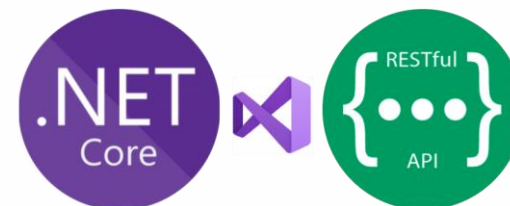


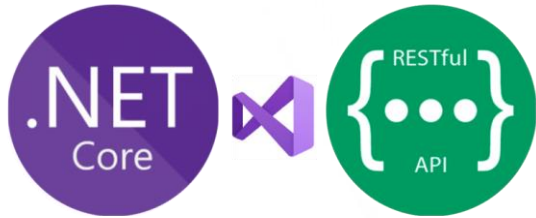


Aplicar JWT (JSON Web Tokens) en API REST



www.sena.edu.co

Adalberto Cárcamo Alvarado



Aplicar JWT (JSON Web Tokens) en API REST

Temas a trabajar en esta guía:

1. Creación del proyecto (usando plantilla ASP.NET Core Web API).
2. **Aplicar JWT (JSON Web Tokens) en API REST.**
3. Consumir WEB API desde ASP.NET CORE.

API ASP.NET CORE

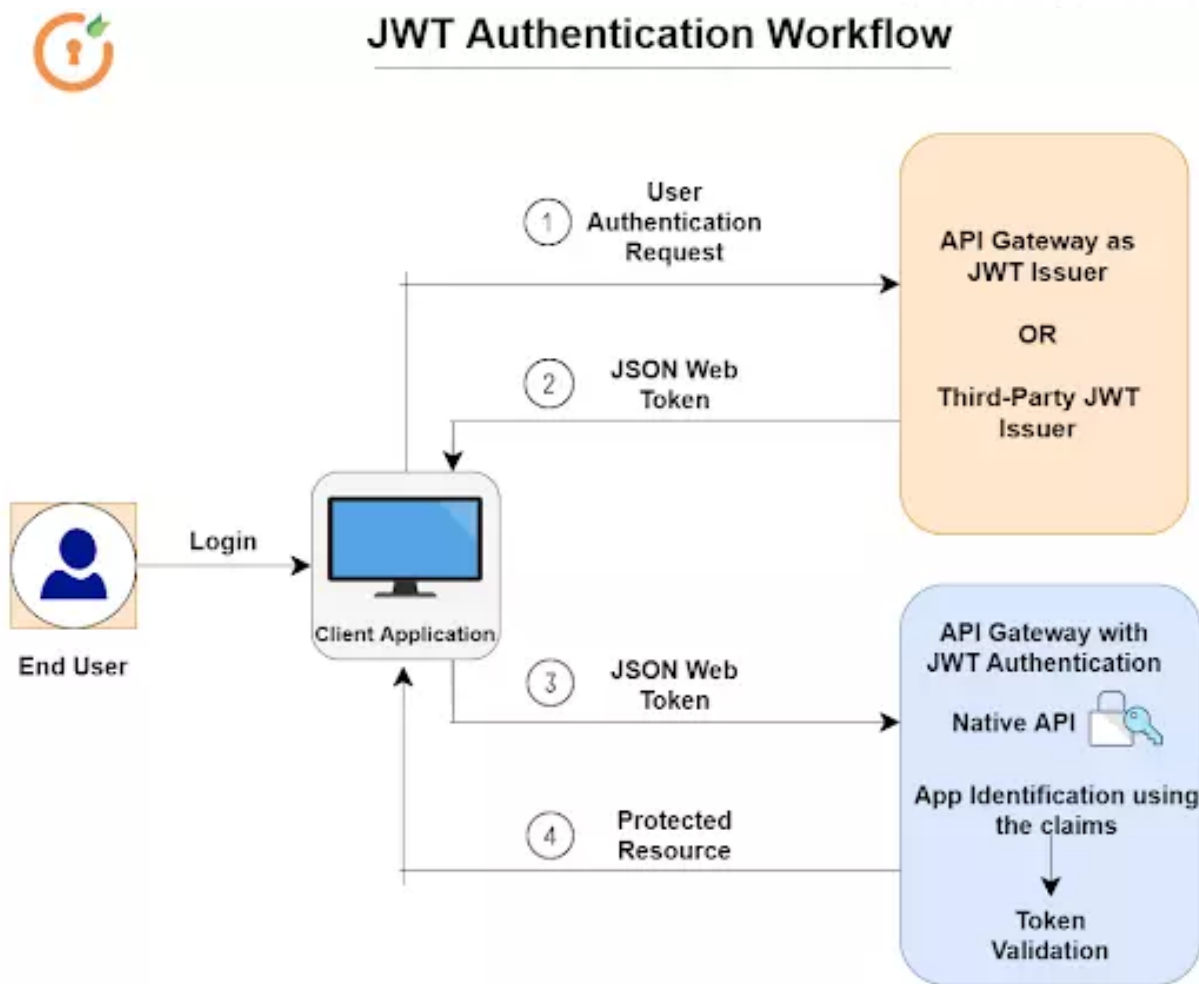


1. Aplicar JWT (JSON Web Tokens) en API REST.

- a. Introducción a JWT.
- b. Instalación de paquetes para JWT.
- c. Configuración de JWT en proyecto.
- d. Crear modelo "USUARIO".
- e. Implementar lógica de autenticación de usuario.
- f. Consumir API JWT desde C#.



a. Introducción a JWT.



WT es un objeto **de** JSON (notación **de** objeto **de** JavaScript), una herramienta **de** estándar abierto cuyo objetivo es establecer una transmisión **de** información entre dos o más campos. A partir **de** estos, se puede propagar información **de** forma segura y efectiva, que, además, es verificada, pues se firma **de** forma virtual.

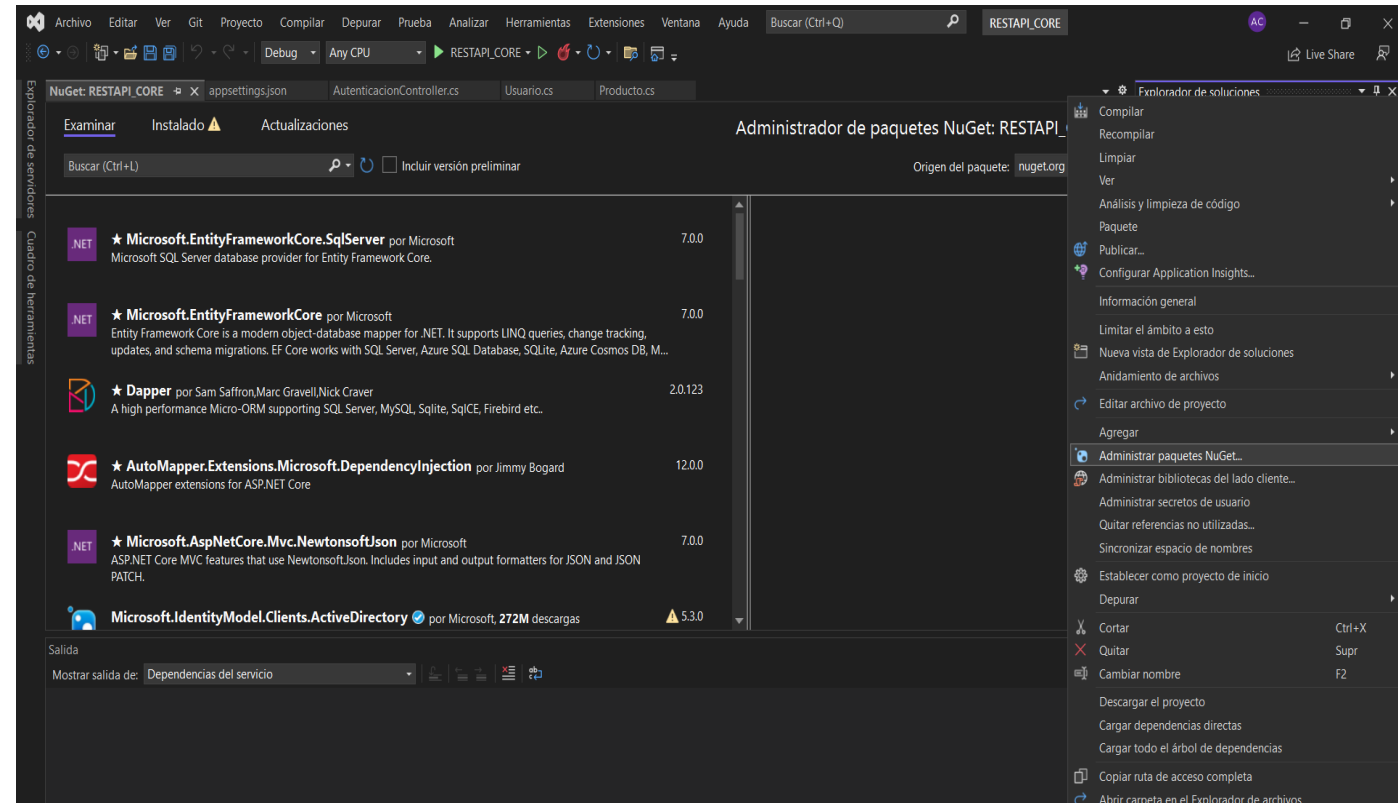
b. Instalación de paquetes para JWT.



Instalamos los siguiente paquetes requeridos para JWT

Presionamos click derecho en el proyecto y escogemos Administrador Paquetes NuGet e instalamos los siguientes paquetes:

- Microsoft.AspNetCore.Authentication.JwtBearer
- System.IdentityModel.Tokens.Jwt



c. Configuración de JWT en proyecto.



Configuración de la llave secreta:
Ingresamos al archivo appsettings.json

Digitamos el siguiente código:

```
"settings": {  
  "secretKey": "=Adsi2022="  
}
```



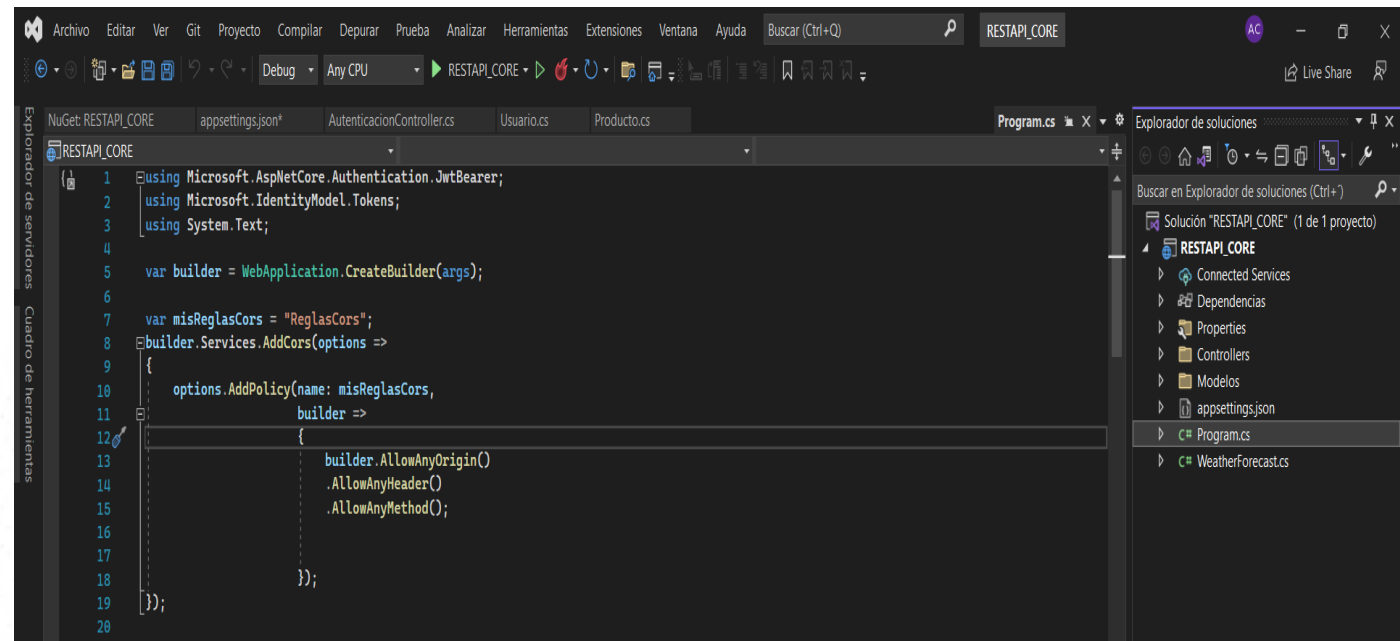
```
1  {  
2  "ConnectionStrings": {  
3    "CadenaSQL": "Data Source=DESKTOP-UGSFCI7\\SQLEXPRES  
4  },  
5  "Logging": {  
6    "LogLevel": {  
7      "Default": "Information",  
8      "Microsoft.AspNetCore": "Warning"  
9    }  
10 },  
11 "AllowedHosts": "*",  
12 "settings": {  
13   "secretKey": "*Adsi2022*"  
14 }  
15 }  
16
```

c. Configuración de JWT en proyecto.



Vamos al archivo Program.cs y usamos los paquetes que instalamos

```
using Microsoft.AspNetCore.Authentication.JwtBearer;  
using Microsoft.IdentityModel.Tokens;  
using System.Text;
```



c. Configuración de JWT en proyecto.



Implementamos ahora el JWT:

En el mismo archivo Program.cs

Usando el siguiente código que nos permite crear el JWT de acuerdo a la clave secreta que configuramos en appsettings.cs

```
//===== PRIMERO =====  
builder.Configuration.AddJsonFile("appsettings.json");  
var secretKey = builder.Configuration.GetSection("settings").GetSection("secretKey").ToString();// "=Adsi2022=";  
var keyBytes = Encoding.UTF8.GetBytes(secretKey);  
  
builder.Services.AddAuthentication(config => {  
  
    config.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;  
    config.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;  
}).AddJwtBearer(config => {  
    config.RequireHttpsMetadata = false;  
    config.SaveToken = false;  
    config.TokenValidationParameters = new TokenValidationParameters  
    {  
        ValidateIssuerSigningKey = true,  
        IssuerSigningKey = new SymmetricSecurityKey(keyBytes),  
        ValidateIssuer = false,  
        ValidateAudience = false  
    }  
});
```

A screenshot of the Visual Studio IDE showing the implementation of JWT configuration in the Program.cs file. The code is written in C# and follows the same structure as the code block on the left. The file explorer on the left shows the project structure with folders for NuGet, RESTAPI_CORE, appsettings.json, AutenticacionController.cs, Usuario.cs, and Producto.cs. The code editor shows the Program.cs file with the JWT configuration code highlighted in blue. The status bar at the bottom indicates "No se encontraron problemas." (No problems found).

c. Configuración de JWT en proyecto.



Luego en el mismo archivo `appsettings.cs`
Digitamos aplicamos la autorización correspondiente

```
app.UseAuthentication();
```

A screenshot of the Visual Studio IDE showing the `Program.cs` file in a project named `RESTAPI_CORE`. The code is in C# and shows the configuration of the application's request pipeline. The `app.UseAuthentication();` line is highlighted, indicating the step being discussed. The code includes comments in Spanish, such as "Configurar la pipeline de solicitudes HTTP" and "SEGUNDO". The right sidebar shows the Solution Explorer with the project structure, including folders for `Controllers`, `Modelos`, and `appsettings.json`. The bottom status bar shows the current line and character positions.

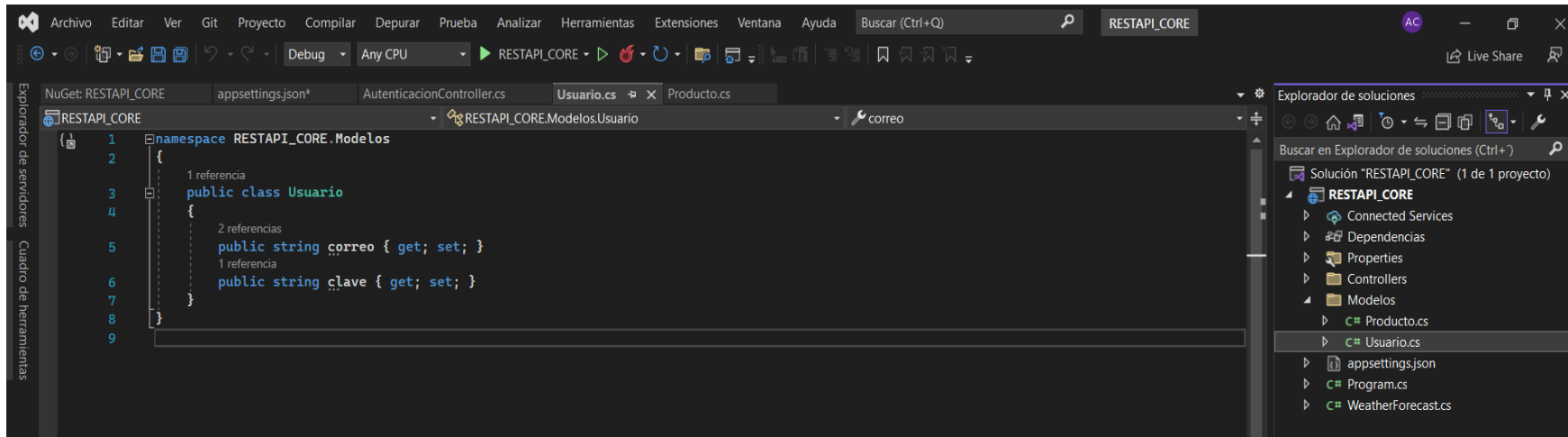
```
46 ValidateAudience = false;
47 };
48 };
49
50
51 var app = builder.Build();
52
53 // Configurar la pipeline de solicitudes HTTP.
54 if (app.Environment.IsDevelopment())
55 {
56     app.UseSwagger();
57     app.UseSwaggerUI();
58 }
59
60 app.UseCors(misReglasCors);
61
62 //===== SEGUNDO =====
63 app.UseAuthentication();
64
65 app.UseAuthorization();
66
67 app.MapControllers();
68
69 app.Run();
70
```

d. Crear modelo "USUARIO".



Creamos el siguiente Modelo:
Nos permitirá autenticarnos para recibir el JWT

```
public class Usuario
{
    public string correo { get; set; }
    public string clave { get; set; }
}
```



e. Implementar lógica de autenticación de usuario.



Creamos un controlador con el siguiente nombre:
AutenticacionController

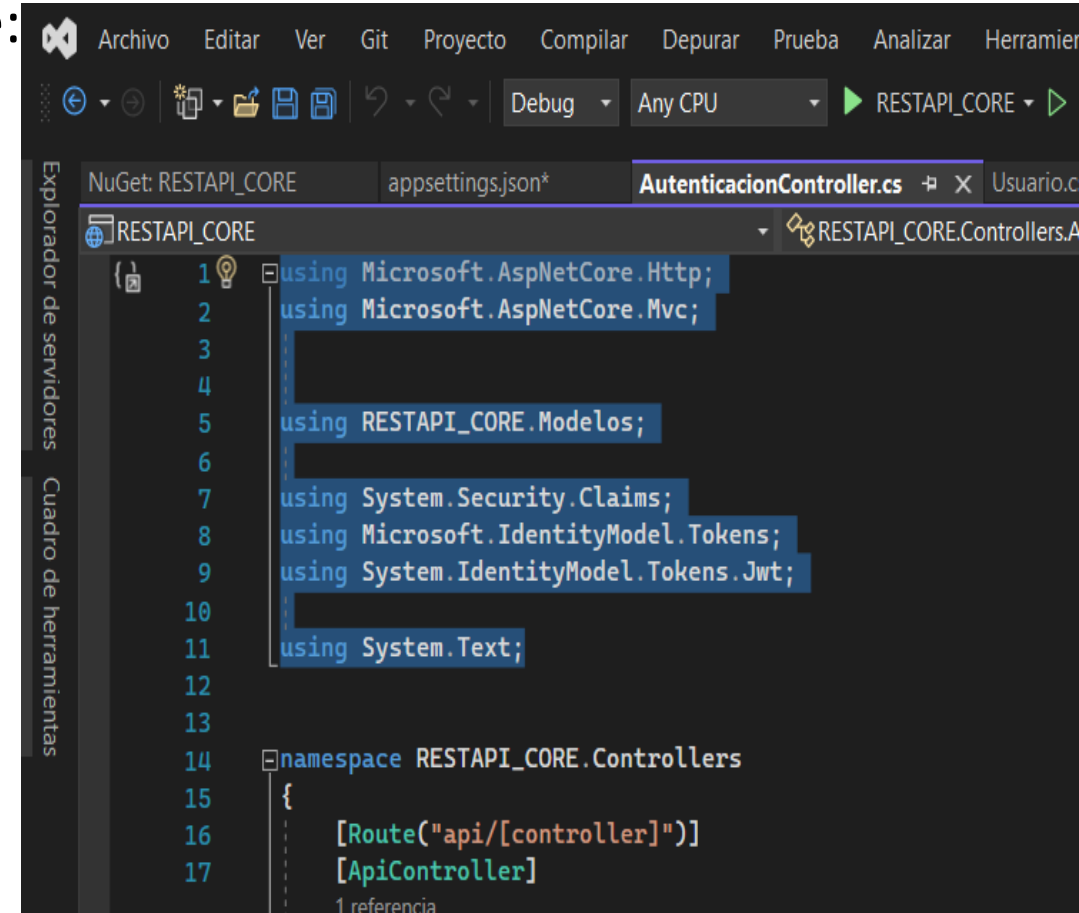
Usamos las siguientes referencias:

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

```
using RESTAPI_CORE.Modelos;
```

```
using System.Security.Claims;  
using Microsoft.IdentityModel.Tokens;  
using System.IdentityModel.Tokens.Jwt;
```

```
using System.Text;
```



e. Implementar lógica de autenticación de usuario.



Código del controlador AutenticacionController

```
namespace RESTAPI_CORE.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AutenticacionController : ControllerBase
    {
        private readonly string secretKey;

        public AutenticacionController(IConfiguration config) {
            secretKey = config.GetSection("settings").GetSection("secretKey").ToString();
        }

        [HttpPost]
        [Route("Validar")]
        public IActionResult Validar([FromBody] Usuario request) {

            if (request.correo == "adsi2022@sena.com" && request.clave == "1234")
            {

                var keyBytes = Encoding.ASCII.GetBytes(secretKey);
                var claims = new ClaimsIdentity();
                claims.AddClaim(new Claim(ClaimTypes.NameIdentifier, request.correo));

                var tokenDescriptor = new SecurityTokenDescriptor
                {
                    Subject = claims,
                    Expires = DateTime.UtcNow.AddMinutes(5),
                    SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(keyBytes), SecurityAlgorithms.HmacSha256Signature)
                };

                var tokenHandler = new JwtSecurityTokenHandler();
                var tokenConfig = tokenHandler.CreateToken(tokenDescriptor);

                string tokencreado = tokenHandler.WriteToken(tokenConfig);

                return StatusCode(StatusCodes.Status200OK, new { token = tokencreado });
            }
            else {
                return StatusCode(StatusCodes.Status401Unauthorized, new { token = "" });
            }
        }
    }
}
```

e. Implementar lógica de autenticación de usuario.



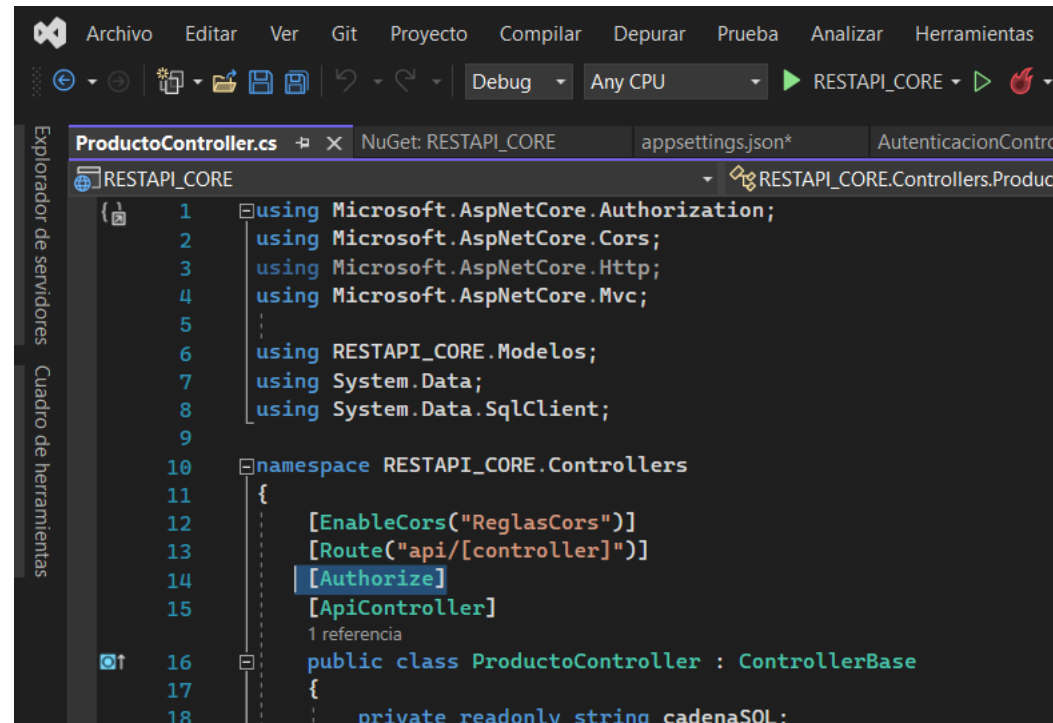
Por ultimo autorizamos en el controlador Producto a todas sus APIS que sean autenticadas antes de ser ejecutados.

Usamos en la parte superior la referencia:

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

Luego autorizamos la APIS en la parte superior de la clase `public class ProductoController`

Con la siguiente instrucción
`[Authorize]`

A screenshot of the Visual Studio IDE showing the code for the `ProductoController.cs` file. The file is part of the `RESTAPI_CORE` project. The code includes several using statements at the top, followed by a namespace declaration for `RESTAPI_CORE.Controllers`. Inside this namespace, there is a class `ProductoController` that inherits from `ControllerBase`. The `[Authorize]` attribute is applied to the class. The code is as follows:

```
1 using Microsoft.AspNetCore.Authorization;
2 using Microsoft.AspNetCore.Cors;
3 using Microsoft.AspNetCore.Http;
4 using Microsoft.AspNetCore.Mvc;
5
6 using RESTAPI_CORE.Modelos;
7 using System.Data;
8 using System.Data.SqlClient;
9
10 namespace RESTAPI_CORE.Controllers
11 {
12     [EnableCors("ReglasCors")]
13     [Route("api/[controller]")]
14     [Authorize]
15     [ApiController]
16     public class ProductoController : ControllerBase
17     {
18         private readonly string cadenaSQL;
```

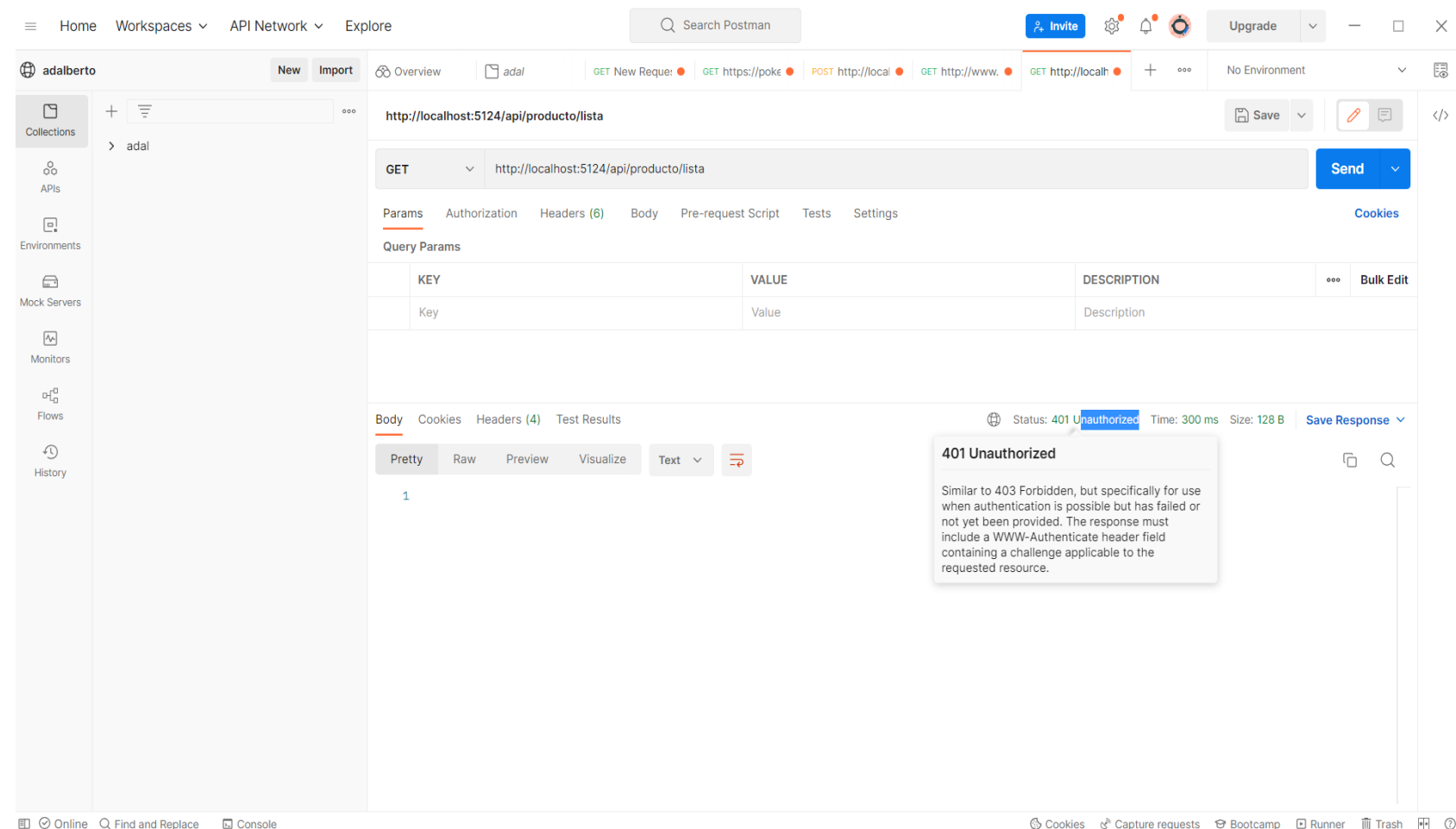
f. Realizamos las pruebas desde Postman



Ejecutamos nuestra API en Postman

<http://localhost:5124/api/producto/lista>

Notamos que nos aparece un error 401
Que no permite mostrar la información
Ya que no tenemos autorización.



f. Realizamos las pruebas desde Postman



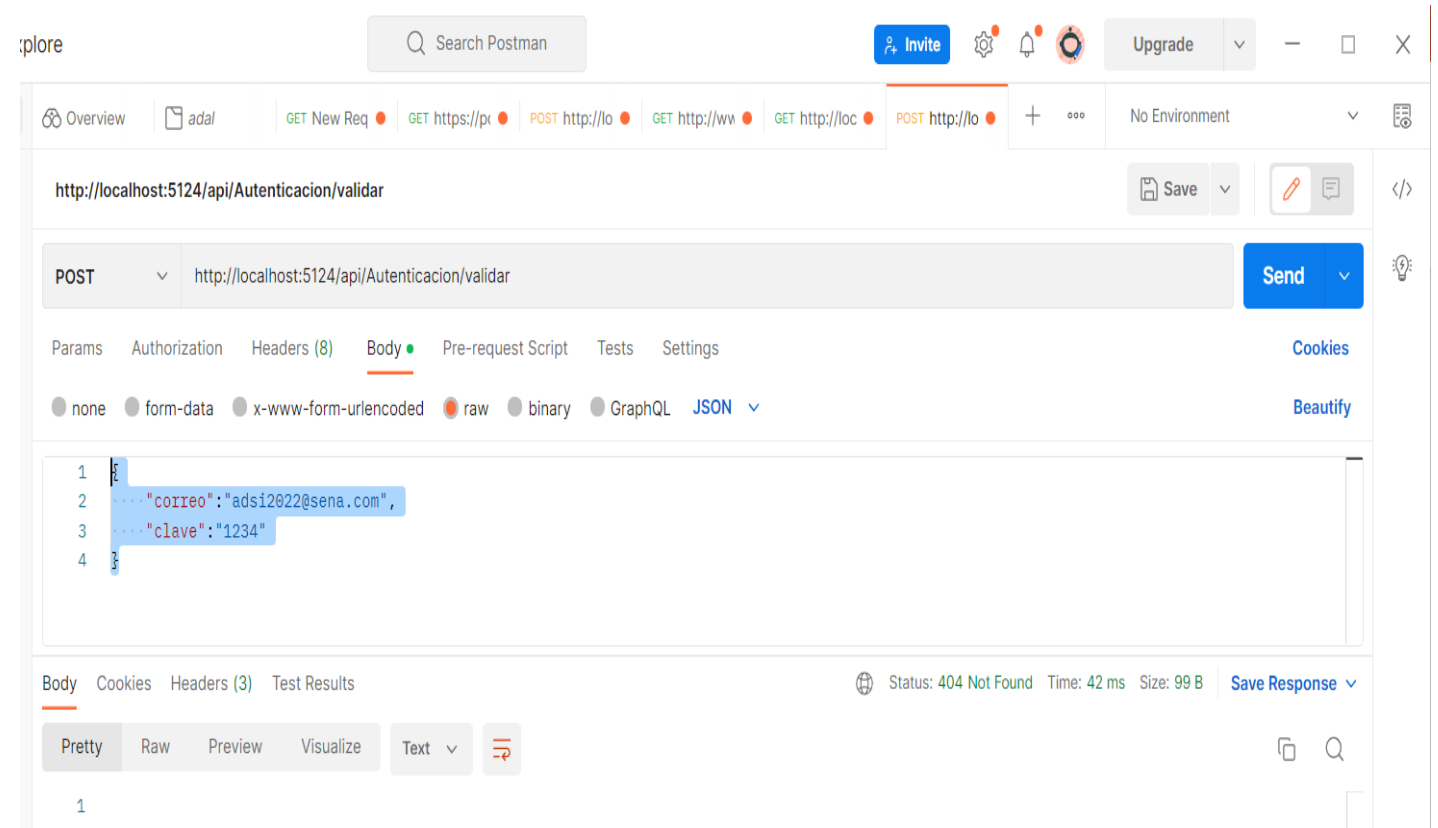
Ejecutamos ahora nuestra API de validación en la autenticación y probamos.

Usamos la API por medio de POST:

<http://localhost:5124/api/Autenticacion/validar>

E ingresamos las credenciales

```
{  
  "correo": "ads2022@sena.com",  
  "clave": "1234"  
}
```

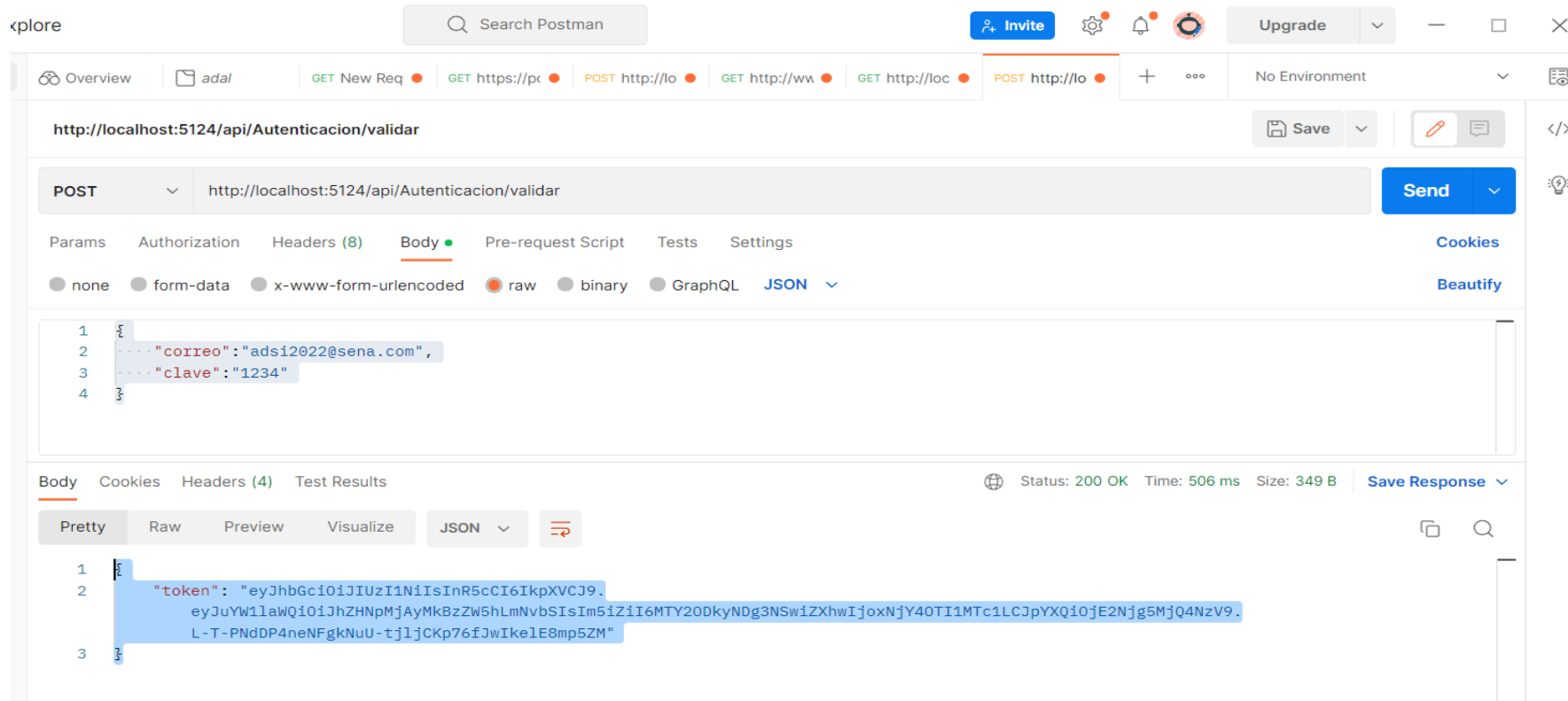


f. Realizamos las pruebas desde Postman



Nos devuelve el JWTToken como respuesta.

```
{  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiJhZHNpMjAyMkZlbnVbSIsIm5iZiI6MTY2ODkyNDg3NSwiZXhwIjoxNjY4OTI1MTc1LCJpYXQiOiJlY2Njg5MjQ4NzV9.L-T-PNdDP4neNFgkNuU-tjljCKp76fJwIkelE8mp5ZM"  
}
```



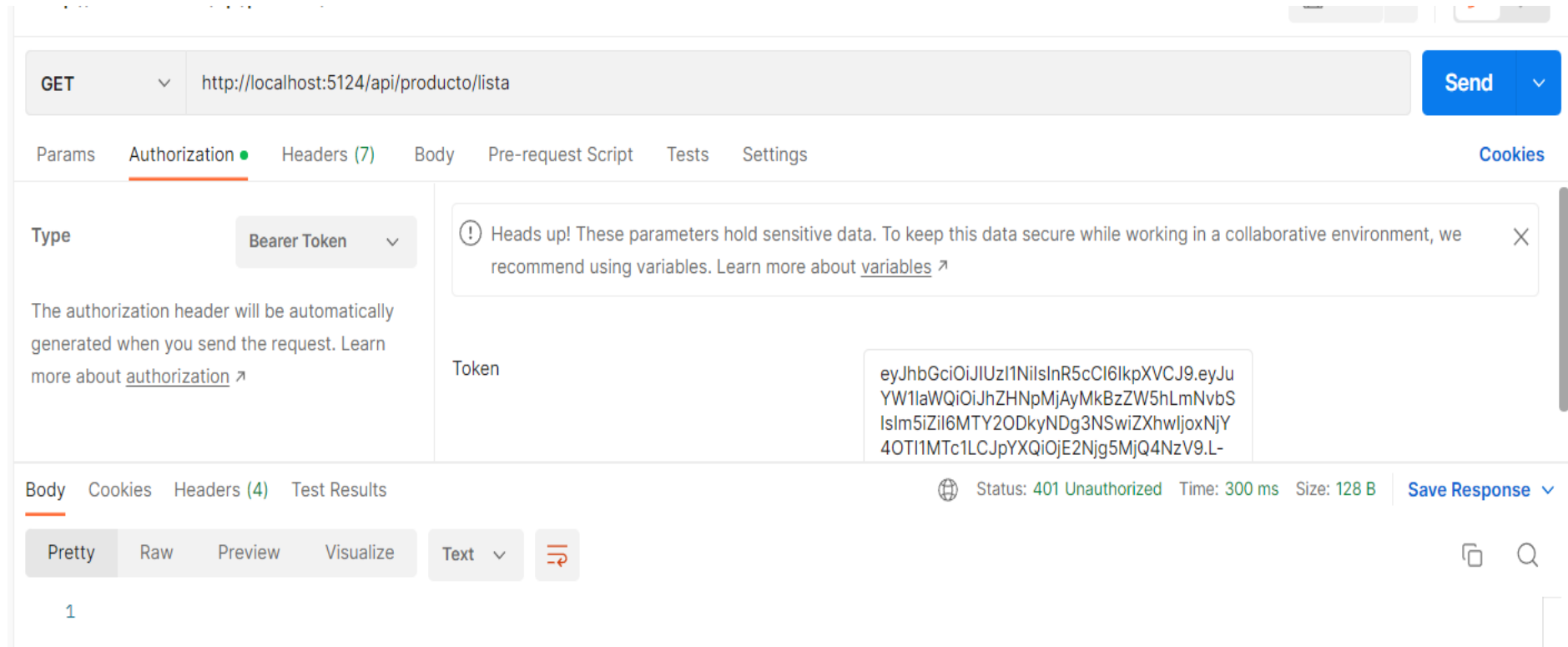
f. Realizamos las pruebas desde Postman



Ahora seleccionamos el token recibido y ejecutamos nuevamente la API de listar producto.

<http://localhost:5124/api/producto/lista>

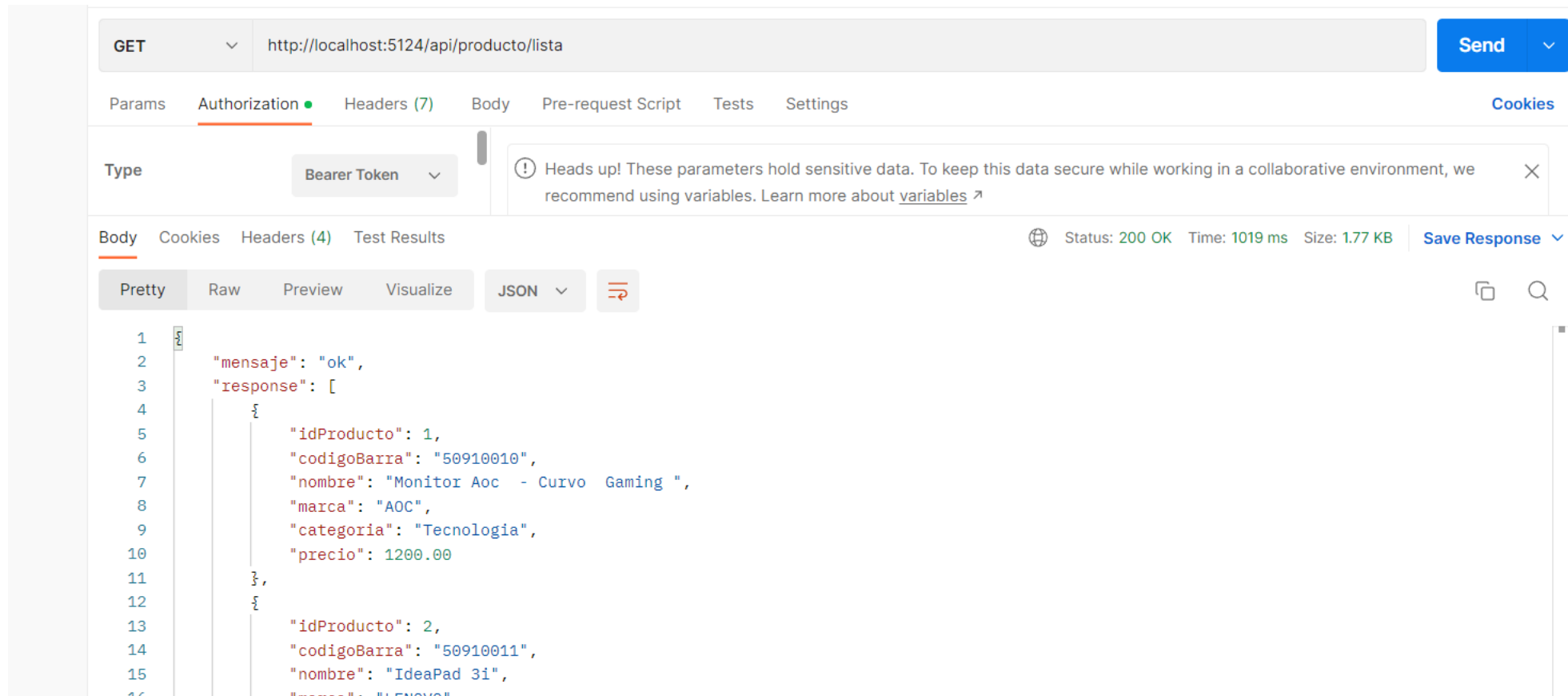
E ingresamos en Autorización y luego en Bearer Token el token recibido y ejecutamos la API



f. Realizamos las pruebas desde Postman



Nos lista correctamente la información ya que le asignamos la autorización recibida:





GRACIAS

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682



www.sena.edu.co