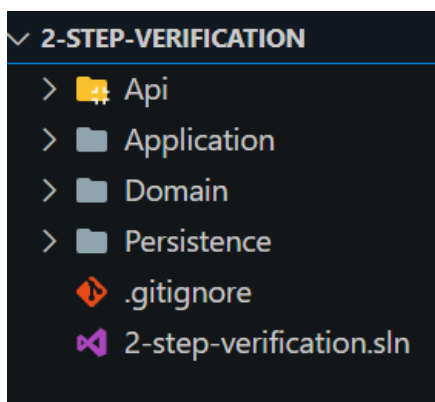
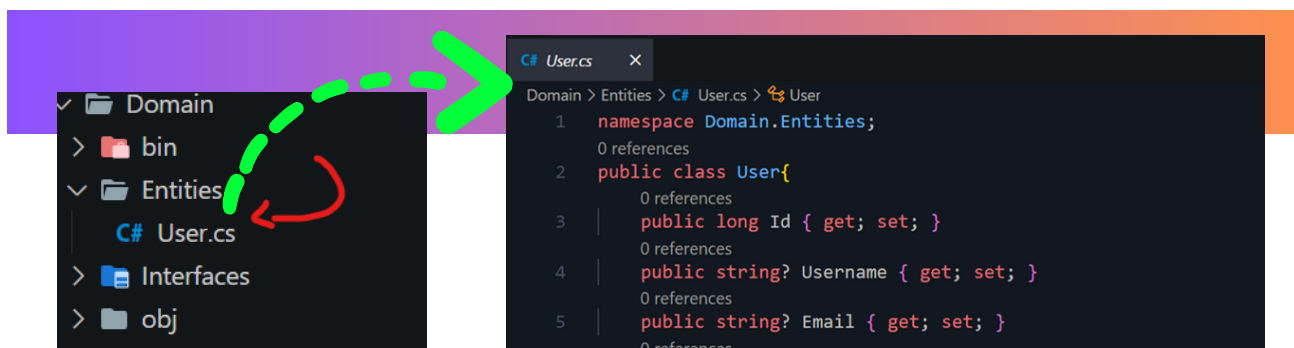




**IMPLEMENTAR UN PROYECTO .NET SIGUIENDO UNA ARQUITECTURA DE CUATRO CAPAS QUE PERMITA LA INCORPORACIÓN DE UN PROCESO DE VERIFICACIÓN DE DOS PASOS.**

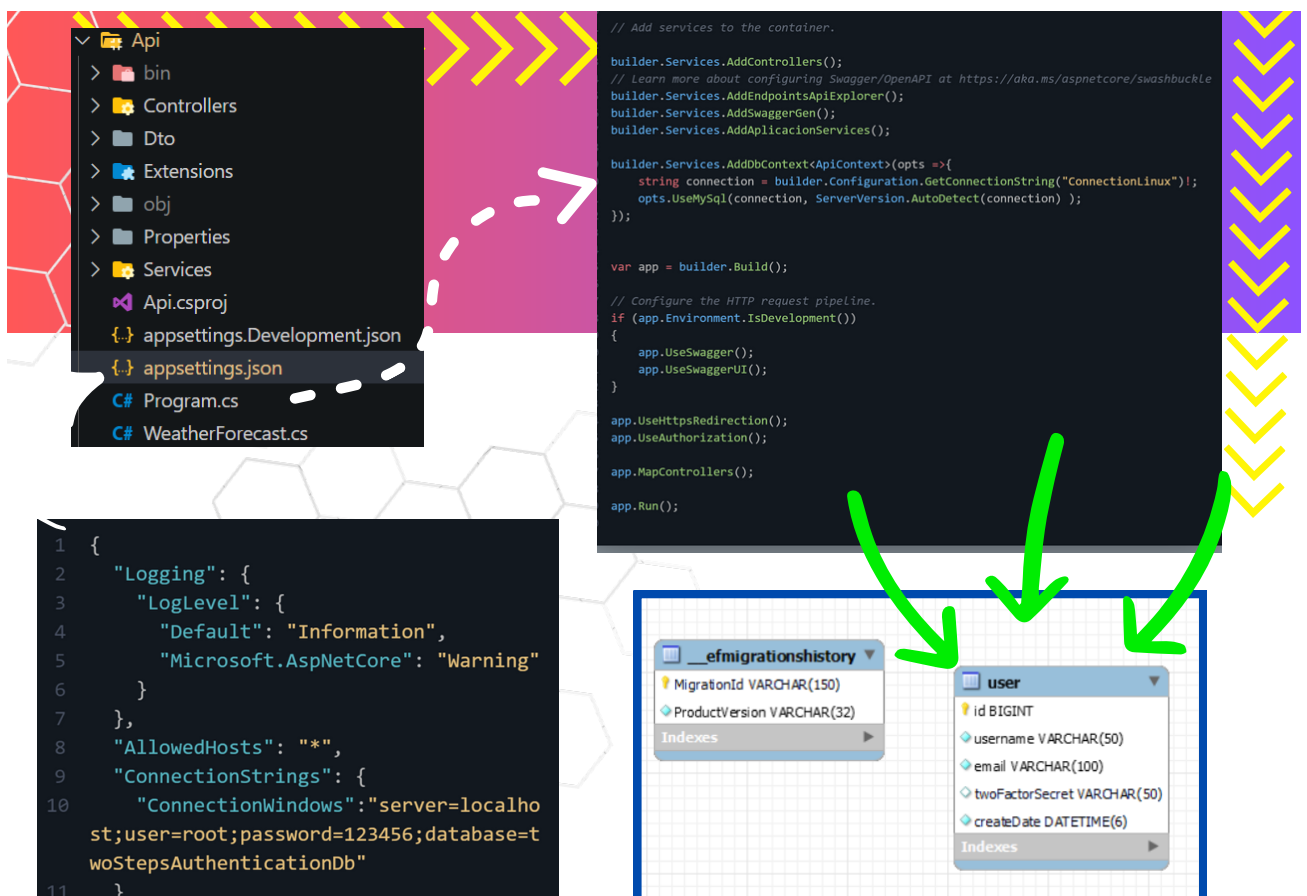


**1 SE PROCEDE A ESTABLECER LA ESTRUCTURA BASE DE UN PROYECTO, IMPLEMENTANDO LA ARQUITECTURA DE DISEÑO EN CUATRO CAPAS.**





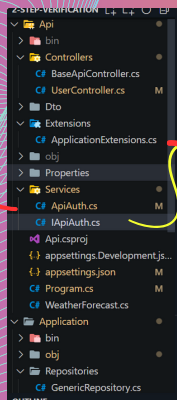
**2 SE DEFINE LA ENTIDAD BASE, SE CREAN LAS RESPECTIVAS CONFIGURACIONES Y AJUSTES ADICIONALES PARA CON ESTO LOGRAR DEFINIR LA CLASE DE CONTEXTO.**



```

1 using Domain.Entities;
2 using TwoFactorAuthNet;
3 using TwoFactorAuthNet.Providers.Qr;
4
5 namespace VerificationProject.Services;
6 public class AuthService : IAuthService{
7     private readonly IConfiguration _conf;
8     public AuthService(
9         IConfiguration conf,
10         ILogger<AuthService> logger
11     ){
12         _conf = conf;
13     }
14
15     public byte[] CreateQR(ref User u){
16         if(u.Email == null){
17             throw new ArgumentNullException(u.Email);
18         }
19
20         var tfa = new TwoFactorAuth(
21             _conf["JWTSettings:Issuer"], // Issuer
22             6, // Longitud del codigo
23             30, // Duracion de la generacion
24             Algorithm.SHA256, // Algoritmo de cifrado
25             new ImageChartsQrCodeProvider() // Creador del qr
26         );
27
28         string secret = tfa.CreateSecret(160); // Crea una patron secreto de 160 bites
29         u.TwoFactorSecret = secret;
30
31         var qr = tfa.GetQrCodeImageAsDataUri(
32             u.Email, // EL Label
33             u.TwoFactorSecret // Patron secreto
34         ); // Genera la uri del QR
35
36         string UriQR = qr.Replace("data:image/png;base64,", "");
37
38         return Convert.FromBase64String(UriQR); // Regresamos el qr en forma de bytes
39     }
40
41     public bool VerifyCode(string secret, string code){
42         //
43         var tfa = new TwoFactorAuth(_conf["JWTSettings:Issuer"], 6, 30, Algorithm.SHA256);
44         return tfa.VerifyCode( // valida que el codigo sea generado usando el patron
45             secret, // Patron del Usuario
46             code // Codigo generado por la aplicacion de autenticacion
47         );
48     }
49
50 }
51

```



```

1 using Domain.Entities;
2
3 namespace VerificationProject.Services;
4 references
5 public interface IAuthService{
6     2 references
7     byte[] CreateQR(ref User u);
8     2 references
9     bool VerifyCode(string secret, string code);
10 }

```

```

5 using Microsoft.AspNetCore.Mvc;
6 using VerificationProject.Services;
7
8 namespace VerificationProject.Extensions;
9 0 references
10 public static class ApplicationExtensions{
11     1 reference
12     public static void AddAplicacionServices(this IServiceCollection services){
13         services.AddScoped<IUnitOfWork, UnitOfWork>();
14         services.AddScoped<IPasswordHasher<User>, PasswordHasher<User>>();
15         services.AddScoped<IUnitOfWork, UnitOfWork>();
16         services.AddScoped<IAuthService, AuthService>();
17     }
18 }

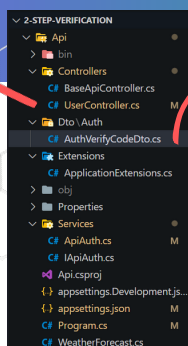
```

## 4 CREACION DEL SERVICIO DE AUTENTICACION EN DOS FACTORES, SE AÑADE AL SCOPE.

```

1 using Domain.Entities;
2 using Domain.Interfaces;
3 using Microsoft.AspNetCore.Mvc;
4 using VerificationProject.Dto.Auth;
5 using VerificationProject.Services;
6
7 namespace VerificationProject.Controllers;
8 public class UserController : BaseController{
9     private readonly ILogger<UserController> _logger;
10     private readonly IUnitOfWork _unitOfWork;
11     private readonly IAuthService _auth;
12     public UserController(
13         ILogger<UserController> logger,
14         IUnitOfWork unitOfWork,
15         IAuthService auth
16     ){
17         _logger = logger;
18         _unitOfWork = unitOfWork;
19         _auth = auth;
20     }
21
22     [HttpPost("qr/{id}")]
23     [ProducesResponseType(typeof(StatusCodes.Status200OK))]
24     [ProducesResponseType(typeof(StatusCodes.Status400BadRequest))]
25     public async Task<ActionResult> GetQr(long id){
26         try{
27             User u = await _unitOfWork.Users.FindAsync(x => x.Id == id);
28             byte[] qr = _auth.CreateQR(ref u);
29             _unitOfWork.Users.Update(u);
30             await _unitOfWork.SaveChangesAsync();
31             return File(qr, "image/png"); // se transforman los bytes en una imagen
32         }
33         catch (Exception ex){
34             _logger.LogError(ex, "Error al generar el QR");
35             return BadRequest("Error al generar el QR");
36         }
37     }
38
39     [HttpPost("verify")]
40     [ProducesResponseType(typeof(StatusCodes.Status200OK))]
41     [ProducesResponseType(typeof(StatusCodes.Status400BadRequest))]
42     [ProducesResponseType(typeof(StatusCodes.Status401Unauthorized))]
43     public async Task<ActionResult> VerifyCode(string secret, string code){
44         bool isValid = _auth.VerifyCode(secret, code);
45         if(!isValid){
46             return BadRequest("Codigo invalido");
47         }
48         return Ok("Codigo valido");
49     }
50 }

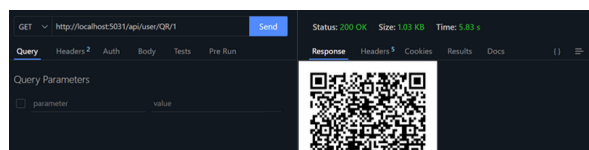
```

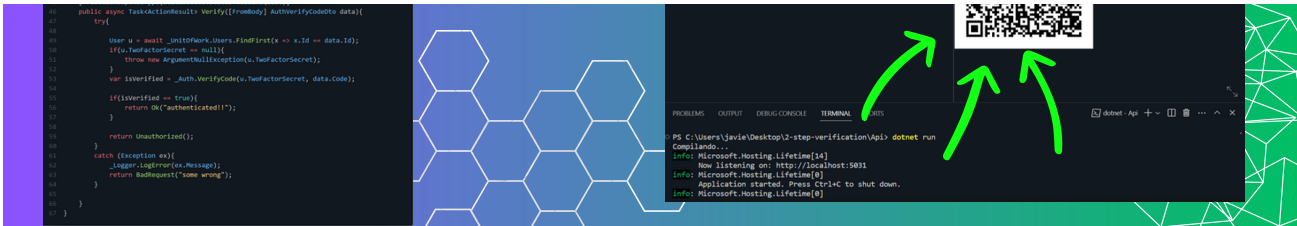


```

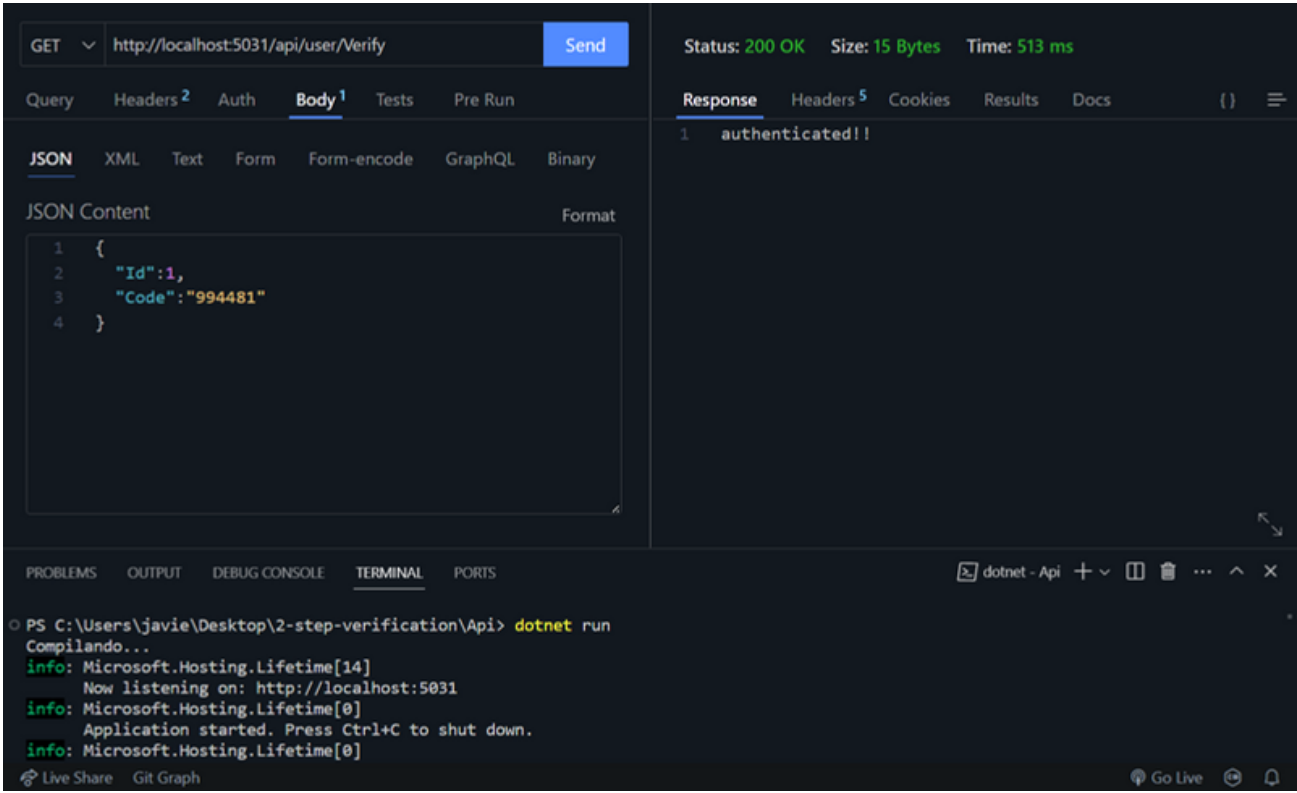
1 using System.ComponentModel.DataAnnotations;
2
3 namespace VerificationProject.Dto.Auth;
4 1 reference
5 public class AuthVerifyCodeDto{
6     [Required]
7     1 reference
8     public string Code { get; set; } = String.Empty;
9
10     [Required]
11     1 reference
12     public long Id { get; set; }
13 }

```





**5 SE IMPLEMENTA SU FUNCIONAMIENTO EN EL CONTROLADOR, CREANDO UN DTO PARA LA OBTENCIÓN DEL CODIGO.**



**5 AL UTILIZAR LA APLICACIÓN GOOGLE AUTHENTICATOR Y REGISTRAR EL CÓDIGO QR, SE GENERARÁN CÓDIGOS DE 6 DÍGITOS CADA 30 SEGUNDOS, SEGÚN LA CONFIGURACIÓN PREVIAMENTE ESPECIFICADA.**