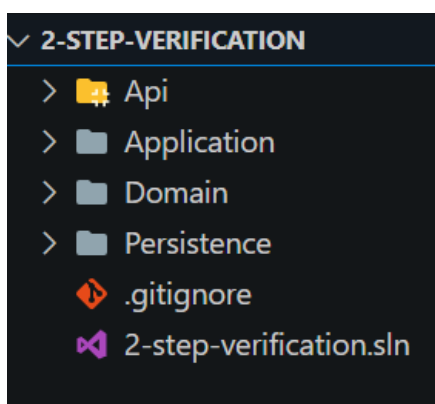
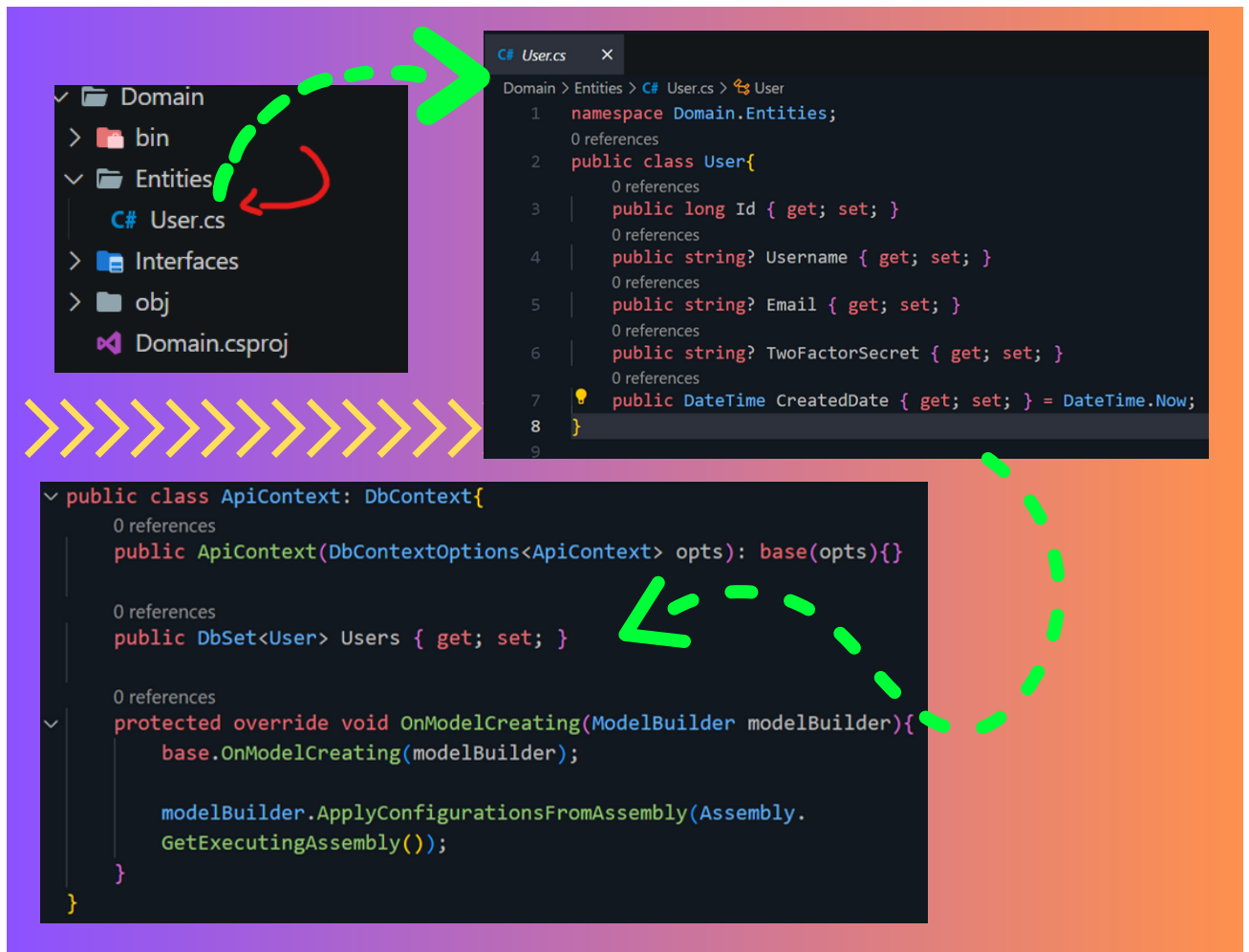




**IMPLEMENTAR UN PROYECTO .NET SIGUIENDO UNA ARQUITECTURA DE CUATRO CAPAS QUE PERMITA LA INCORPORACIÓN DE UN PROCESO DE VERIFICACIÓN DE DOS PASOS.**



**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.**

The image is a composite screenshot from Visual Studio illustrating the configuration of a database and migrations for an ASP.NET Core API. It features a colorful background with pink, purple, and orange geometric patterns.

- Top Left:** A file explorer view of the 'Api' project. A dashed white arrow points from the 'appsettings.json' file to the code editor.
- Top Right:** A code editor showing the 'Program.cs' file. It contains C# code for adding services to the container, including Swagger/OpenAPI and a database context. A green arrow points from this code to the database schema view.
- Bottom Left:** A code editor showing the 'appsettings.json' file. It contains JSON configuration for logging, allowed hosts, and connection strings. A green arrow points from this file to the database schema view.
- Bottom Right:** A database schema view showing two tables: '\_efmigrationshistory' and 'user'. The 'user' table has columns: 'id' (BIGINT), 'username' (VARCHAR(50)), 'email' (VARCHAR(100)), 'twoFactorSecret' (VARCHAR(50)), and 'createDate' (DATETIME(6)). A green arrow points from the 'user' table back to the 'Program.cs' code.

```
// Add services to the container.
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddApplicationServices();

builder.Services.AddDbContext<ApiContext>(opts =>{
    string connection = builder.Configuration.GetConnectionString("ConnectionLinux");
    opts.UseMySQL(connection, ServerVersion.AutoDetect(connection));
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "ConnectionWindows": "server=localhost;user=root;password=123456;database=testStepsAuthenticationDb"
  }
}
```

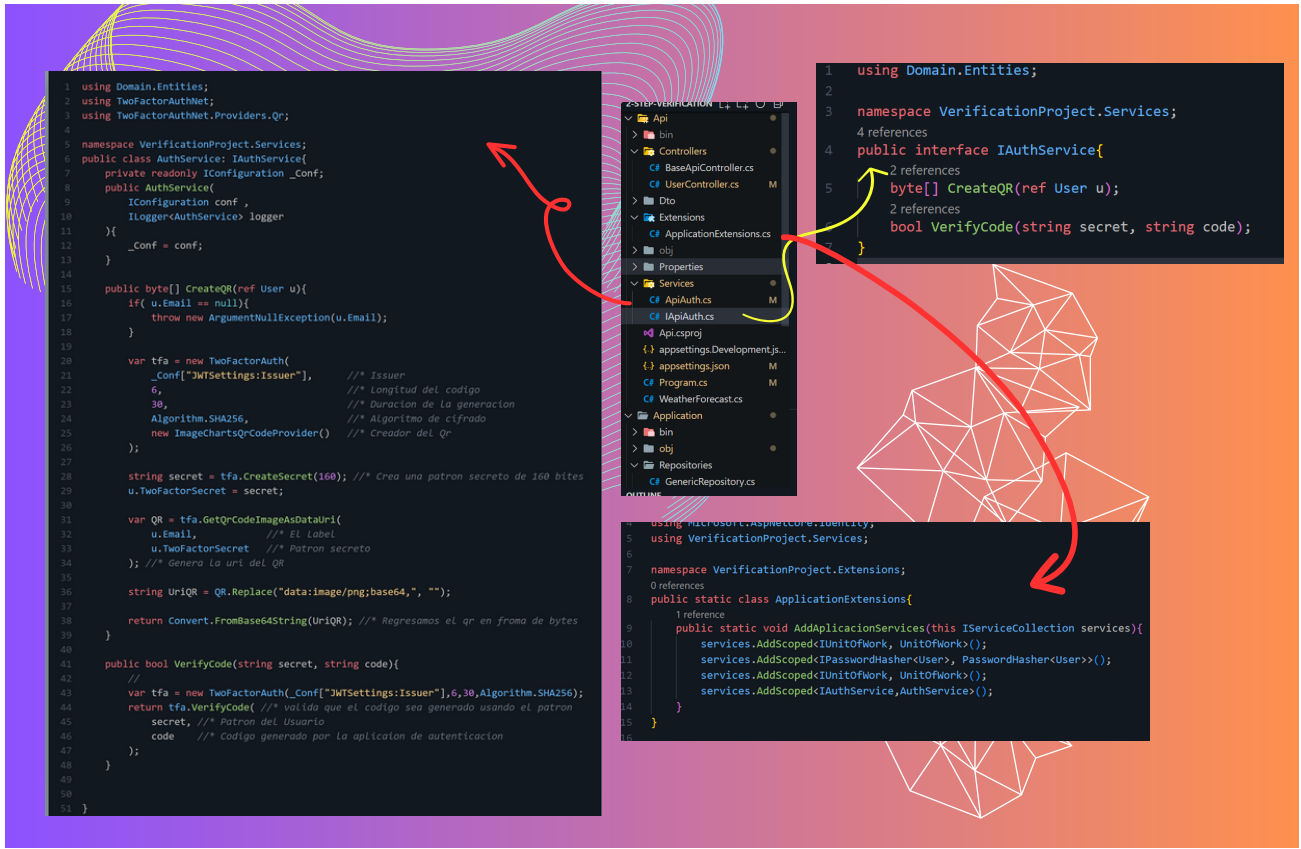
**\_efmigrationshistory**

MigrationId	ProductVersion
-------------	----------------

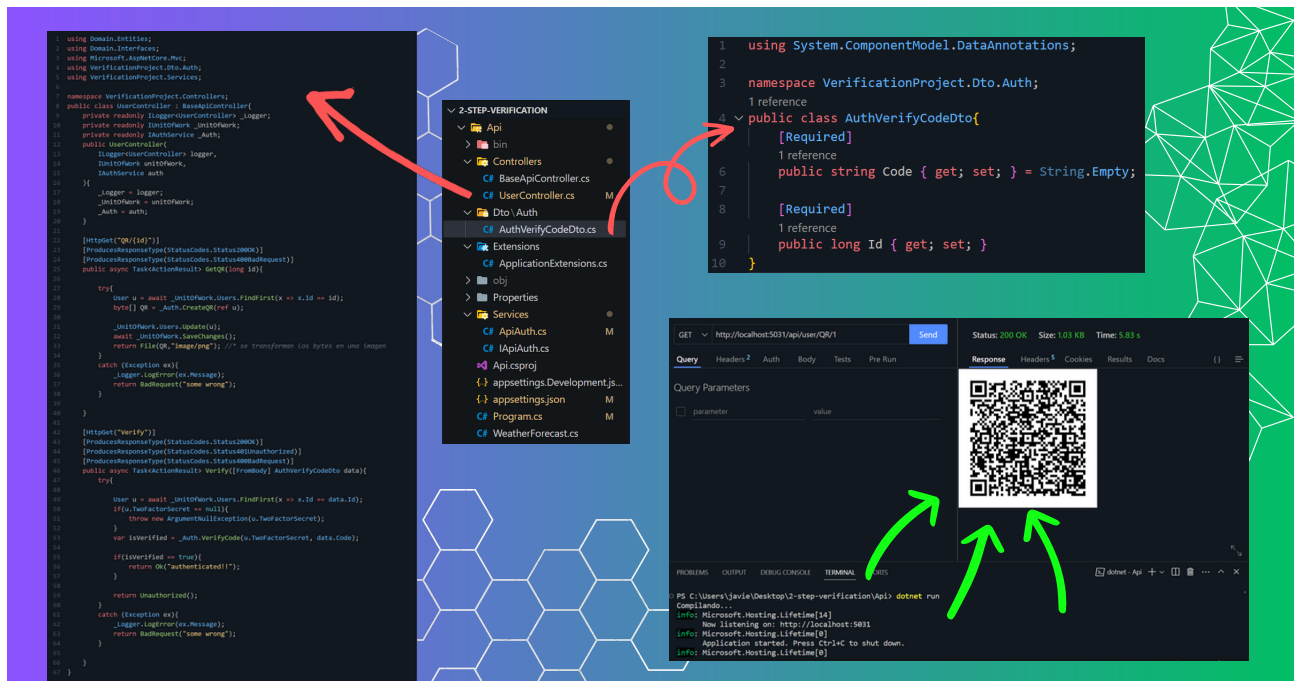
**user**

id	username	email	twoFactorSecret	createDate
----	----------	-------	-----------------	------------

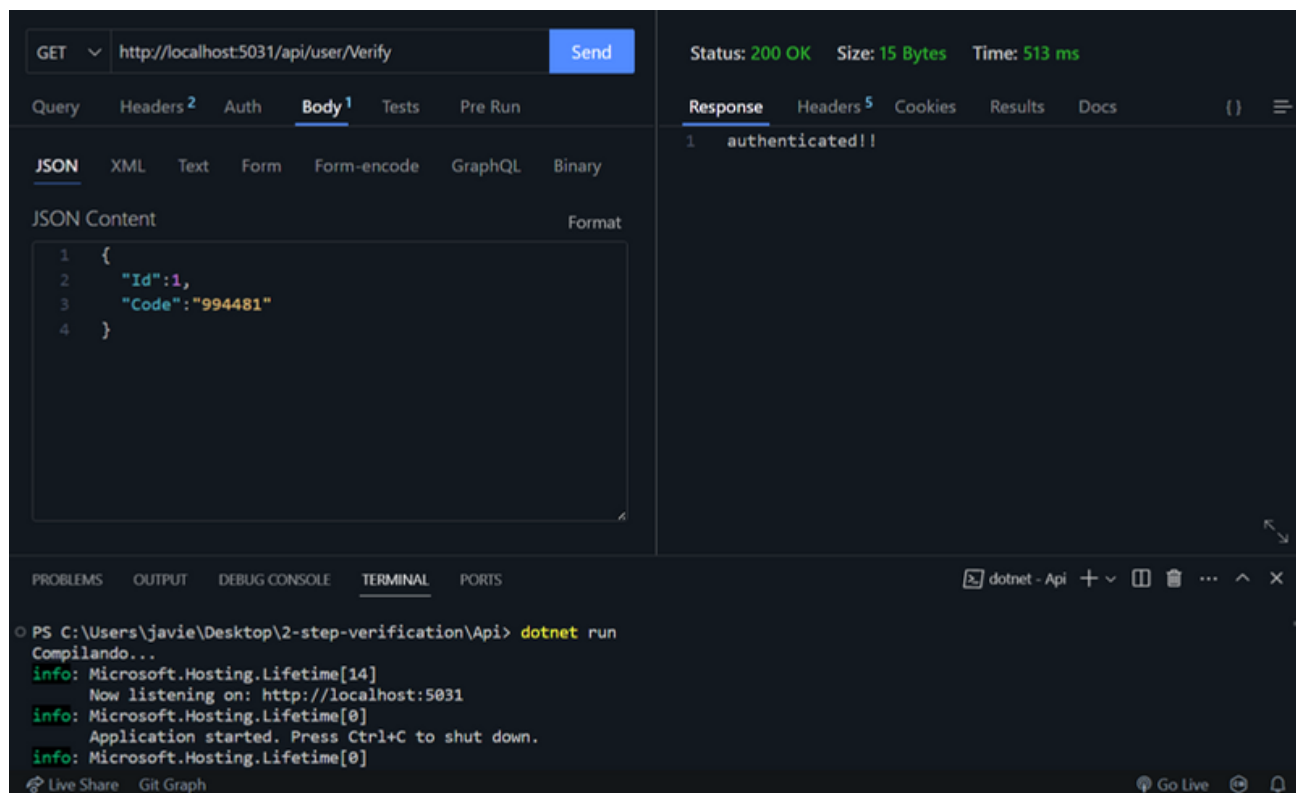
### 3 CONFIGURACIÓN DE LA DB Y MIGRACIONES.



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



**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.**