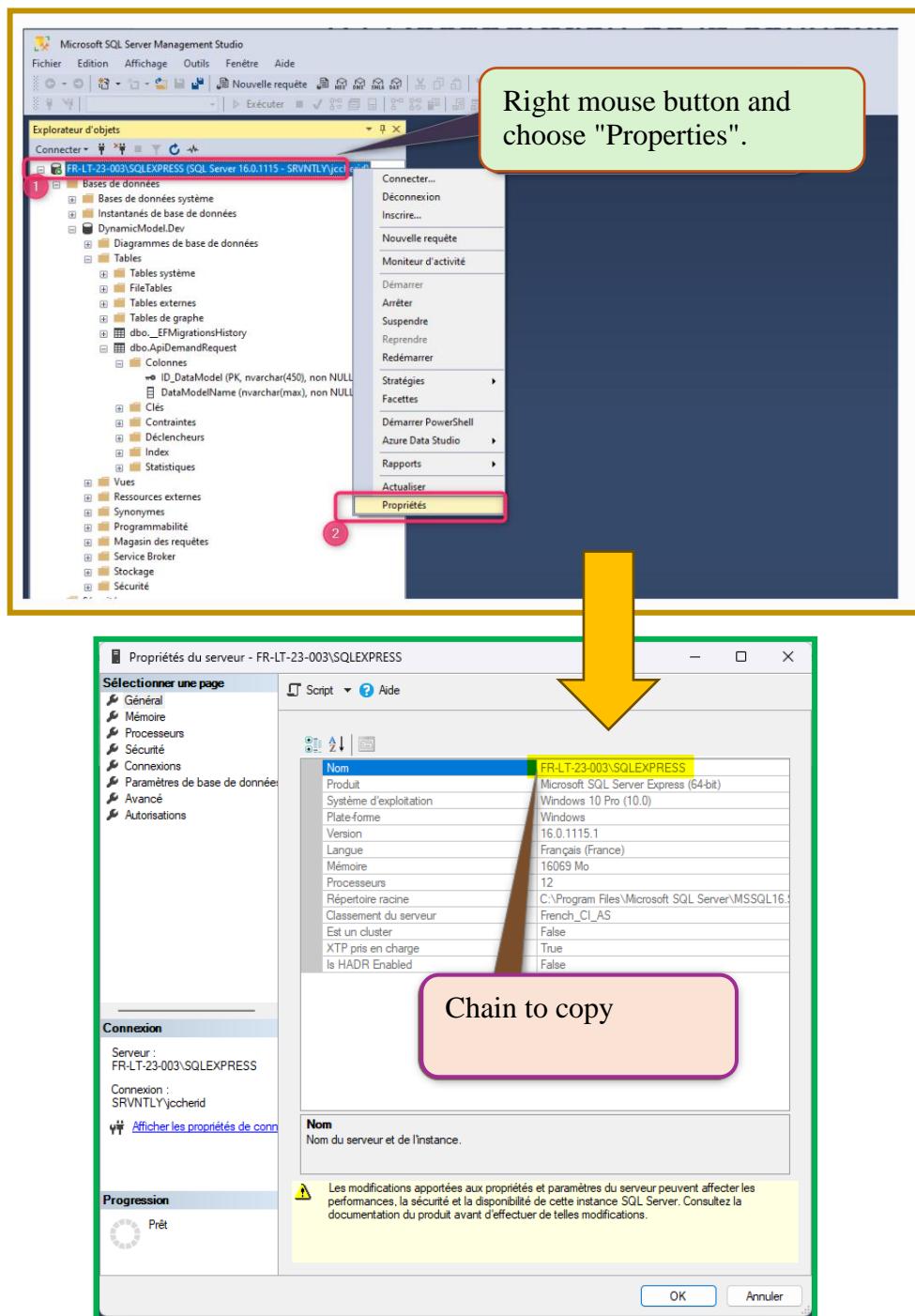
💡 In the “**DataContext**” class, add the missing “**using**”:

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



### 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.

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays `Program.cs` with several sections of C# code. A large yellow box highlights the configuration of the Swagger endpoint and the creation of the database context. A red arrow points from this highlighted area to the `APIResponseDynamic.Model` project in the Solution Explorer on the right. The Solution Explorer lists the following projects and files:

- `APIResponseDynamic.Model` (selected)
- `Libs`
- `APIResponseDynamic.Model.DataMigration`
- `Dépendances`
- `Migrations`
  - `20241031194820_addNewDescription.cs`
  - `20241031194820_addNewDescription.Designer.cs`
  - `20241112192540_NewTables_1A.cs`
  - `20241113125830_NewTables_2A.cs`
  - `20241113125830_NewTables_2A.Designer.cs`
  - `DataContextModelSnapshot.cs`
- `APIResponseDynamic.Model.Domain`
- `APIResponseDynamic.Model.Framework`
- `APIResponseDynamic.Model.Infrastructures`
- `Dépendances`
- `Data`
  - `TypeConfiguration`
    - `CFG_ApiDemandProperties.cs`
    - `CFG_ApiDemandRequest.cs`
    - `CFG_ApiDemandSqlCode.cs`
    - `DataContext.cs`
  - `Repositories`
  - `Settings`
  - `DataContextFactory.cs`
- `APIResponseDynamic.Model`
- `Connected Services`
- `Dépendances`
- `Properties`
- `wwwroot`
- `Controllers`
  - `APIResponseDynamic.Model.http`
- `appsettings.json`
- `appsettings.Development.json`
- `Program.cs`

```
// -----
//> 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.

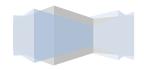
👉 Below is an example of a key:

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

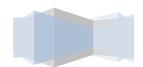
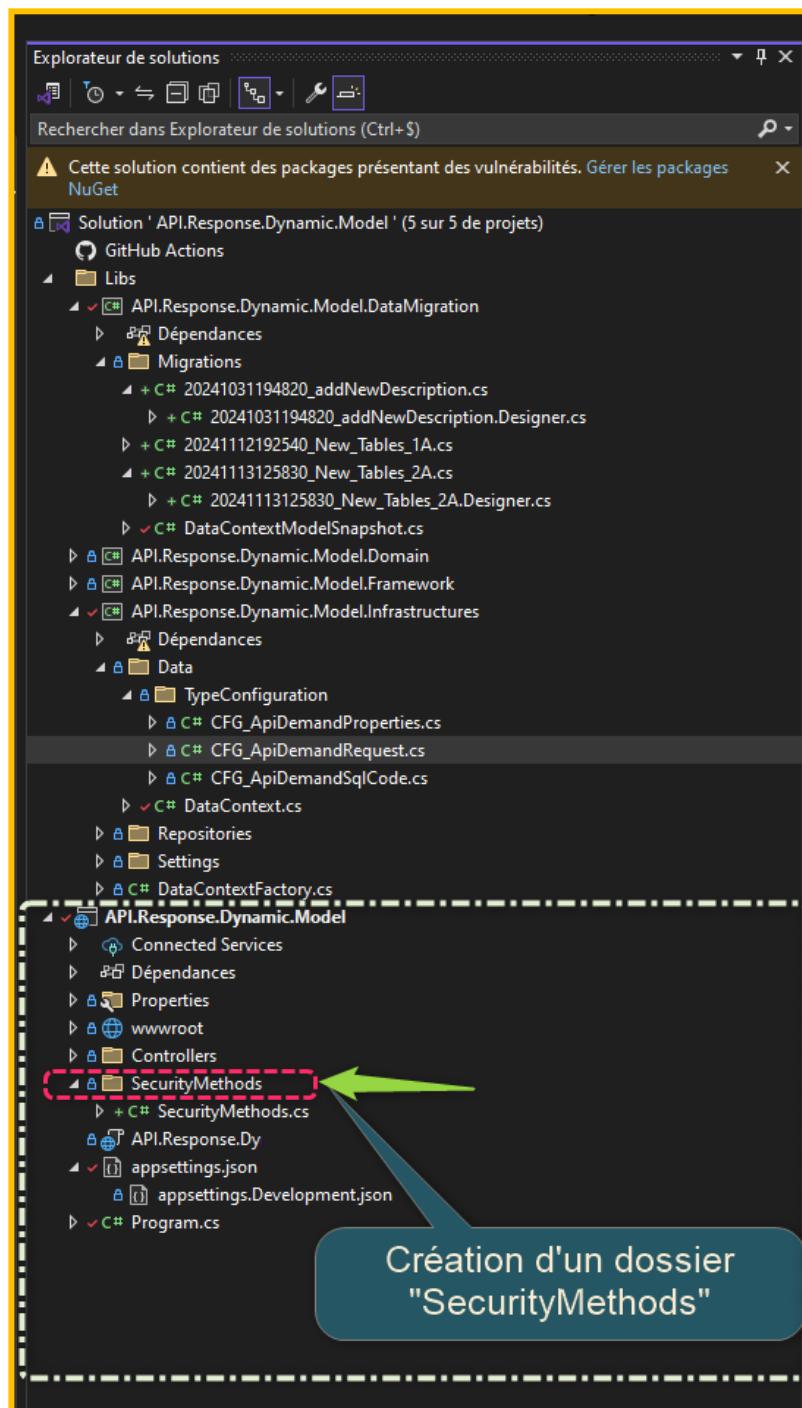
The screenshot shows the Visual Studio interface. On the left, the code editor displays the `appsettings.json` file with the following content:

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8
9    "ConnectionStrings": {
10      "PROD_BDD": "Server=LAPTOP-8MU46MIG\SQLEXPRESS;Database=DynamicModel.Dev;Trusted_Connection=True;Encrypt=true; TrustServerCertificate=true;"
11    },
12
13    "Jwt": {
14      "key": "A4e78145A369874AAjrt@128545!p@AyQ"
15    },
16
17    "AllowedHosts": "*"
18  }
```

A red dashed box highlights the `Jwt` section, and a red arrow points from this box to the `appSettings.json` file in the Solution Explorer on the right. The Solution Explorer shows the project structure, including files like `AuthenticatedUser.cs`, `AuthenticatedUserDto.cs`, `AuthController.cs`, `DynamicResponseController.cs`, `APIResponseAssemblyInfo.cs`, `SecurityMethods.cs`, `ApiDemandRequest.cs`, `ApiDemandProperties.cs`, `appSettings.Development.json`, and `DataContext.cs`.



## IV-B Create a “SecurityMethods” folder



## IV-C Install the “System.IdentityModel.Tokens.JWT” package

The screenshot shows the jwt.io homepage. At the top, there's a navigation bar with links for Debugger, Libraries (which is highlighted with a red box), Introduction, and Ask. A call-to-action button labeled "Cliquer ici" (Click here) points to the Libraries section. Below the navigation, there's a yellow starburst icon with an exclamation mark containing the text: "Le site JWT.IO offre une liste de packages à installer pour la mise en place d'un JSON Web Tokens.". To the right of this is a colorful circular logo. The main content area contains text about JSON Web Tokens and their RFC 7519 standard, followed by a statement that JWT.IO allows decoding, verifying, and generating JWTs. At the bottom, there are two buttons: "LEARN MORE ABOUT JWT" and "SEE JWT LIBRARIES". A large green downward arrow is centered below these buttons.

The screenshot shows the "Libraries" section of jwt.io. It displays three cards for .NET packages:

- .NET**: Microsoft.IdentityModel.Tokens.Jwt (907 stars). A callout bubble says "On va utiliser le package microsoft.". The package page includes a GitHub link to Microsoft's repository and a command to install it via NuGet: `Install-Package System.IdentityModel.Tokens.Jwt`.
- .NET**: JWT.NET (1939 stars). The package page includes a GitHub link to Alexander Batishchev's repository and a command to install it via NuGet: `Install-Package JWT.NET`.
- .NET**: jose-jwt (836 stars). The package page includes a GitHub link to DV's repository and a command to install it via NuGet: `Install-Package jose-jwt`.



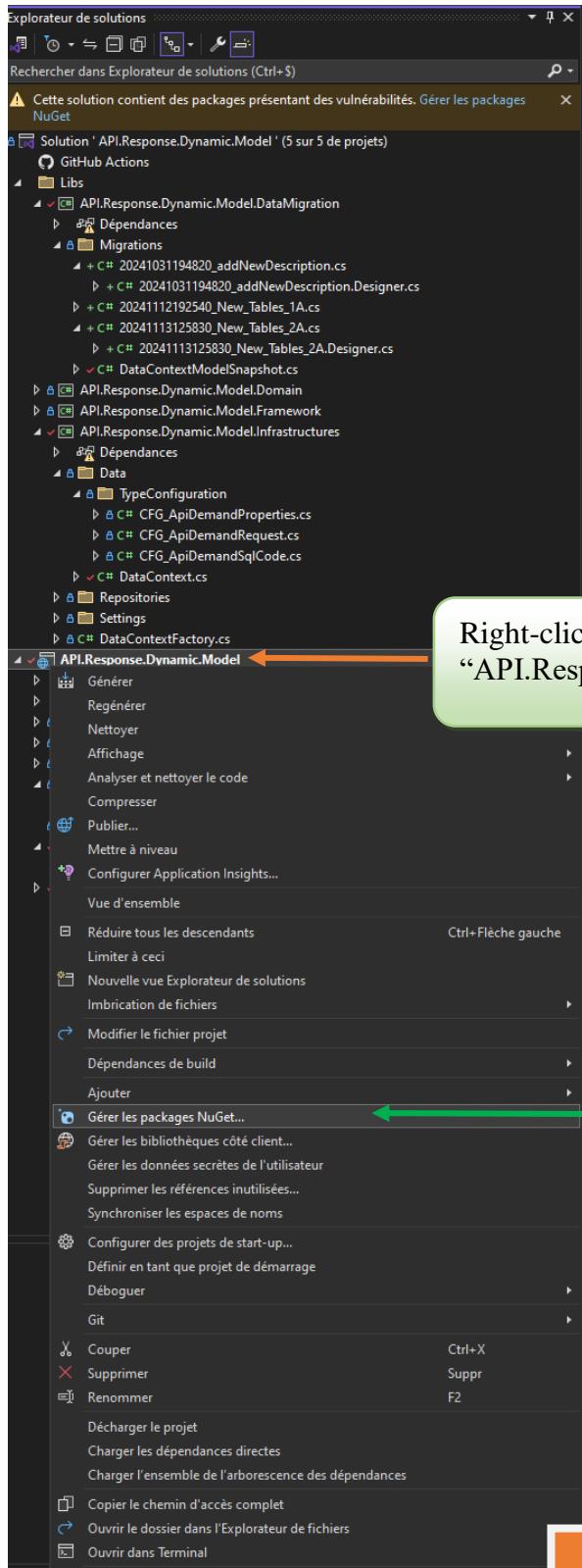
💡 When you click on “**Install-Package System.IdentityModel.Tokens.Jwt**”, you arrive on the page below:

✓ Link : <https://www.nuget.org/packages/System.IdentityModel.Tokens.Jwt/>

The screenshot shows the NuGetGallery page for the **System.IdentityModel.Tokens.Jwt** package. The package is version 8.2.0, released on March 13, 2024. It has 576.6K downloads per day on average. The page includes sections for **About**, **Owners**, and **Tags**. A callout box highlights the package details section, which contains the following text:

- 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 screenshot shows the NuGet package manager interface and the Solution Explorer. A yellow arrow points from the NuGet search results to the Solution Explorer, where a red arrow highlights the newly added package.

**NuGet : API.Response.Dynamic.Model**

- SecurityMethods.cs\*
- API.Response.Dynamic.Model.csproj\*
- 2024111312583...Tables\_2A.cs
- 2024111312583...Tables\_1A.cs
- ApiDemandRequest.cs
- ApiDemandProperties.cs
- appsettings.json
- DataContext.cs
- Program.cs
- API.Response.Dynamic.Model.csproj
- API.Response.Dynamic.Model.http
- appsettings.Development.json
- Program.cs
- CFG\_ApiDemandRequest.cs
- Nouveautés
- CFG\_ApiDemandRequest.cs

Parcourir    Installé    Mises à jour

System.IdentityModel.Tokens.Jwt

Gestionnaire de package NuGet : API.Response.Dynamic.Model

Source de package : nuget.org

**System.IdentityModel.Tokens.Jwt** par AzureAD, Microsoft, 1,98B téléchargements

includes types that provide support for creating, serializing and validating JSON Web Tokens. As of IdentityModel 7x, this is a legacy tool that should be replaced with Microsoft.IdentityModel.JsonWebTokens.

8.2.0

**Microsoft.IdentityModel.JsonWebTokens** par AzureAD, Microsoft, 1,85B téléchargements

Includes types that provide support for creating, serializing and validating JSON Web Tokens. This is a newer, faster version of System.IdentityModel.Tokens.Jwt that has additional functionality.

8.2.0

**System.IdentityModel.Tokens.ValidatingIssuerNameRegistry** par AzureAD, Microsoft, 4,85M téléchargements

This package provides an assembly containing classes which extend the .NET Framework 4.5 with the necessary logic that extends token validation to check that the signer of a token and the issuer of the token are a valid pair. This capability can be applied both within the...

4.5.1

**Thinktecture.IdentityModel.SystemWeb** par thinktecture, 158K téléchargements

Helper library for claims based identity & access control in ASP.NET and MVCs.

1.1.0

**System.IdentityModel.Extensions.UsernameTokenHandler** par Ahmed.Mabrouk, 11,7K téléchargements

Custom Windows Identity Foundation 4.5 UsernameTokenHandler Base Class That Can Authenticate and Authorize UsernameToken By Implementing 2 Abstract Methods:

1.0.1

**IdentityModel.Unofficial** par cuteant, 36,2K téléchargements

Helper library for claims-based identity, System.IdentityModel, JSON Web Tokens, OAuth 2.0 and OpenID Connect client

2.8.1

**HiNetCloud.System.IdentityModel.Tokens.Jwt** par jszyong, 12K téléchargements

23.10.10.15

**Beiyuan.WebSystem.IdentityModel** par tanson, 2,63K téléchargements

Package Description

3.1.0

**Beiyuan.WebSystem.Http.Client.IdentityModel** par tanson, 1,83K téléchargements

Package Description

3.1.0

**M365.System.IdentityModel** par yag, 833 téléchargements

Package Description

1.0.0

**Beiyuan.WebSystem.Http.Client.IdentityModel.Web** par tanson, 1,36K téléchargements

Package Description

3.1.0

**Explorateur de solutions**

Rechercher dans Explorateur de solutions (Ctrl+S)

Solution ' API.Response.Dynamic.Model ' (5 sur 5 de projets)

GitHub Actions

Libs

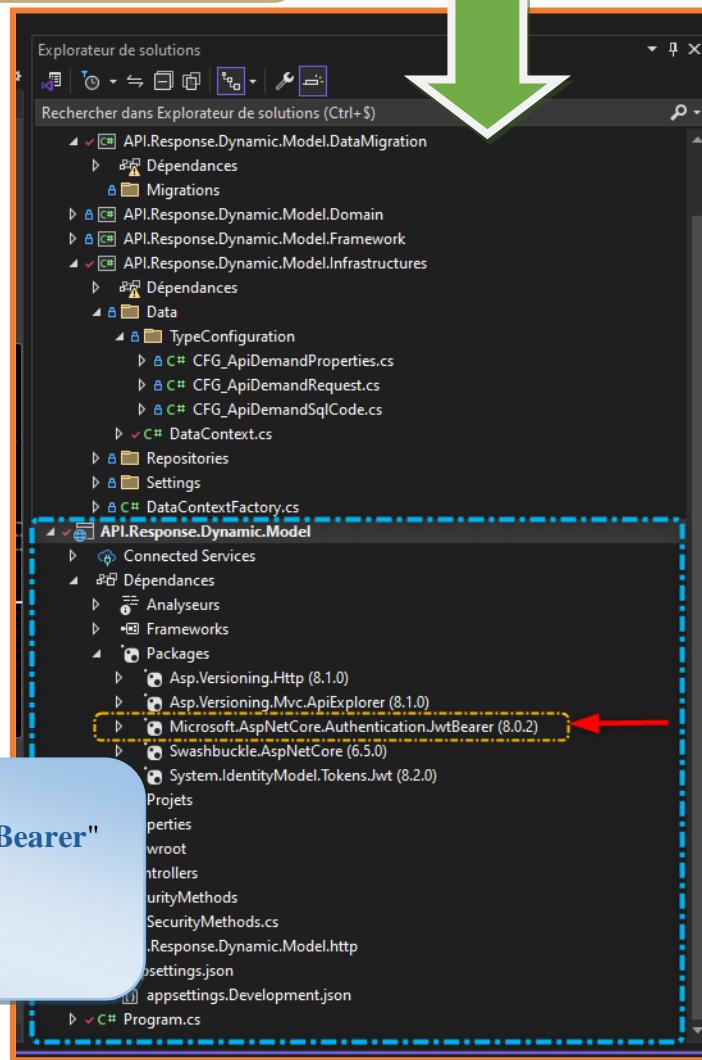
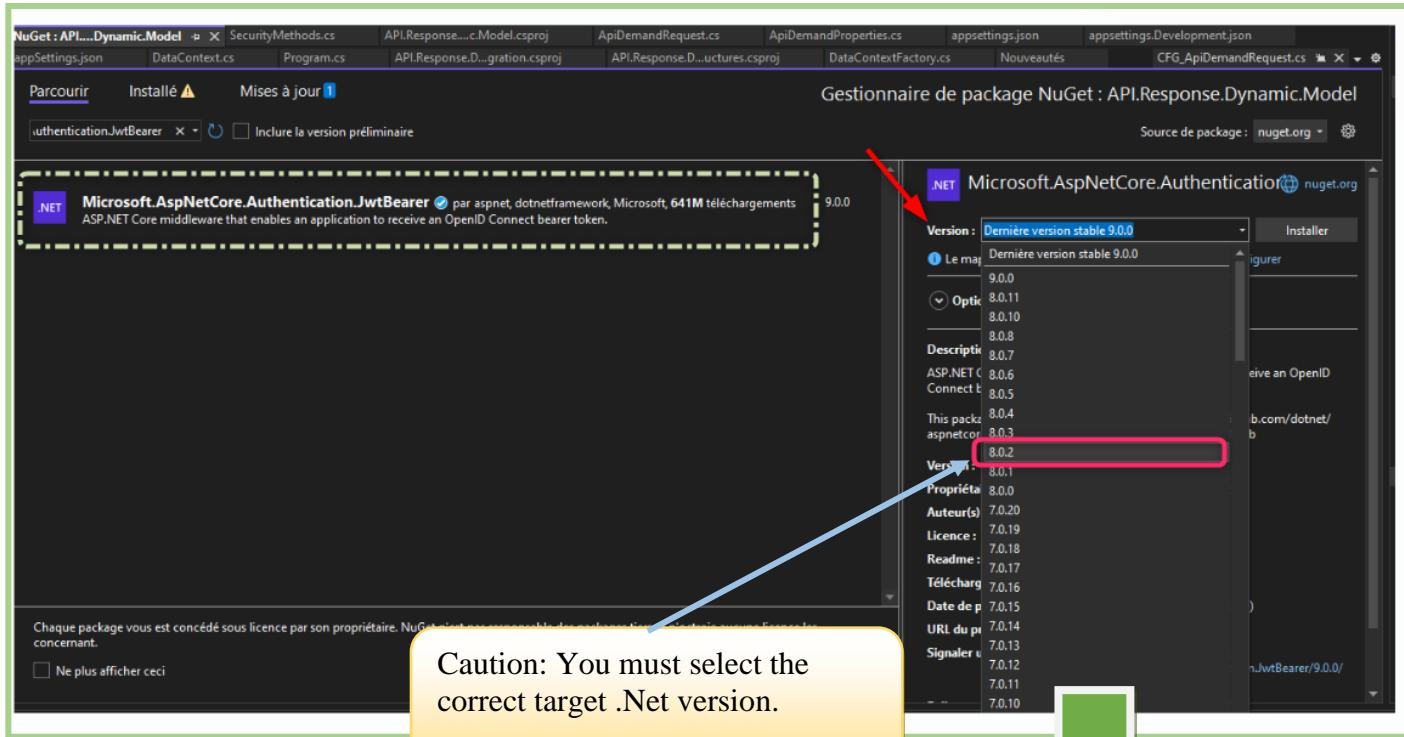
- API.Response.Dynamic.Model.DataMigration
- Dépendances
- Migrations
- API.Response.Dynamic.Model.Domain
- API.Response.Dynamic.Model.Framework
- API.Response.Dynamic.Model.Infrastructures
- Dépendances
- Data
- TypeConfiguration
- CFG\_ApiDemandProperties.cs
- CFG\_ApiDemandRequest.cs
- CFG\_ApiDemandSqlCode.cs
- DataContext.cs
- Repositories
- Settings
- DataContextFactory.cs

API.Response.Dynamic.Model

- Connected Services
- Dépendances
- Analysateurs
- Frameworks
- Packages
- Asp.Versioning.Http (8.1.0)
- Asp.Versioning.Mvc.ApiExplorer (8.1.0)
- Swashbuckle.AspNetCore (6.5.0)
- System.IdentityModel.Tokens.Jwt (8.2.0)
- Projets
- Properties
- wwwroot
- Controllers
- SecurityMethods
- SecurityMethods.cs
- API.Response.Dynamic.Model.http
- appsettings.json
- appsettings.Development.json
- Program.cs

The package "System.IdentityModel.Tokens.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

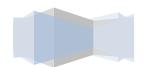
The screenshot shows the Visual Studio IDE with the 'SecurityMethods.cs' file open in the code editor. The file contains C# code for a static class 'SecurityMethods' within the 'API.Response.Dynamic.Model.SecurityMethods' namespace. The class includes methods for handling JWT authentication, such as 'AddCustomAuthentication' which uses a builder pattern to configure a 'ConfigurationBuilder'. The 'Explorateur de solutions' (Solution Explorer) window on the right shows the project structure for 'API.Response.Dynamic.Model', which contains several sub-projects like 'Libs' and 'Domain'. A yellow callout box highlights the 'SecurityMethods' folder in the Solution Explorer.

```

1 // > Authentication JWT <
2 using Microsoft.AspNetCore.Authentication.JwtBearer;
3 using Microsoft.IdentityModel.Tokens;
4
5
6 namespace API.Response.Dynamic.Model.SecurityMethods
7 {
8     0 références
9     public static class SecurityMethods
10    {
11        #region properties
12        //> Déclare clé pour la construction du Token <
13        2 références
14        private static string MyKeyJwt { get; set; }
15
16        //> Crée une variable "args" de type tableau de string nullable (<)
17        // ( pour " var builder = WebApplication.CreateBuilder(args) "
18        // ( Remarque : on peut l'appeler autrement que "args" si on veut )
19        1 référence
20        private static string[]? args { get; set; }
21
22        #endregion
23        /// <summary>
24        /// Construction du processus d'authentification JWT
25        /// </summary>
26        /// <param name="_builder"></param>
27        /// <returns></returns>
28        0 références
29        public static object AddCustomAuthentication( object _builder )
30        {
31            //> 1/ Création d'un "configurationBuilder" <
32            ConfigurationBuilder configurationBuilder = new ConfigurationBuilder();
33
34            //> 2/ Création d'un objet "Configuration" à partir du...
35            //> ... "configurationBuilder" <
36            IConfiguration configuration = configurationBuilder.Build();
37
38            //> 3/ Récupération de la clé d'authentification <
39            //> ( On récupère la clé depuis "appsettings.json" )
40            MyKeyJwt = configuration["Jwt:key"];
41
42            //> 4/ On déclare une variable "builder" de type...
43            //> ... "WebApplicationBuilder" <
44            var builder = WebApplication.CreateBuilder(args);
45
46
47            //> 5 Si le paramètre transmis est non nul, alors on récupère son contenu <
48            if ( _builder is not null )
49            {
50                //> 6 On récupère la clé "MyKeyJwt" <
51                MyKeyJwt = _builder.Configuration["Jwt:key"];
52            }
53
54        }
55
56        //> 7 On déclare une méthode "AddCustomAuthentication" <
57        public static void AddCustomAuthentication( WebApplicationBuilder builder )
58        {
59            builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
60            .AddJwtBearer(options =>
61            {
62                options.TokenValidationParameters = new TokenValidationParameters
63                {
64                    ValidateIssuerSigningKey = true,
65                    IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(MyKeyJwt)),
66                    ValidateAudience = false,
67                    ValidateLifetime = true
68                };
69            });
70        }
71
72    }
73
74}

```

The static class "SecurityMethods" is in the "SecurityMethods" folder of the "API.Response.Dynamic.Model" project



## 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...
            // ... "WebApplicationBuilder" <
            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

```

137     Description = "En-tête d'autorisation JWT utilisant le schéma Bearer.",
138     Name = "Authorization",
139     In = ParameterLocation.Header,
140     Type = SecuritySchemeType.ApiKey
141   );
142
143   // > Ajout Authentification JWT 2/2 <
144   options.AddSecurityRequirement(new OpenApiSecurityRequirement
145   {
146     {
147       new OpenApiSecurityScheme
148       {
149         Reference = new OpenApiReference
150         {
151           Type = ReferenceType.SecurityScheme,
152           Id = "Bearer"
153         },
154         Scheme = "oauth2", Name = "Bearer",
155         In = ParameterLocation.Header,
156       },
157       new List<string>()
158     },
159   });
160 }
161
162 });
163
164 // -----
165 // Configuration du VERSIONNING de l'API ( FIN )
166 // -----
167
168
169
170 // === *** JWT .Net 8 ***
171 // -----
172
173 SecurityMethods.AddCustomAuthentication(builder);
174
175
176 // > Construction de l'objet builder <
177 var app = builder.Build();

```

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 “SecurityMethods” class.

### IV-F-2 Line of code

```

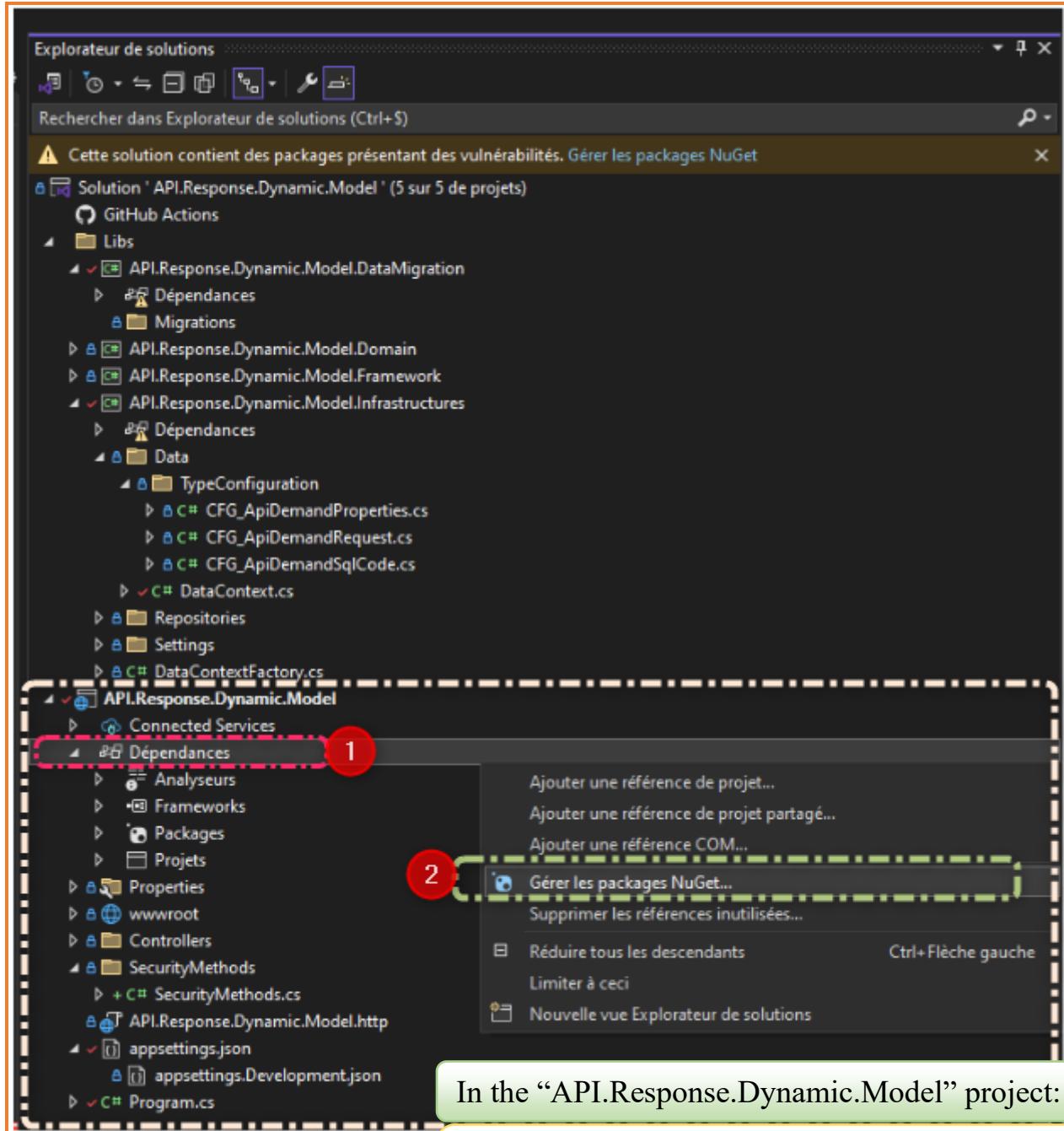
// =====
// ### *** JWT .Net 8 ***
// =====
SecurityMethods.AddCustomAuthentication(builder);

```

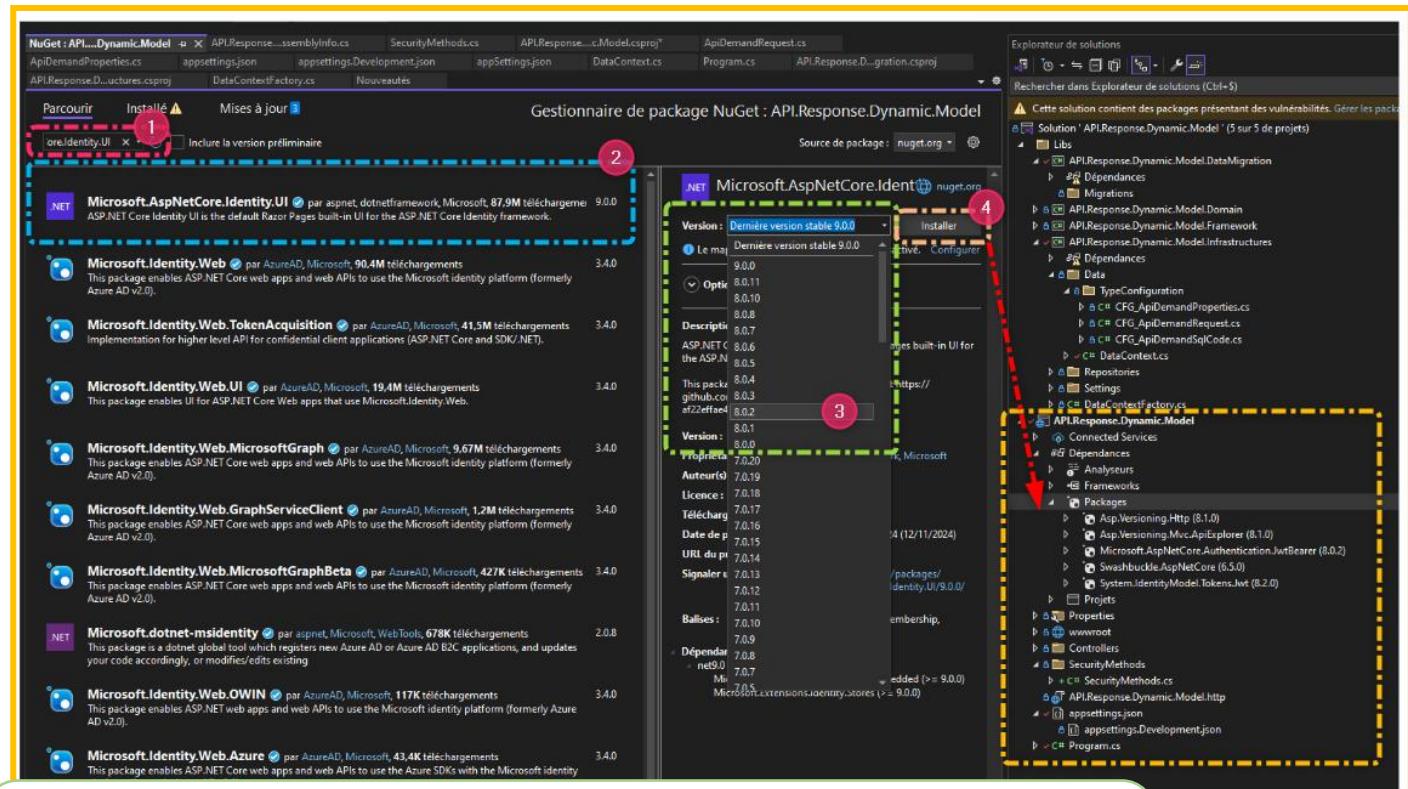


## V Configuring the “Context Identity”

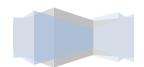
### V-A Installing the NuGet “Microsoft.AspNetCore.Identity.UI”



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

The screenshot shows the 'Program.cs' file in the 'APIResponse.Dynamic.Model' project. The code has been inserted into the builder.Services.AddDefaultIdentity block. A callout box points to the inserted code with the text: 'The "Program.cs" class with the inserted code.'

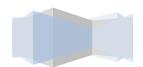
```
160     new List<string>()
161     );
162   });
163 }
164 // -----
165 // Configuration du VERSIONNING de l'API ( FIN )
166 // -----
167
168 // =====
169 // ### *** JWT .Net 8 ***
170 // ====
171 // SecurityMethods.AddCustomAuthentication(builder);
172
173 // =====
174 // > On demande à se ce que le mot de passe...
175 // ...contienne des chiffres <
176 options.Password.RequireDigit = true;

177 // > On demande à se ce que le mot de passe...
178 // ...contienne des majuscules <
179 options.Password.RequireUppercase = true;

180 // > On demande à se ce que le mot de passe...
181 // ...contienne minimum 12 caractères <
182 options.Password.RequiredLength = 12;

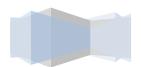
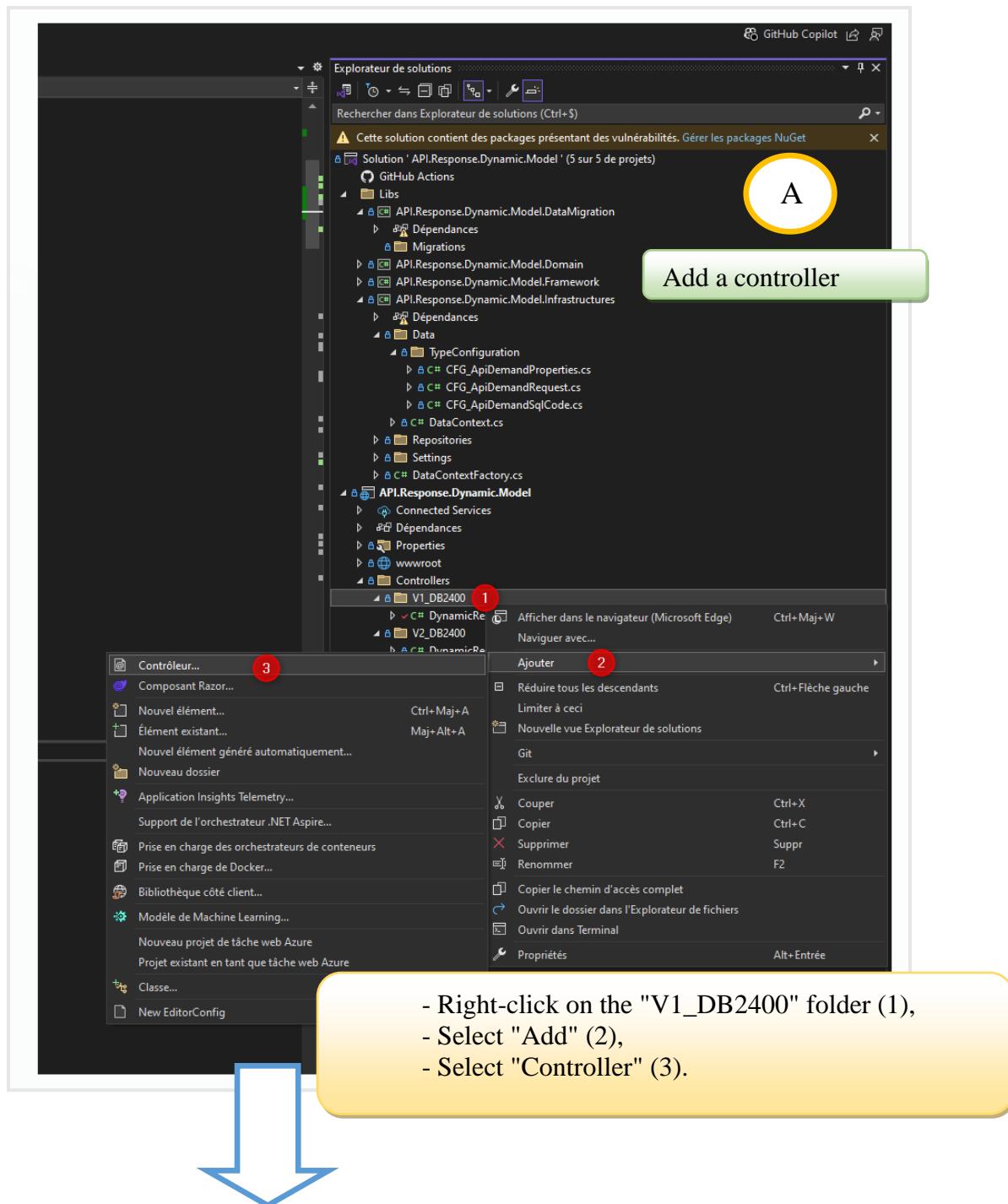
183 // Association avec notre Context pour effectuer l'authentification
184 }).AddEntityFrameworkStores<DataContext>();
185
186 // > Construction de l'objet builder <
187 var app = builder.Build();
```

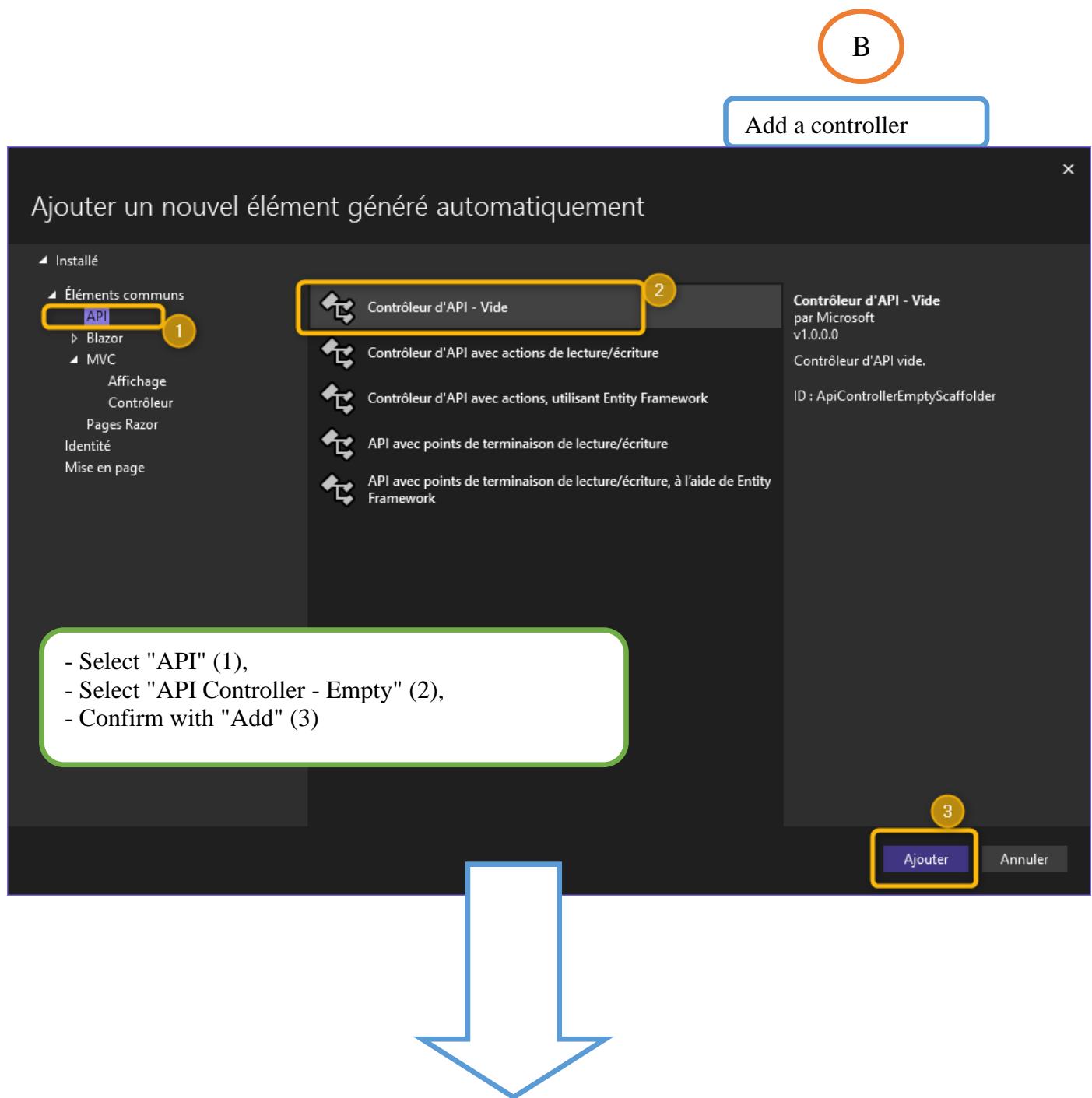
The “Program.cs” class with the inserted code.

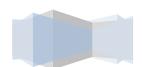
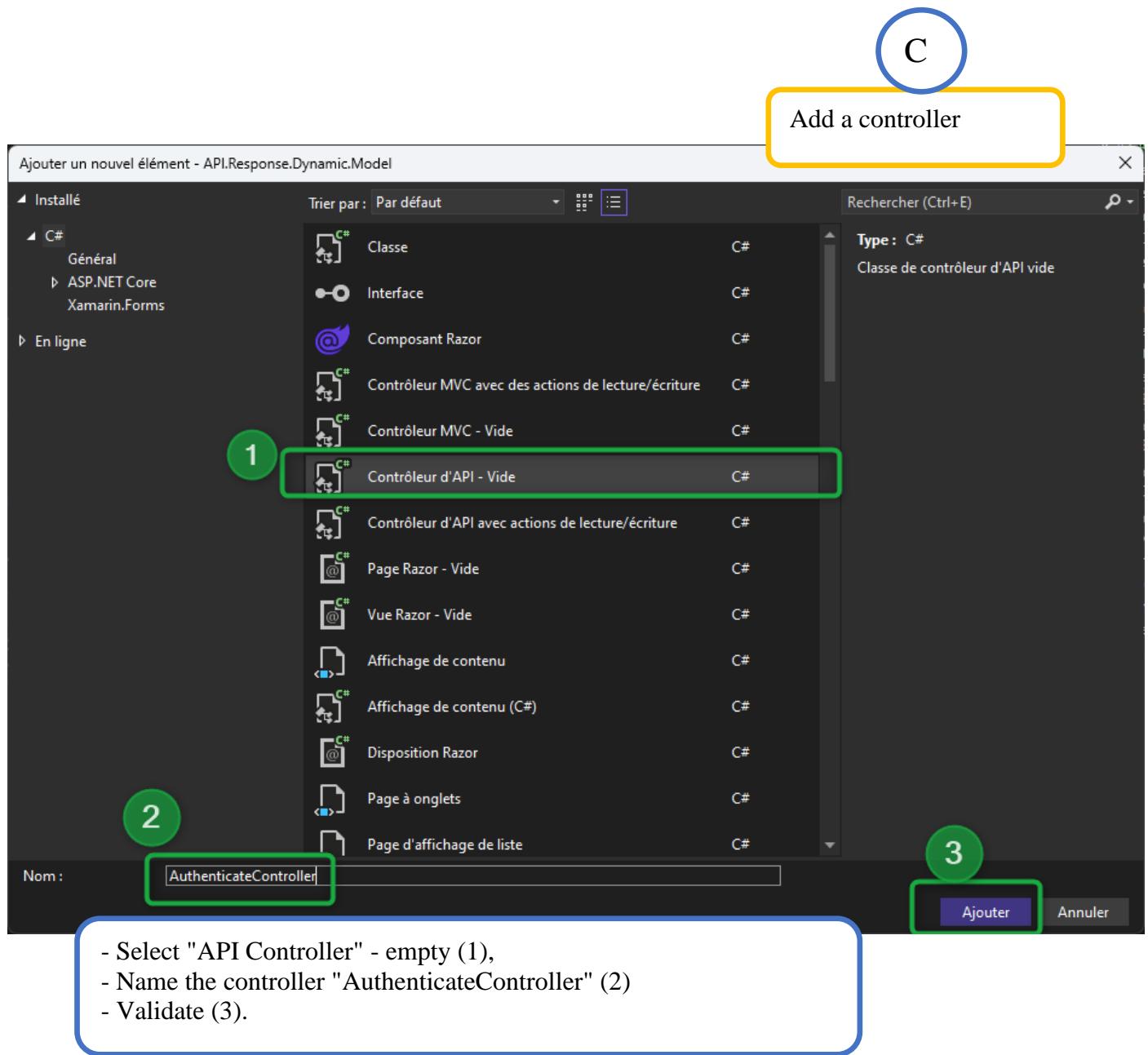


## VI Added the “AuthenticateController” controller

### VI-A Procedure for adding the controller







## VI-B Declaring variables in the “AuthenticateController” controller (with versioning syntax)

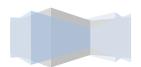
- ⚠ In this version of the code, I use versioning (\"using Asp.Versioning\").
- 👉 See below for the version without versioning

```
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
}
}

```



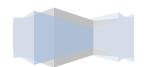
## VI-E The constructor's code as it should appear

```

1  using Asp.Versioning;
2  using Microsoft.AspNetCore.Http;
3  using Microsoft.AspNetCore.Identity;
4  using Microsoft.AspNetCore.Mvc;
5
6  namespace API.Response.Dynamic.Model.Controllers.V1_DB2400
7  {
8      [ApiController]
9      [ApiVersion("1.0")]
10     [Route("api/v{version:apiVersion}/[controller]")]
11     1 référence
12     public class AuthenticateController : ControllerBase
13     {
14         #region fields
15         // - - - - -
16         // *** Authentication Début ***
17         // - - - - -
18         // > Déclare un "UserManager" <
19         private UserManager<IdentityUser> _userManager = null;
20
21         // > Déclare variable pour extraire la clé de "Jwt:Key" <
22         private string ElemKey;
23
24         // Tableau de strings utilisé pour la construction...
25         // ... de l'objet "host"
26         private string[] Elems = null;
27
28         // *** Authentication Fin ***
29         // - - - - -
30         #endregion
31
32         #region Constructor
33         // > Constructeur <
34         // Remarque : On récupère "userManager" par injection de dépendance <
35         public AuthenticateController(UserManager<IdentityUser> userManager)
36         {
37             // > Chargement du Paramètre <
38             this._userManager = userManager;
39
40             // > On construit un objet "Host" pour pouvoir...
41             // ...manipuler un "appsettings.json" <
42             using IHost host = Host.CreateDefaultBuilder(Elems).Build();
43             IConfiguration config = host.Services.GetService<IConfiguration>();
44
45             // > On extrait la valeur de la clé "JWT:Key" depuis le "AppSettings" <
46             ELEMKEY = config.GetValue<string>("Jwt:Key");
47         }
48         #endregion
49     }
50

```

The "AuthenticateController" controller code (versioning mode)

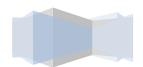
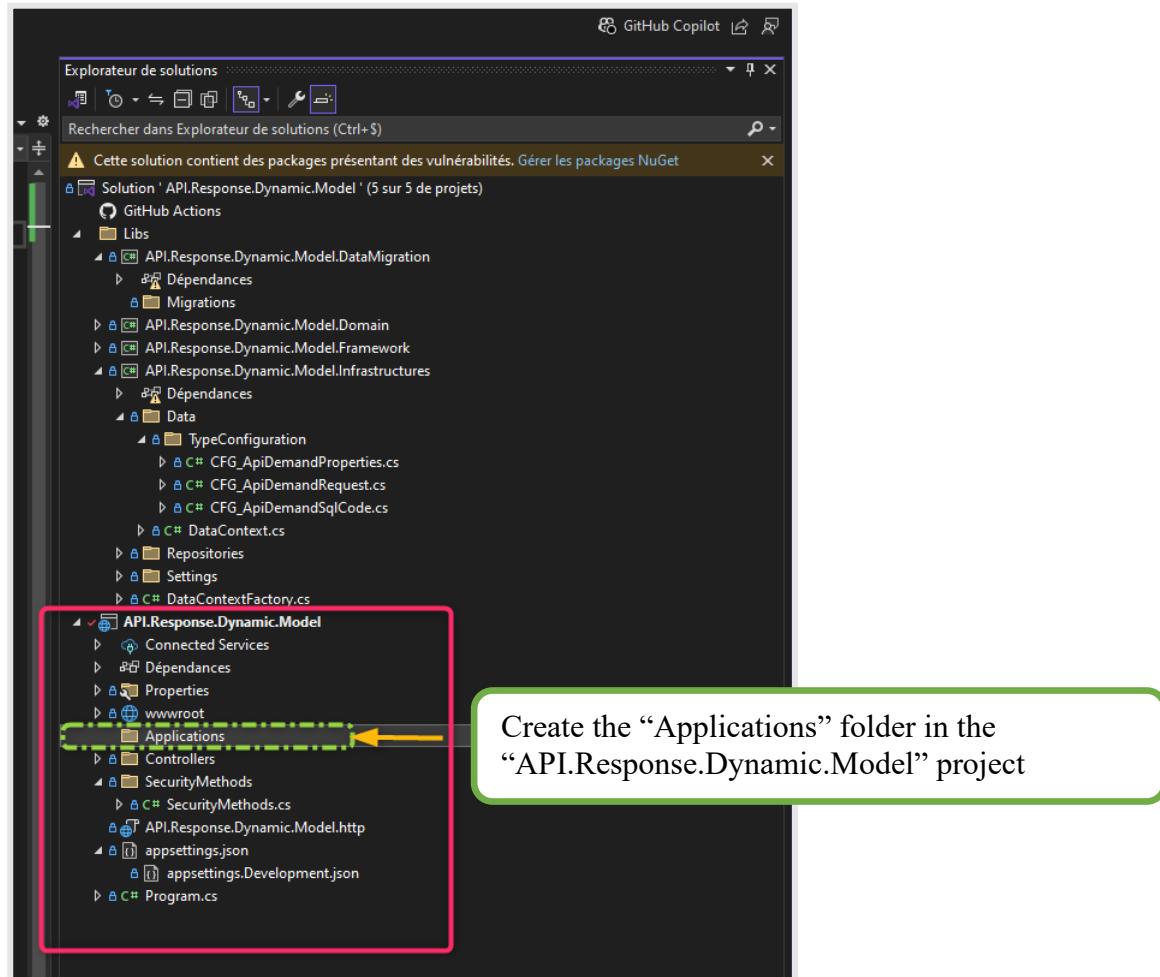


## VII Added the Dto (Data Transfer Object) "AuthenticateUserDto"

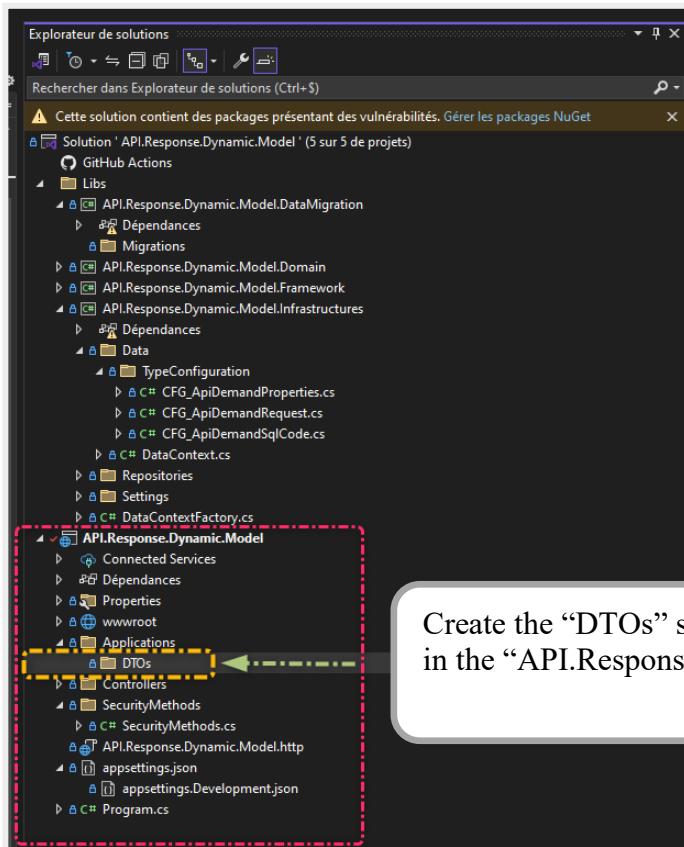
⚠ **Note:** The "AuthenticateUserDto" class contains all the login information.

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

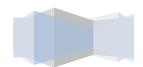
### VII-A Creating an “Applications” folder



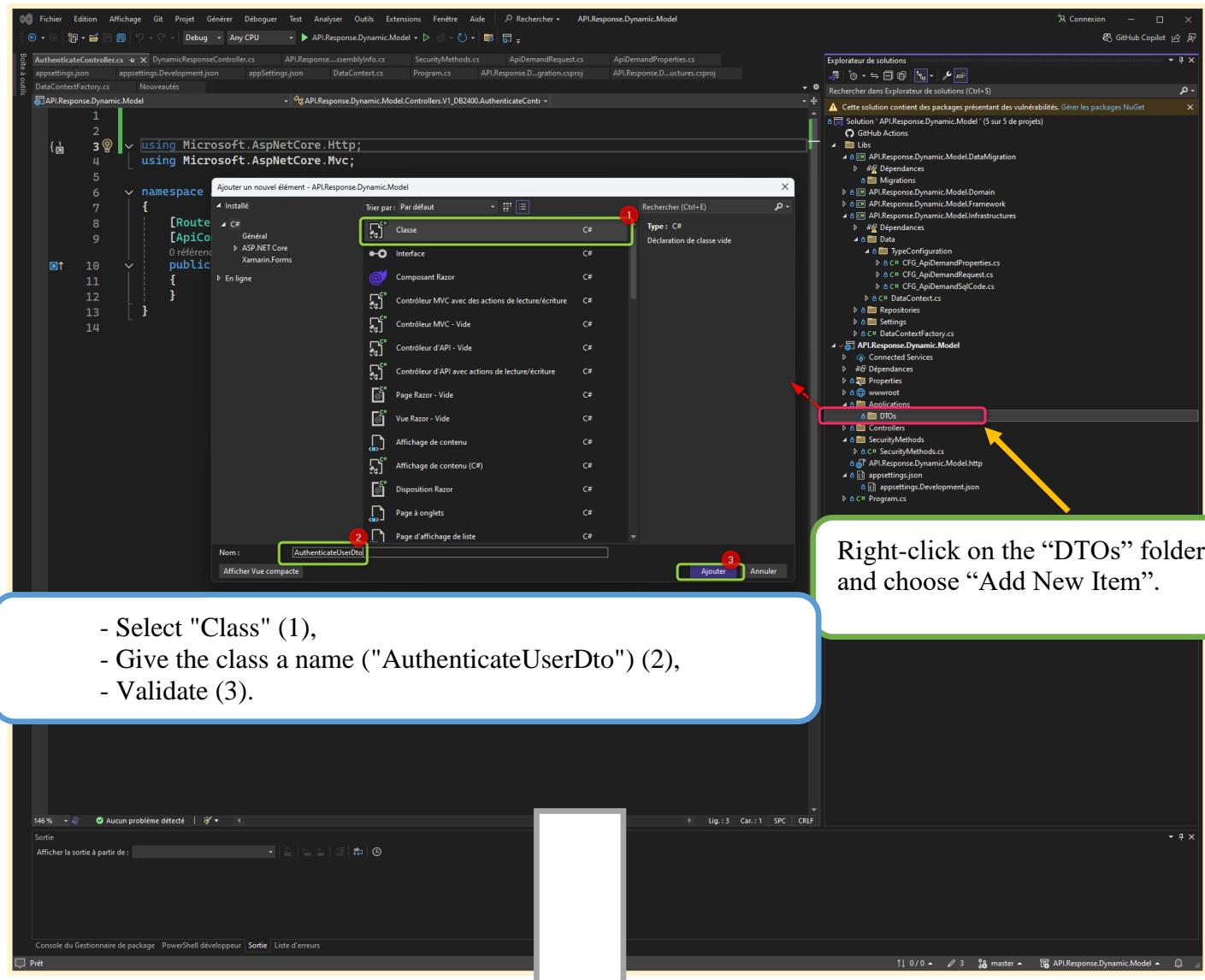
## VII-B Creating a “DTOs” folder



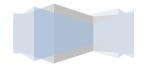
Create the “DTOs” subfolder in the “Applications” folder, still in the “API.Response.Dynamic.Model” project.



## VII-C Added the “AuthenticateUserDto” DTO



Right-click on the “DTOs” folder, and choose “Add New Item”.



```
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);

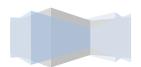
    if (success.Succeeded)
    {
        try
        {
            // > We retrieve the Token calculated by the "GenerateJwtToken" method <
            userDto.Token = this.GenerateJwtToken(user);

            // > 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.ErrorMsg = sb.ToString();
    result = this.BadRequest(userDto);
}

return result;
}

```

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various database objects like System Tables, External Tables, and the AspNetUsers table under the dbo schema. The main pane displays a query results grid. The query executed is:

```

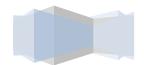
SELECT TOP (1000) [Id]
    ,[UserName]
    ,[NormalizedUserName]
    ,[Email]
    ,[NormalizedEmail]
    ,[EmailConfirmed]
    ,[PasswordHash]
    ,[SecurityStamp]
    ,[ConcurrencyStamp]
    ,[PhoneNumber]
    ,[PhoneNumberConfirmed]
    ,[TwoFactorEnabled]
    ,[LockoutEnd]
    ,[LockoutEnabled]
    ,[AccessFailedCount]
FROM [DynamicModel.Dev].[dbo].[AspNetUsers]

```

The results grid shows one row with the following values:

		User Name	Normalized User Name	Email	Normalized Email	Email Confirmed	Password Hash
1	d2519726-ed6a-4c72-b260-9fc686edc237	string	STRING			0	AQAAAIAAYagAAAAElkrgDK00xE2XMo0cBc2Rx0nJTuG...Kf

A yellow callout bubble with the text "The password is encrypted by the Framework." is positioned over the Password Hash column.



## 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:

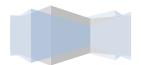
```
private string GenerateJwtToken(IdentityUser user)
{
    // We define the Jwt token which will be responsible...
    // ...of the creation of our Token <
    var jwtTokenHandler = new JwtSecurityTokenHandler();

    // > 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;
    }
}
```



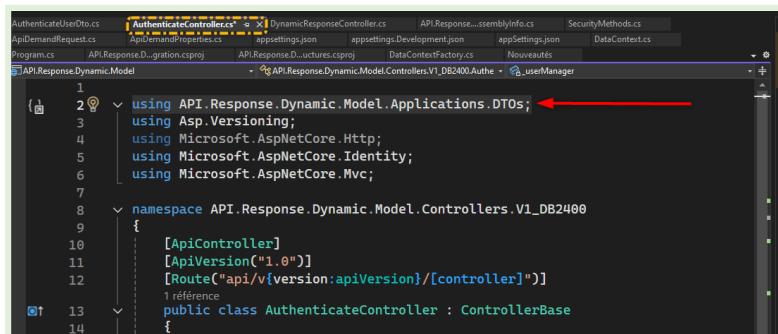
## IX Added the “Login” method in the “AuthenticateController” controller

- 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.

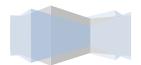
### IX-A The code for the public method “Login”

**Note:** Remember to add the using “`using API.Response.Dynamic.Model.Applications.DTOs;`” in the “AuthenticateController” controller.

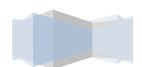


```
1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 [Route("Login")]
2 [HttpPost]
3 public async Task<IActionResult> Login([FromBody] AuthenticateUserDto DtoUser)
4 {
5     //> By default, pessimistic <
6     IActionResult result = this.BadRequest();
7
8     //> We'll verify the profile using the email address and password <
9     //> The idea is to be able to authenticate with the login that...
10    // ...IS the email address. <
11    //> Since we're searching using the email address (contained in...
12    // ...the login, we use the "FindByEmailAsync" method <
13
14 }
```

```
1 //> Reminder: when our "Login" method is "async",...
2 // ...we must do an "await" <
3 var user = await this._userManager.FindByEmailAsync(DtoUser.Login);
4
5 if (user != null)
6 {
7     //> We will check the password <
8     var verif = await this._userManager.CheckPasswordAsync(user, DtoUser.Password);
9     if (verif)
10     {
11         //> We load the DTO with the calculation of the token returned by the method...
12         // ..."GenerateJwtToken" <
13         result = this.Ok(new AuthenticateUserDto()
14         {
15             Login = user.Email,
16             Name = user.UserName,
17             Token = this.GenerateJwtToken(user),
18         });
19     }
20 }
```



```
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

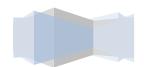
### X-A Added the “[Authorize]” attribute

```

1  using Microsoft.AspNetCore.Mvc;
2  using System.Dynamic;
3  using System.Reflection.Emit;
4  using System.Reflection;
5  using API.Response.Dynamic.Model.Infrastructure.Services;
6  using API.Responses.Dynamic.Model.Infrastructure.Services;
7  using API.Response.Dynamic.Model.Infrastructure.Connectors;
8  using API.Response.Dynamic.Model.Domain.Models;
9  using System.Text;
10 using Asp.Versioning;
11 using Microsoft.AspNetCore.Identity;
12 using Microsoft.AspNetCore.Authorization;
13 using Microsoft.AspNetCore.Authentication.JwtBearer;
14 using Microsoft.AspNetCore.Cors;
15
16 // -----
17 // Versionning API
18 // Voir :
19 // https://mohsen.es/api-versioning-and-swagger-in-asp-net-core-7-0-fe45f67d8419
20 // -----
21
22
23
24 namespace API.Response.Dynamic.Model.Controllers.V1_DB2400
25 {
26     [ApiController]
27     [ApiVersion("1.0")]
28     [Route("api/v{version:apiVersion}/{controller}")]
29     [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)] ←
30     1 référence
31     public class DynamicResponseController : ControllerBase
32     {
33
34         #region fields
35         // > Pour récupérer l'injection de dépendance <
36         // ( Voir la classe "Program.cs" )
37         private readonly IParamRepository _repository = null;
38
39         // On déclare un membre pour récupérer l'environnement d'exécution <
40         // ( ici on en aura pas besoin, mais c'est pour l'exemple )
41         private readonly IWebHostEnvironment _webHostEnvironment = null;
42         #endregion
43
44         #region Methods
45         // > Constructeur <
46         0 références
47         public DynamicResponseController(IParamRepository repository, IWebHostEnvironment webHost)
48         {
49             // > Récupération du repository <
50             _repository = repository;
51

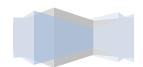
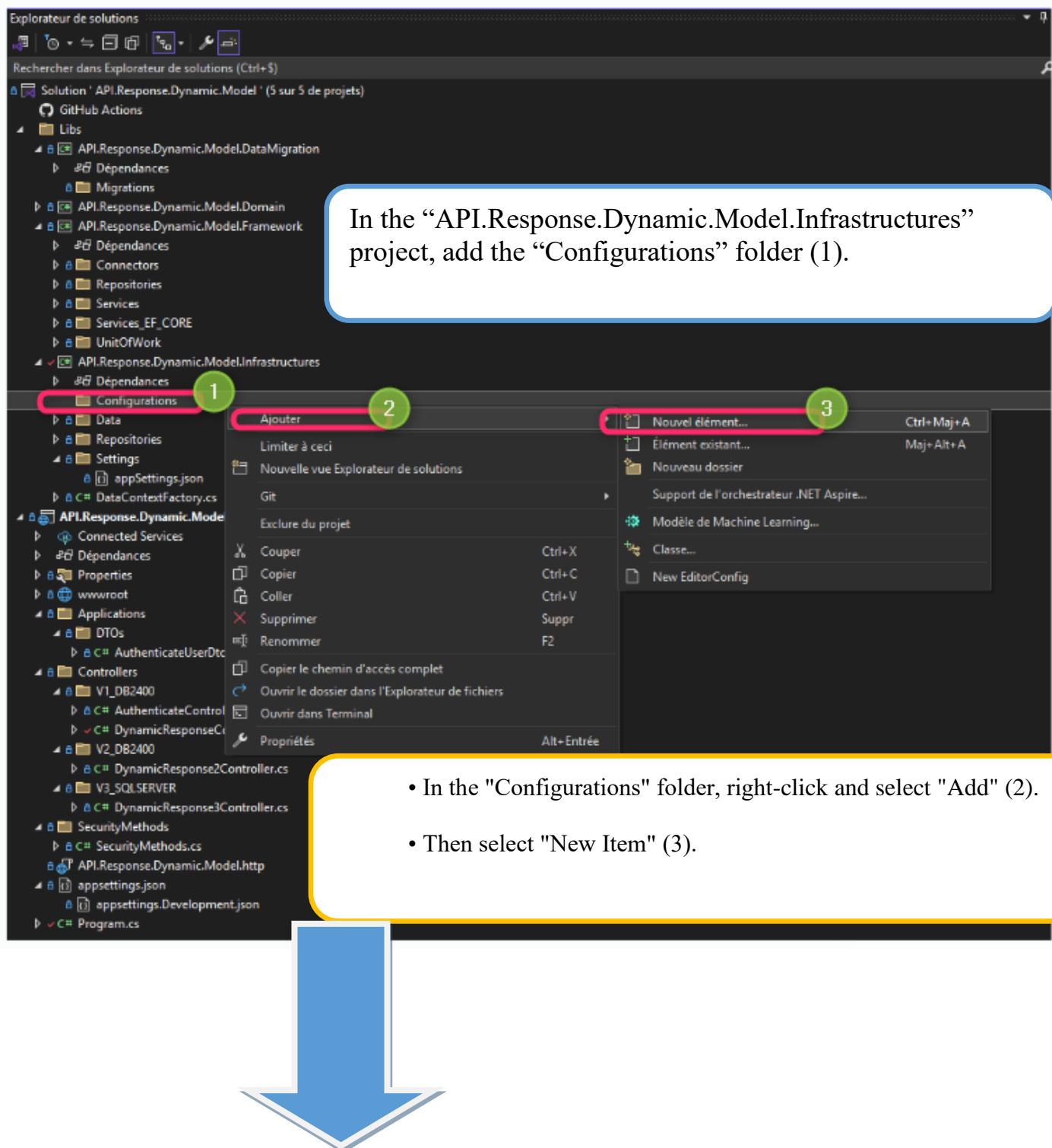
```

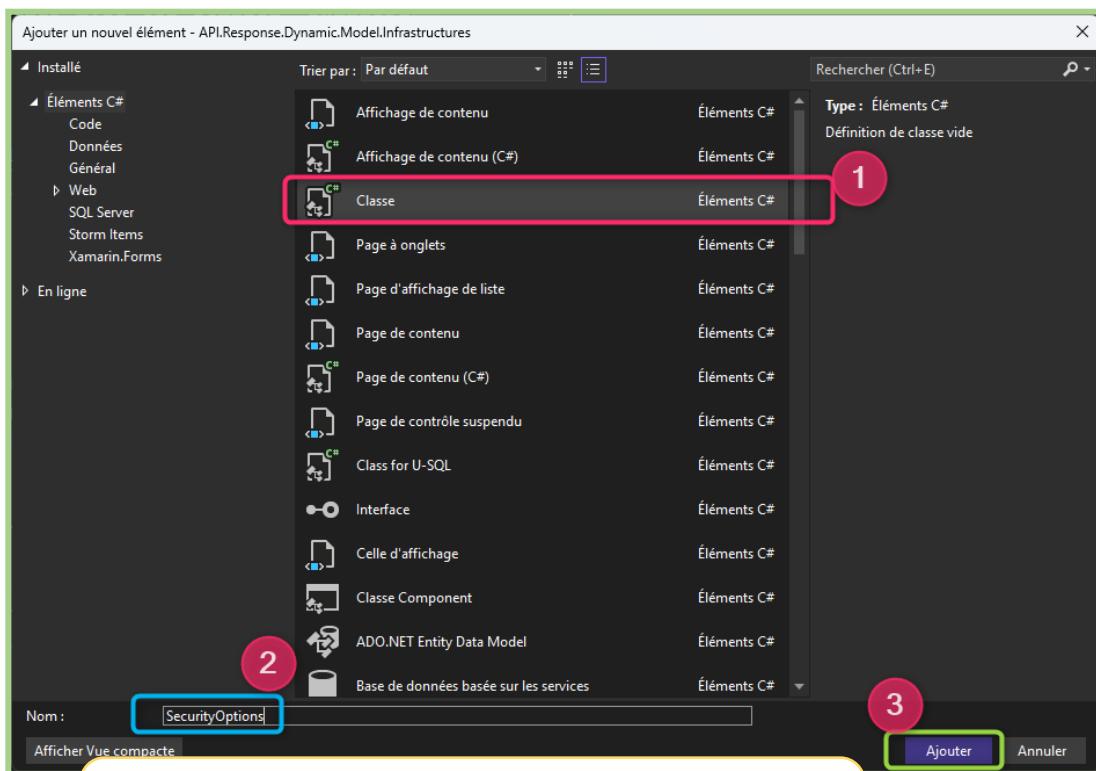
The attribute `[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]` will allow the validity of the transmitted Token to be checked.



## XI Improve the configuration....

### XI-A Create a “SecurityOptions” class to store the Token calculation key





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



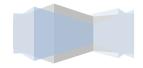
```
SecurityOptions.cs
1 1 .ng System;
2 2 .ng System.Collections.Generic;
3 3 .ng System.Linq;
4 4 .ng System.Text;
5 5 .ng System.Threading.Tasks;
6
7 7 iespace API.Response.Dynamic.Model.Infrastructures.Configurations
8
9 9 0 références
10 10 public class SecurityOptions
11 11 {
12 12     #region properties
13 13         // > Pour récupérer la clé de calcul du JWT Token <
14 14     0 références
15 15     public string key { get; set; }
16 }
```

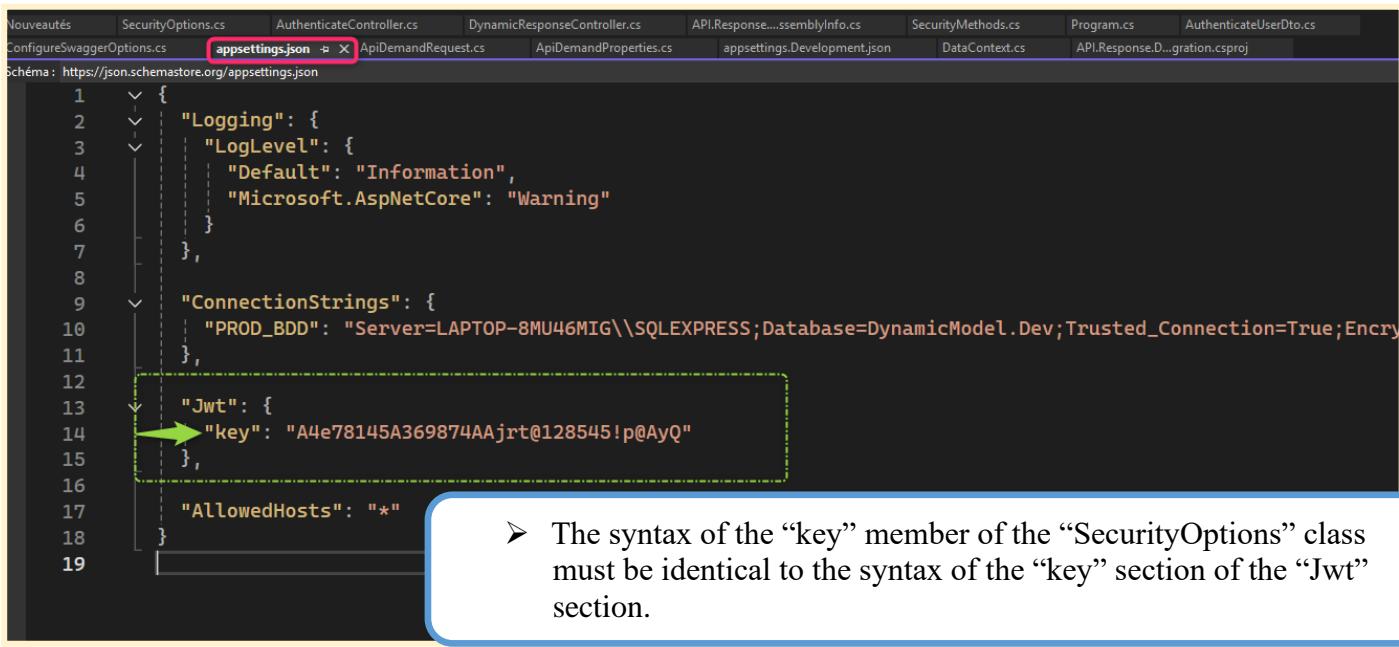
Explorateur de solutions

Solution : API.Response.Dynamic.Model (5 sur 5 de projets)

- Lbs
  - Dépendances
    - Migrations
  - API.Response.Dynamic.Model.Domain
    - Dépendances
    - Connectors
    - Repositories
    - Services
    - UnitOfWork
  - API.Response.Dynamic.Model.Infrastructures
    - Dépendances
      - Configurations
        - SecurityOptions.cs
    - Data
    - Repositories
    - Settings
    - appSettings.json
    - DataContextFactory.cs
  - API.Response.Dynamic.Model
    - Connected Services
    - Dépendances
    - Properties
    - wwwroot
    - Applications
      - DTOs
        - AuthenticateUserDto.cs
      - Controllers
        - V1\_DB2400
          - AuthenticateController.cs
          - DynamicResponseController.cs
        - V2\_DB2400
          - DynamicResponse2Controller.cs
        - V3\_SQLSERVER
          - DynamicResponse3Controller.cs
        - SecurityMethods
          - SecurityMethods.cs
  - appsettings.json
  - Program.cs

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

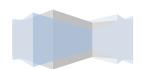




```
Nouveautés SecurityOptions.cs AuthenticateController.cs DynamicResponseController.cs API.Response...semblyInfo.cs SecurityMethods.cs Program.cs AuthenticateUserDto.cs
ConfigureSwaggerOptions.cs appsettings.json > X ApiDemandRequest.cs ApiDemandProperties.cs appsettings.Development.json DataContext.cs API.Response.D...ration.csproj
Schéma : https://json.schemastore.org/appsettings.json

1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8
9    "ConnectionStrings": {
10      "PROD_BDD": "Server=LAPTOP-8MU46MIG\\SQLEXPRESS;Database=DynamicModel.Dev;Trusted_Connection=True;Encrypt=true"
11    },
12
13    "Jwt": {
14      "key": "A4e78145A369874AAjrt@128545!p@AyQ"
15    },
16
17    "AllowedHosts": "*"
18  }
```

- 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

```

1 public static class SecurityMethods
2 {
3     #region properties
4     //> Déclare clé pour la construction du Token <
5     // private static string MyKeyJwt { get; set; }
6     private static string MyKeyJwt;
7
8     //> Crée une variable "args" de type tableau de string nullable (?) <
9     // ( pour " var builder = WebApplication.CreateBuilder(args) )
10    // ( Remarque : on peut l'appeler autrement que "args" si on veut )
11    private static string[]? args { get; set; }
12
13    #endregion
14    /// <summary>
15    /// Construction du processus d'authentification JWT
16    /// </summary>
17    /// <param name="builder"></param>
18    /// <returns></returns>
19    public static object AddCustomAuthentication( object _builder )
20    {
21        //> 1/ Création d'un "configurationBuilder" <
22        ConfigurationBuilder configurationBuilder = new ConfigurationBuilder();
23
24        //> 2/ Création d'un objet "Configuration" à partir du...
25        // ... "configurationBuilder" <
26        IConfiguration configuration = configurationBuilder.Build();
27
28        //> 3/ On déclare une variable "builder" de type...
29        // ... "WebApplicationBuilder" <
30        var builder = WebApplication.CreateBuilder(args);
31
32        //> 4 Si le paramètre transmis est non nul, alors on récupère son contenu <
33        if ( _builder is not null )
34        {
35            //> On Parse "_builder" avec la classe "WebApplicationBuilder" <
36            builder = (WebApplicationBuilder) _builder;
37
38            //> 5/ Récupération de la clé d'authentification <
39            // ( On récupère la clé depuis "appsettings.json" )
40            MyKeyJwt = string.Empty;
41            // [ Ancienne méthode pour récupérer la clé de calcul du Token ]
42            // MyKeyJwt = builder.Configuration.GetSection("Jwt")["key"];
43
44            // [ Nouvelle méthode pour récupérer la clé de calcul du Token ]
45            SecurityOptions ElemeSecurity = new SecurityOptions();
46            builder.Configuration.GetSection("Jwt").Bind(ElemeSecurity);
47
48            MyKeyJwt= ElemeSecurity.key;
49
50
51
52
53
54
55
56
57
58
59
60

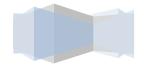
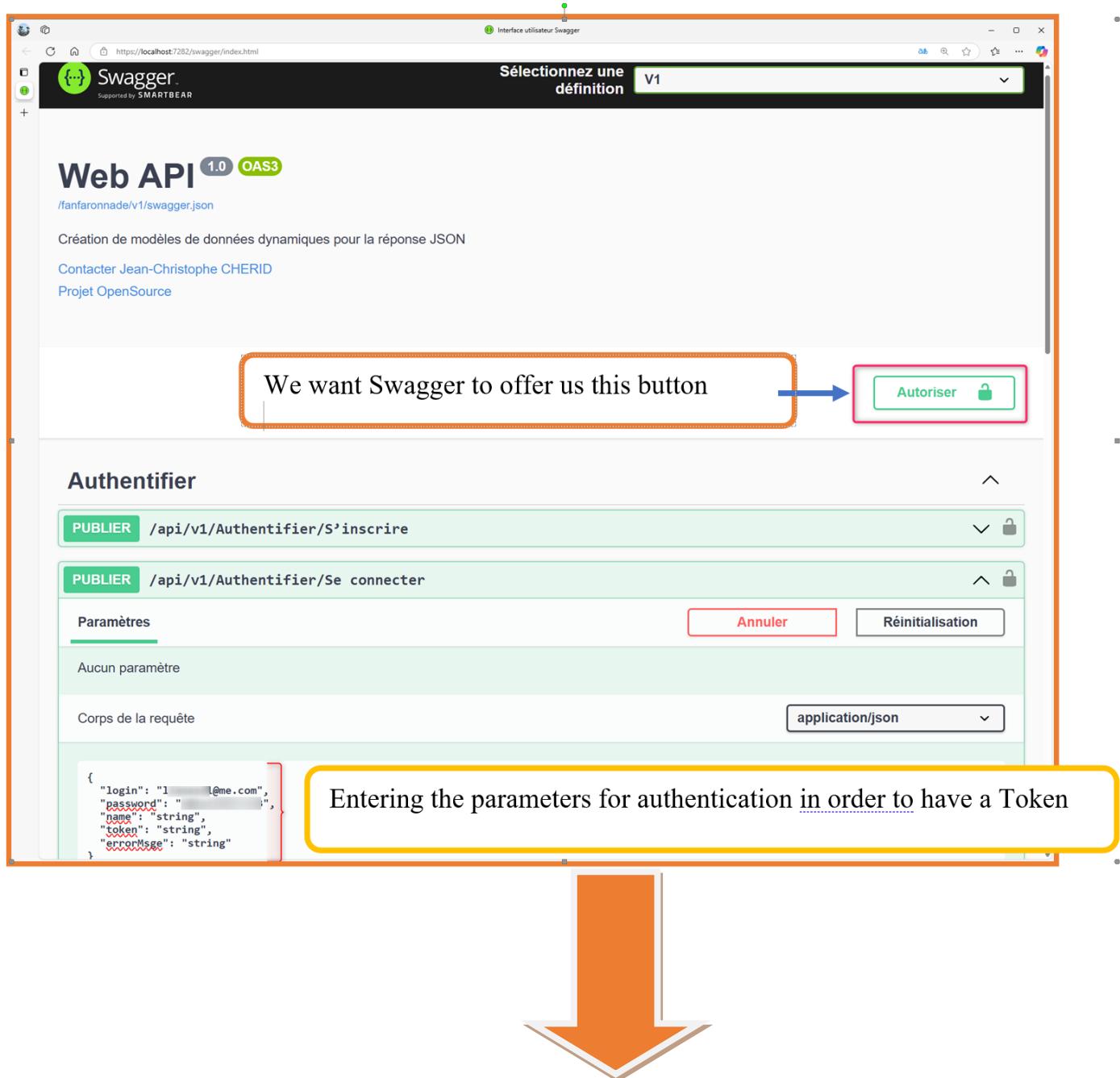
```

Retrieving the Token calculation key by binding.

## 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



**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",
    "errorMse": "string"
}'
```

**URL de la demande**

<https://localhost:7282/api/v1/Authenticate/Login>

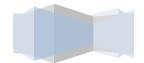
**Réponse du serveur**

Code	Détails
200	<b>Corps de la réponse</b> <pre>{   "login": "████████@me.com",   "password": null,   "name": "string",   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJZC16ImQ5MG1WNDM0L1czZGUTNDg0My1hNDgxLTa5NWQ5MTEXZWQ2NiisInN1Yi16ImxpyW5hemVsQG11LmNbS1s1mVtYlsljoi6gb1hbmF62WxAbwUuY29tiwianRpIjoiOTU2MmVzT0t0Dg3Ny00TiwlLWE3YjotZGU5YThhYzQ1Njc2IiwlbmJmIjoxNzMzk3MzMyLc1leHai0JE3MzMzOTc1MTisImhlhdCI6MTczMzM5NzMzMn0.8Inpw9d_npxMggK0-C-sqX3UB3Xq8949pYErkVK0M_A",   "errorMse": null }</pre> <div style="display: flex; justify-content: space-between;"> <span><a href="#">Copier</a></span> <span><a href="#">Télécharger</a></span> </div> <b>En-têtes de réponse</b> <pre>api-supported-versions: 1.0 content-type: application/json; charset=utf-8 date: Thu, 05 Dec 2024 11:15:32 GMT server: Kestrel</pre>

**Réponses**

Code	Description	Liens
200		Aucun

You must copy the Token between the " " (quotation marks)



Note: you must click on the “Authorize” button (1)

**Authentifier**

PUBLIER /api/v1/Au  
PUBLIER /api/v1/Au

**Autorisations disponibles**

**Bearer (apiKey)**

En-tête d'autorisation JWT utilisant le schéma Bearer.  
Nom: Authorization  
Dans: header  
Valeur:

2 Bearer eyJhbGciOiJIUzI1Nils

3 Autoriser Fermer

Annuler Réinitialisation

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

**Autorisations disponibles**

**Bearer (apiKey)**

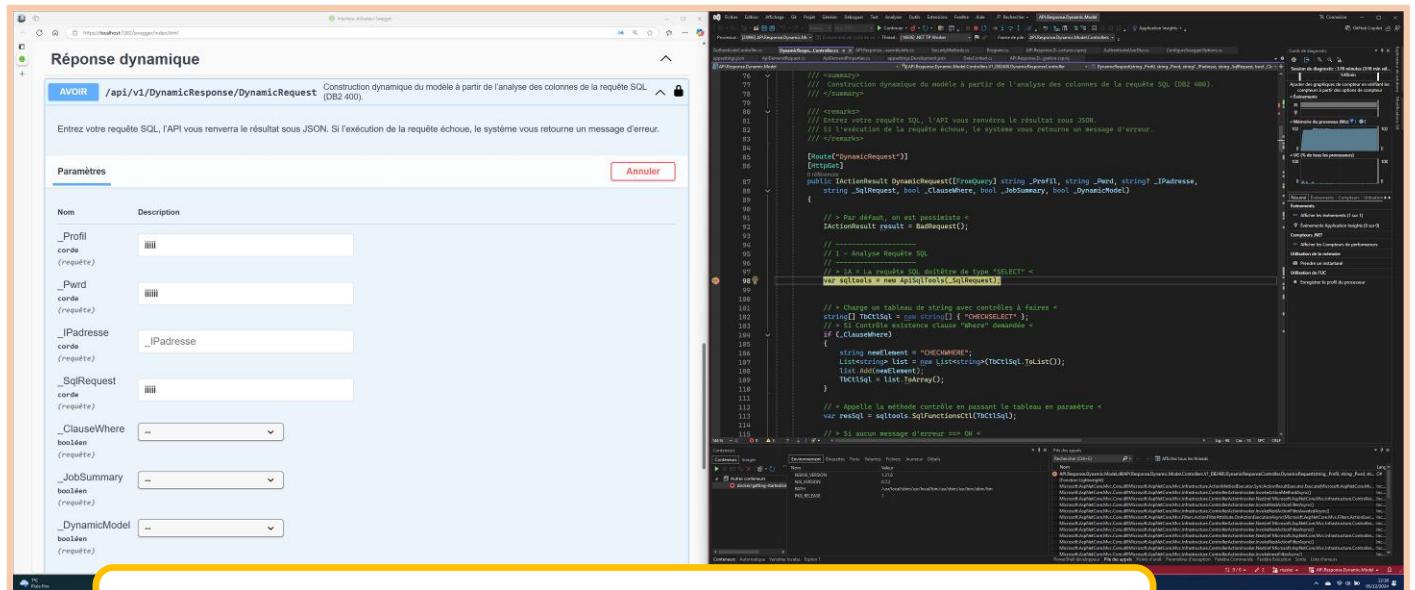
Autorisé

En-tête d'autorisation JWT utilisant le schéma Bearer.  
Nom: Authorization  
Dans: header  
Valeur: \*\*\*\*\*

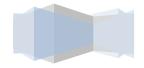
Déconnexion Fermer

Click on “Close” to finish.





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<SwaggerDefaultValueFilter>();

    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>()
        }
    });
});
```

