

**Evaluación Parcial 2: CRUD para Sistema de Gestión de Citas Médicas con Análisis
y Resolución de Conflictos en SonarQub**

Isaac Escobar, Josue Ferrin

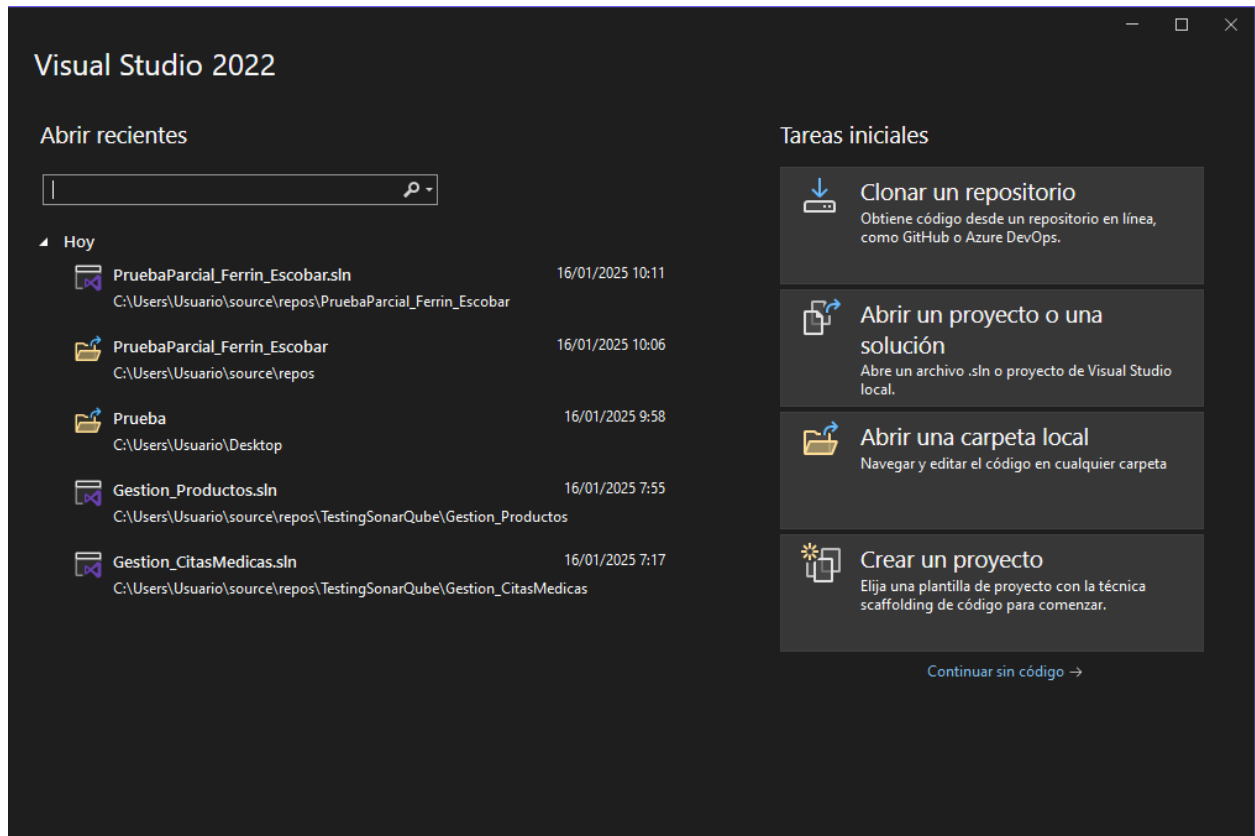
Departamento Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE

NRC 2530: Pruebas de Software

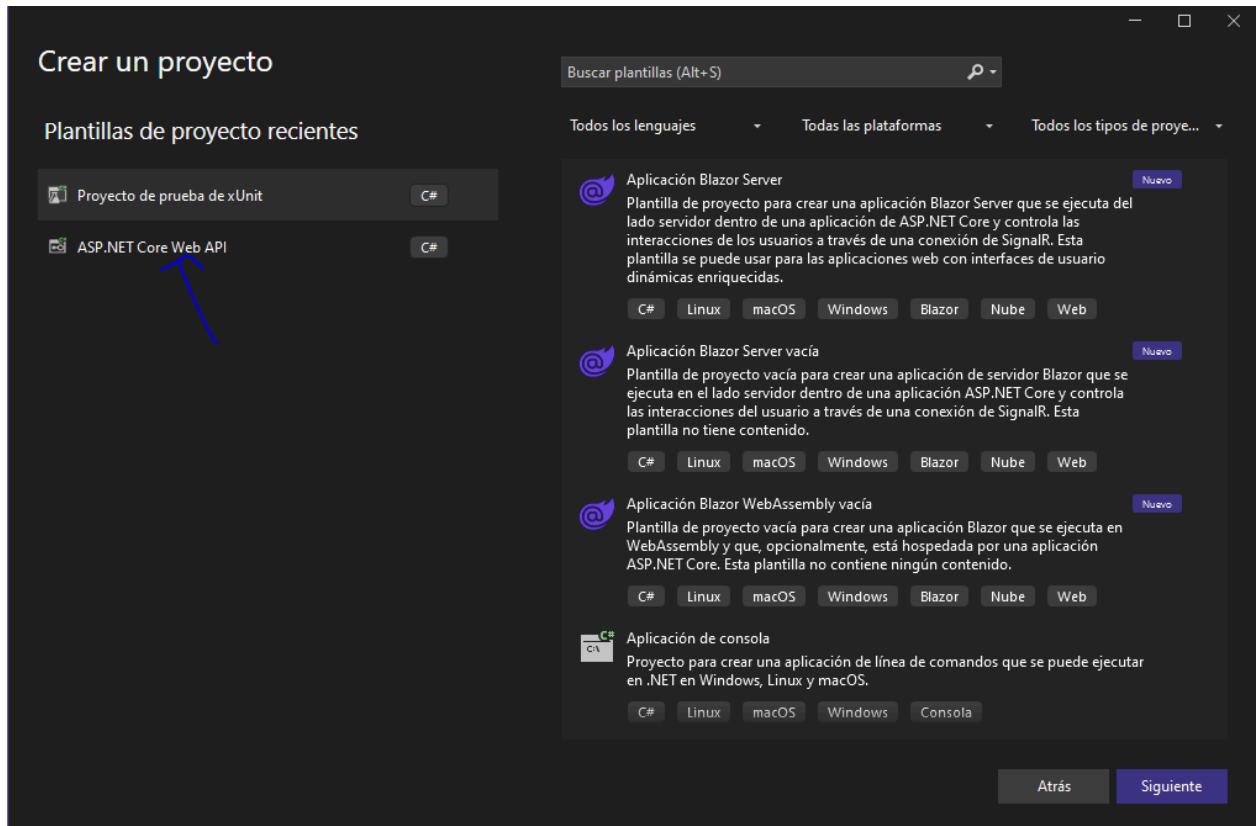
Ing. Diego Gamboa Mgtr.

16/01/2025

Los pasos que se siguieron para hacer el proyecto



Una vez elegido para hacer el proyecto escoges



Elegimos lo que esta señalado la ASP.NET Core Web API para poder seguir con el proyecto que esto nos permite hacer Api

De ahí se elegí un nombre para el proyecto

Configure su nuevo proyecto

Proyecto de prueba de xUnit C# Linux macOS Windows Escritorio Prueba Web xUnit

Nombre del proyecto

TestProject1

Ubicación

C:\Users\Usuario\source\repos\TestingSonarQube

Nombre de la solución ⓘ

TestProject1

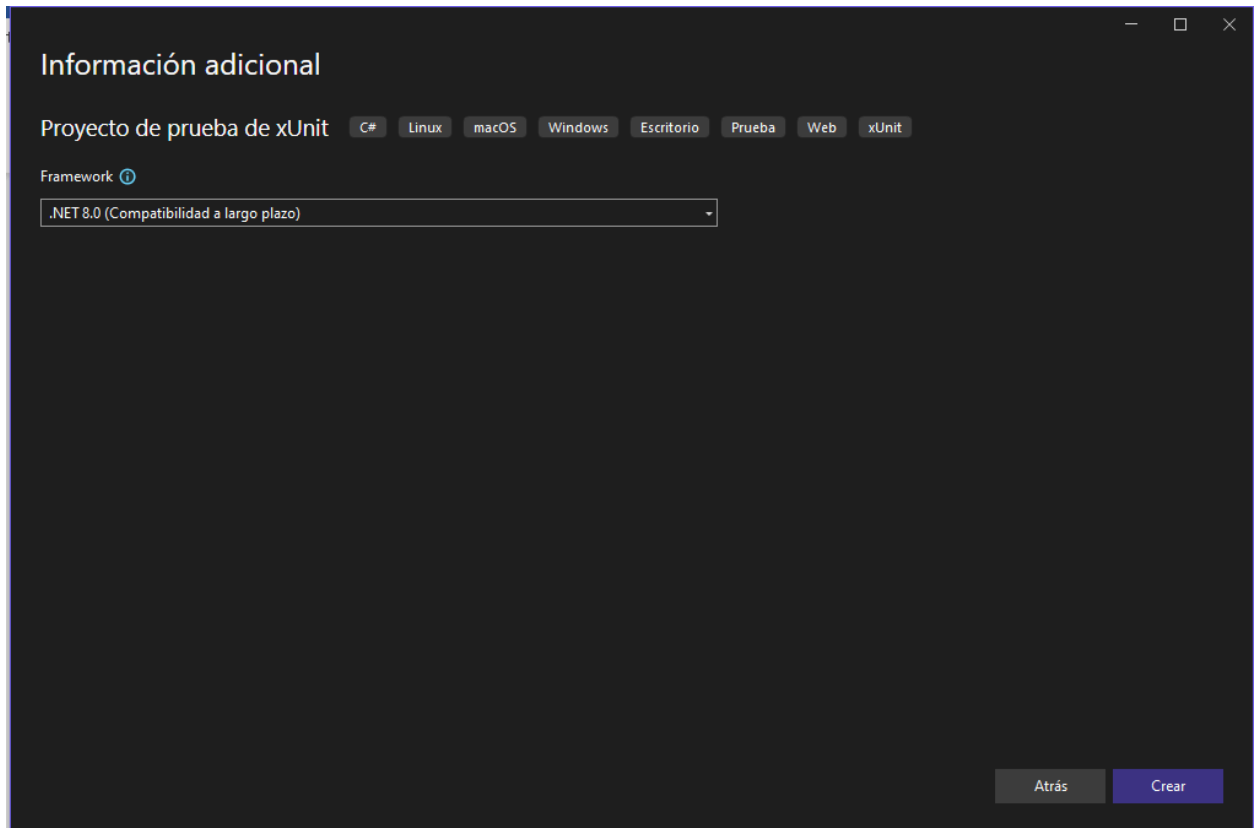
☐ Colocar la solución y el proyecto en el mismo directorio

Proyecto se creará en "C:\Users\Usuario\source\repos\TestingSonarQube\TestProject1\TestProject1"

Atrás Siguiente

Y De ahí le damos a siguiente

Nos va salir esta ventana



Información adicional

Proyecto de prueba de xUnit C# Linux macOS Windows Escritorio Prueba Web xUnit

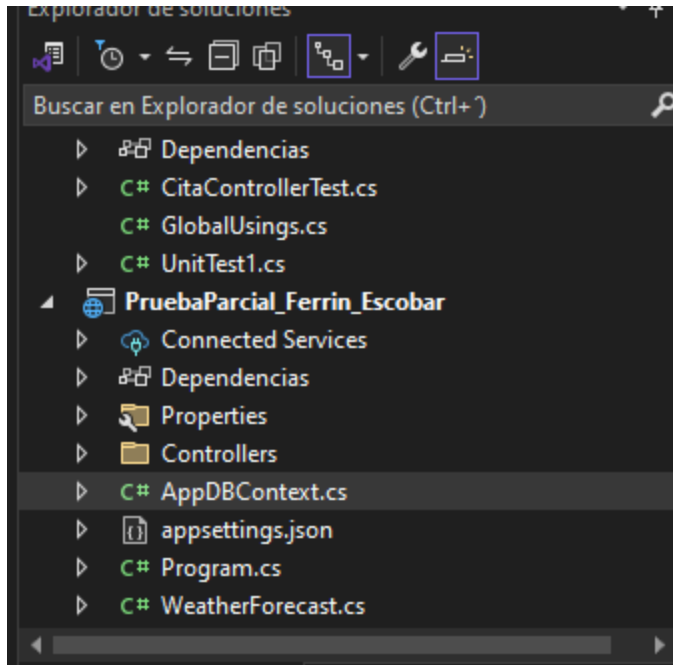
Framework ⓘ

.NET 8.0 (Compatibilidad a largo plazo)

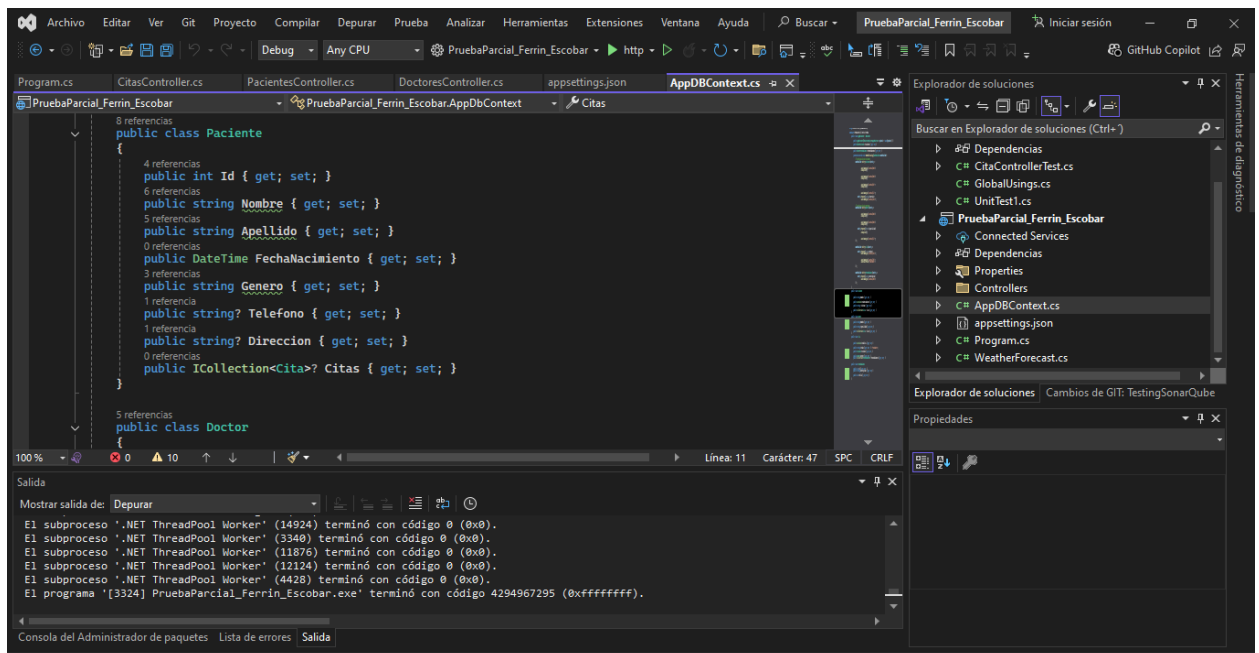
Atrás Crear

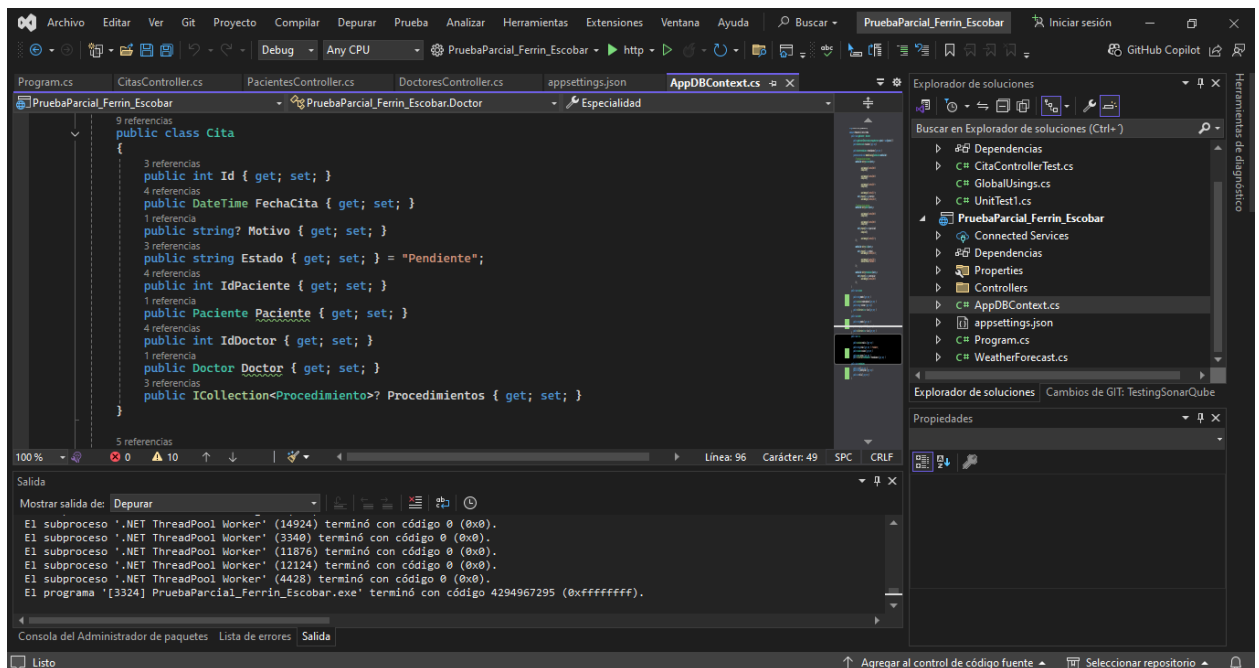
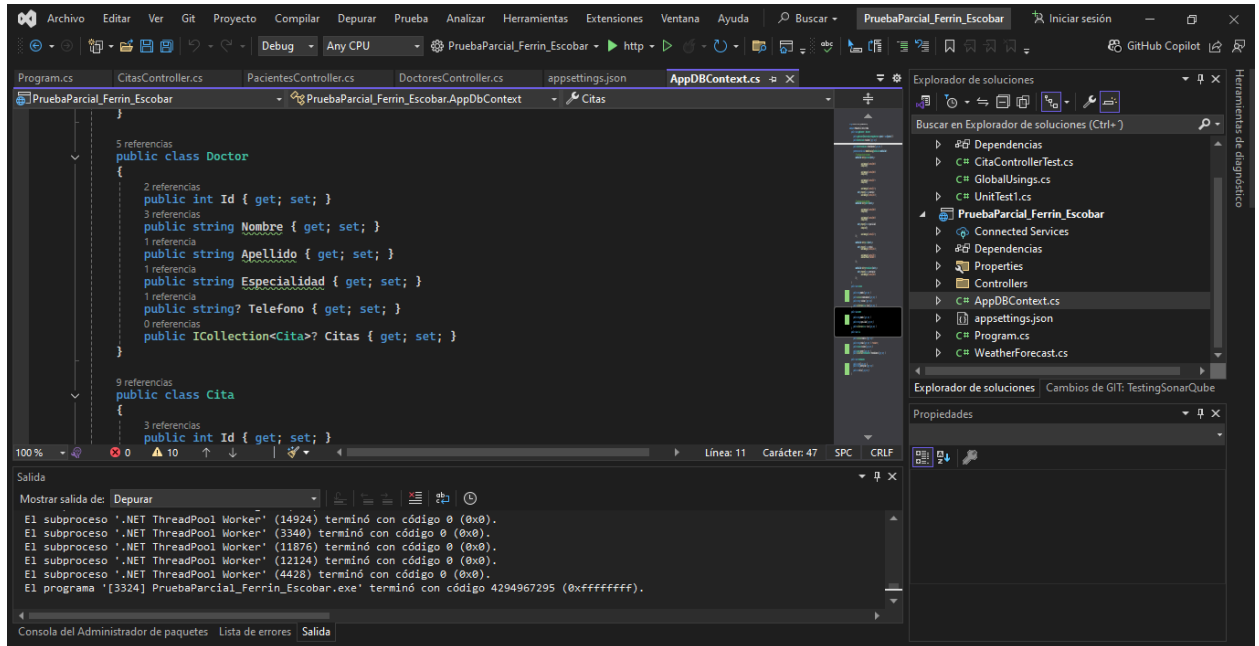
Tuvimos que tener bastante en cuenta en cual nos sirve porque no nos funciona la versión mas reciente y para que funcione bien

Una vez que lo creamos comenzamos a crear las clases que necesitamos crear las tablas correspondientes:



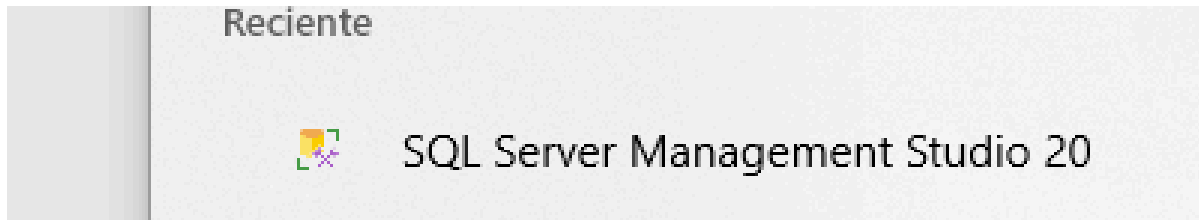
Primero creamos AppDbContext lo cual nos permite hacer las tablas de cada uno con sus parámetros correspondientes



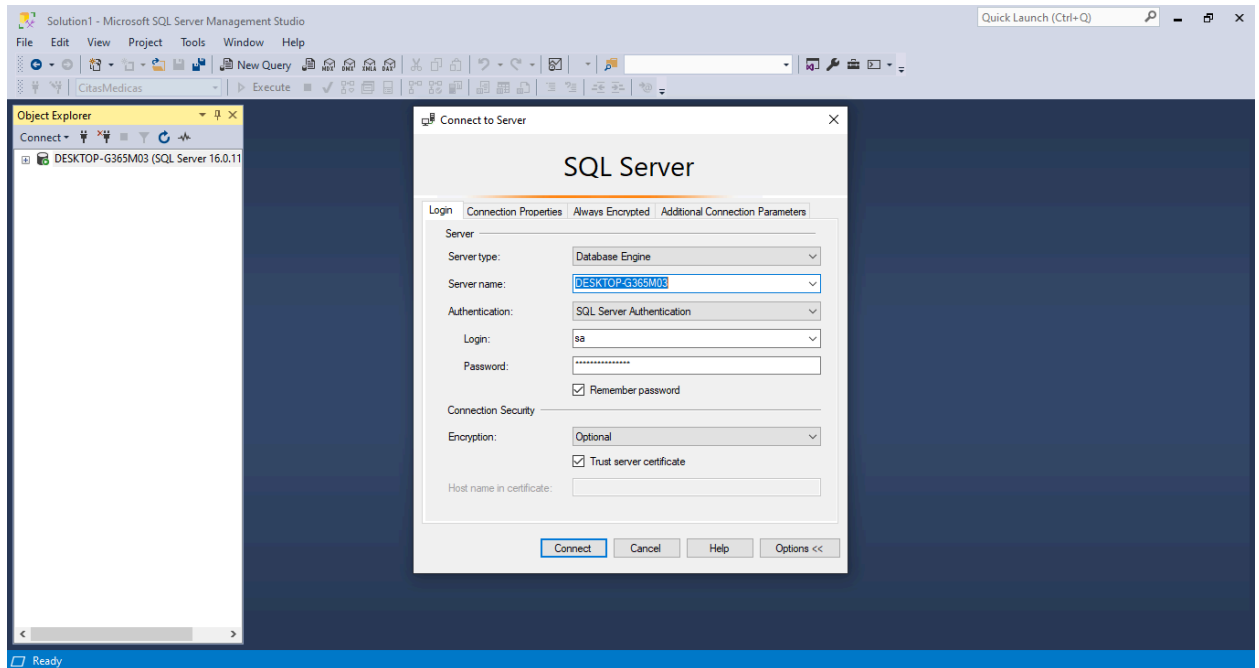


Listo una vez hecho las tablas, nos vamos a nuestro programa SQL Server Management

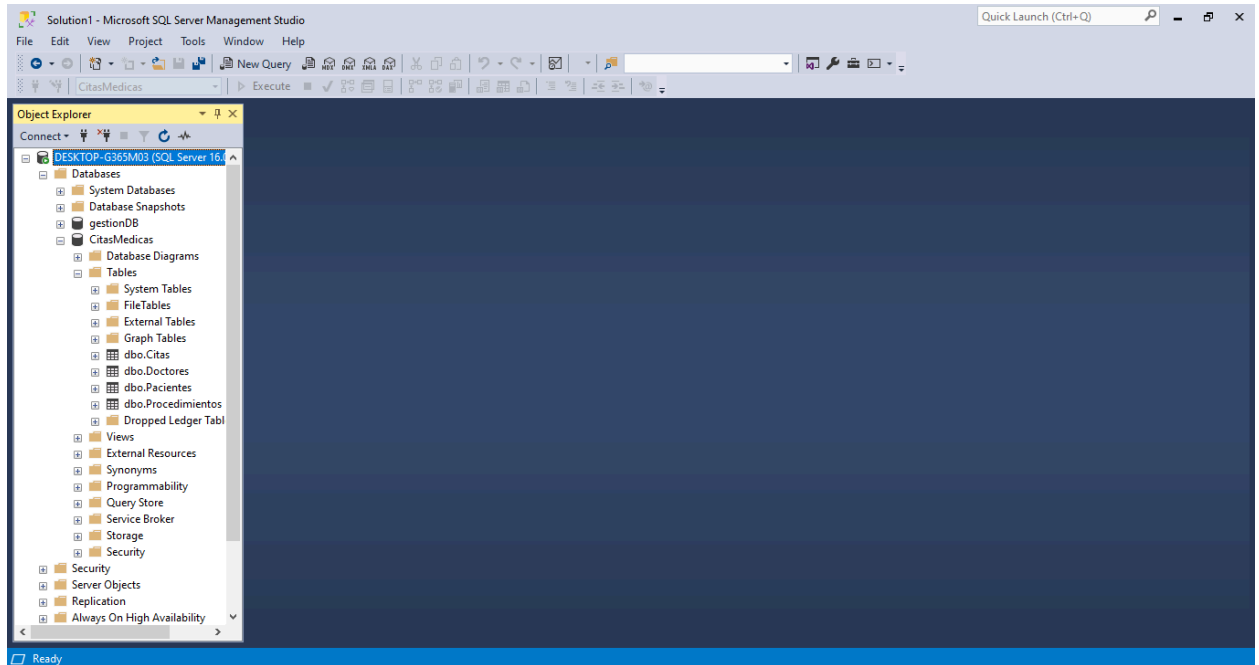
Studio



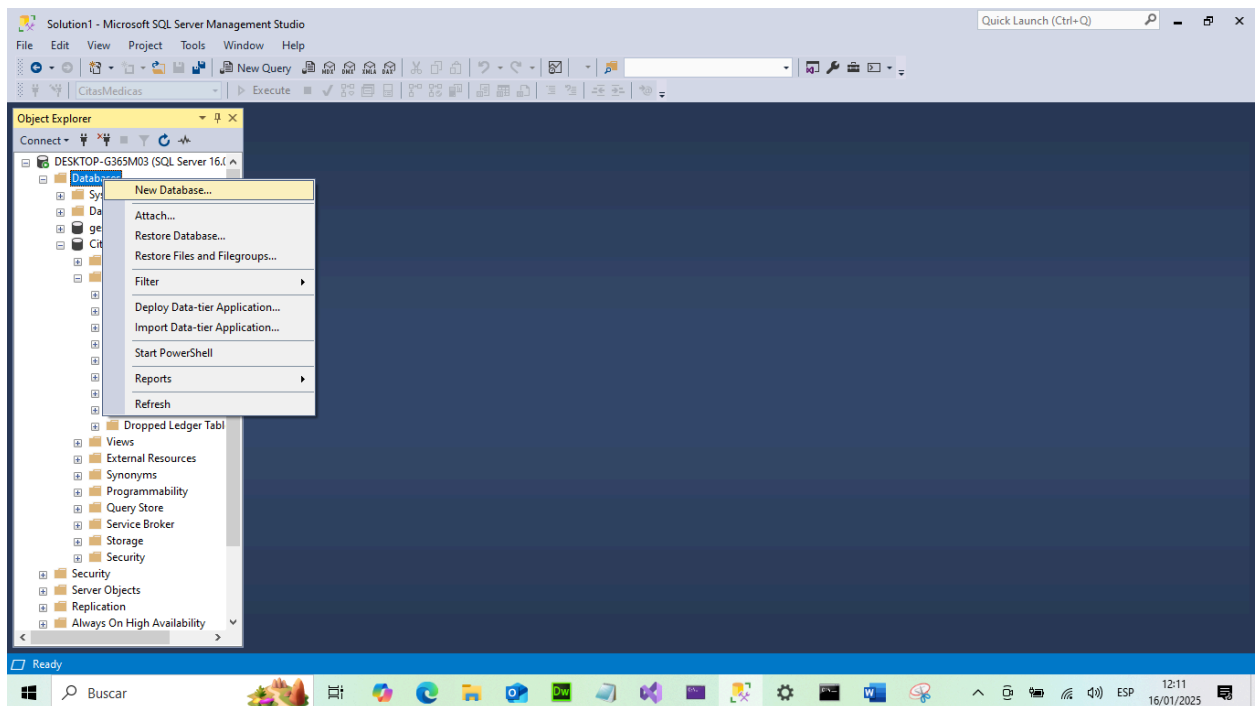
Una vez abierto nos salió esta pantalla

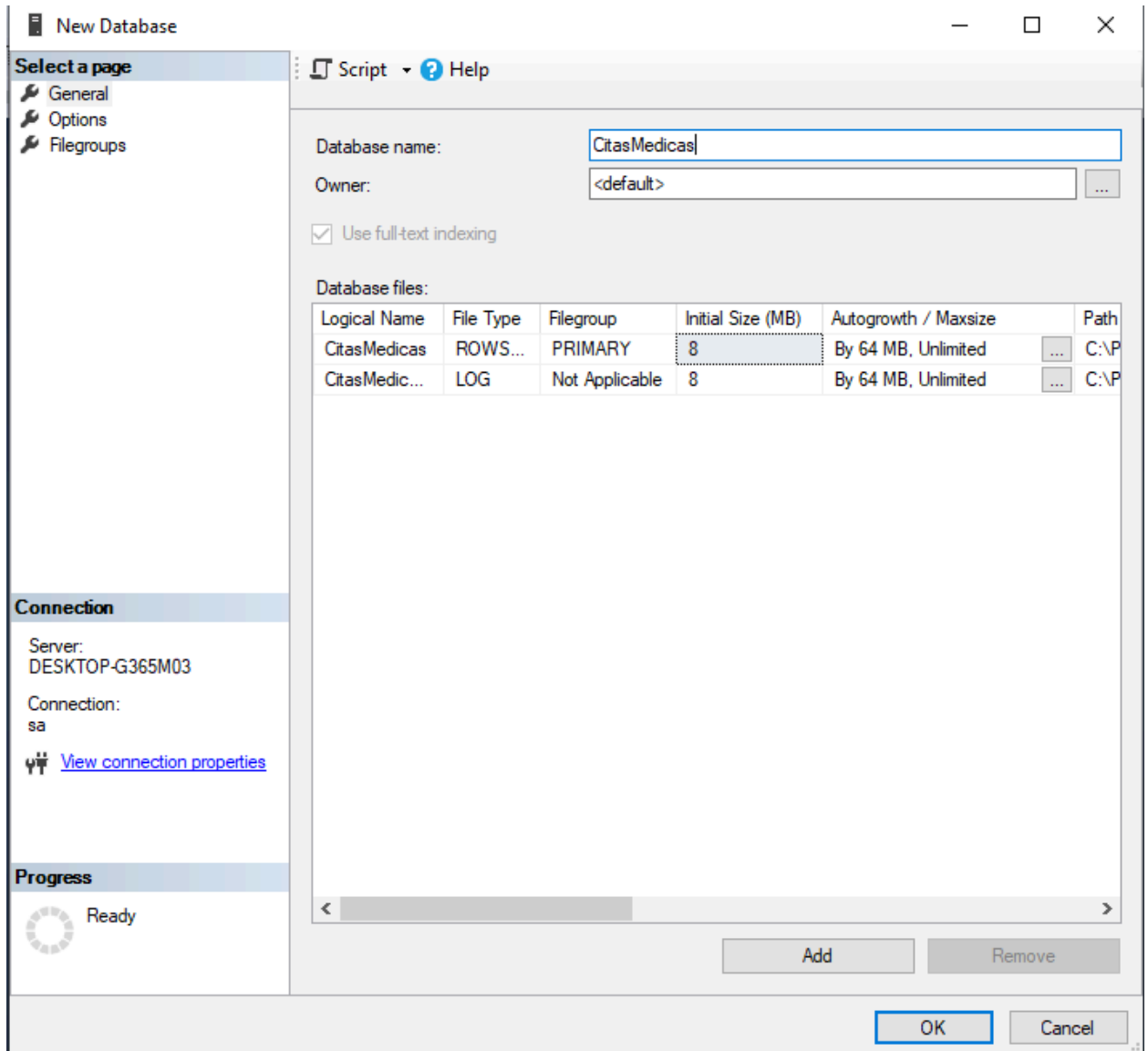


Iniciamos y nos muestra todo lo que nosotros tenemos

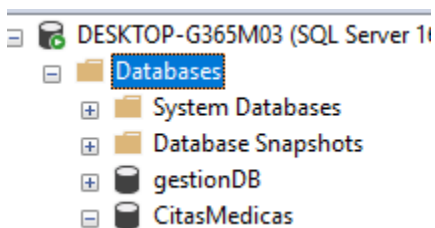


Ahora lo que hacemos es crear una nueva base de datos para lo que necesitamos

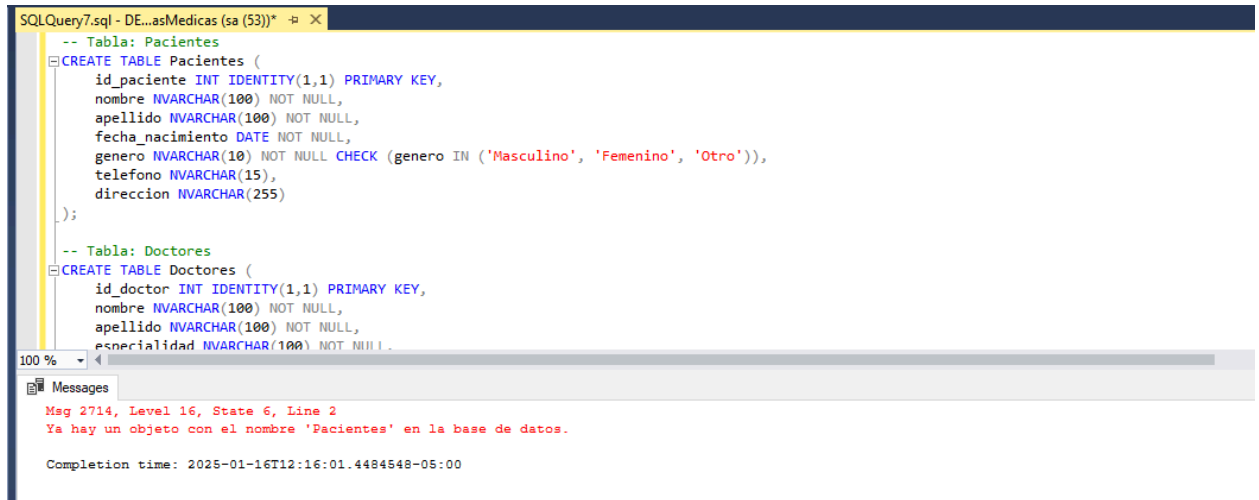




Una vez en la pantalla le dimos nombre a nuestra base de datos y le pusimos ok y logro crearse



Una vez que creamos nuestra base creamos un nuevo archivo para poder crear las tablas basándonos en los valores que dimos anteriormente a nuestras tablas



```
-- Tabla: Pacientes
CREATE TABLE Pacientes (
    id_paciente INT IDENTITY(1,1) PRIMARY KEY,
    nombre NVARCHAR(100) NOT NULL,
    apellido NVARCHAR(100) NOT NULL,
    fecha_nacimiento DATE NOT NULL,
    genero NVARCHAR(10) NOT NULL CHECK (genero IN ('Masculino', 'Femenino', 'Otro')),
    telefono NVARCHAR(15),
    direccion NVARCHAR(255)
);

-- Tabla: Doctores
CREATE TABLE Doctores (
    id_doctor INT IDENTITY(1,1) PRIMARY KEY,
    nombre NVARCHAR(100) NOT NULL,
    apellido NVARCHAR(100) NOT NULL,
    especialidad NVARCHAR(100) NOT NULL
);
```

100 %

Messages

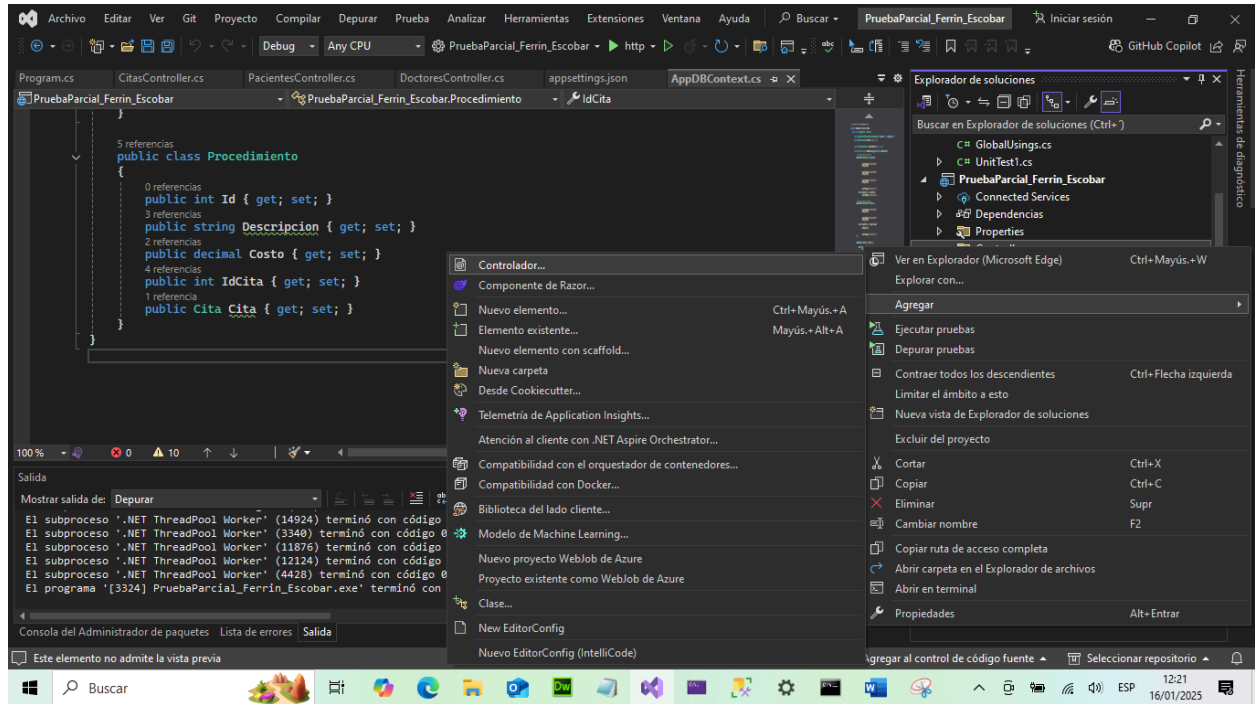
Msg 2714, Level 16, State 6, Line 2
Ya hay un objeto con el nombre 'Pacientes' en la base de datos.

Completion time: 2025-01-16T12:16:01.4484548-05:00

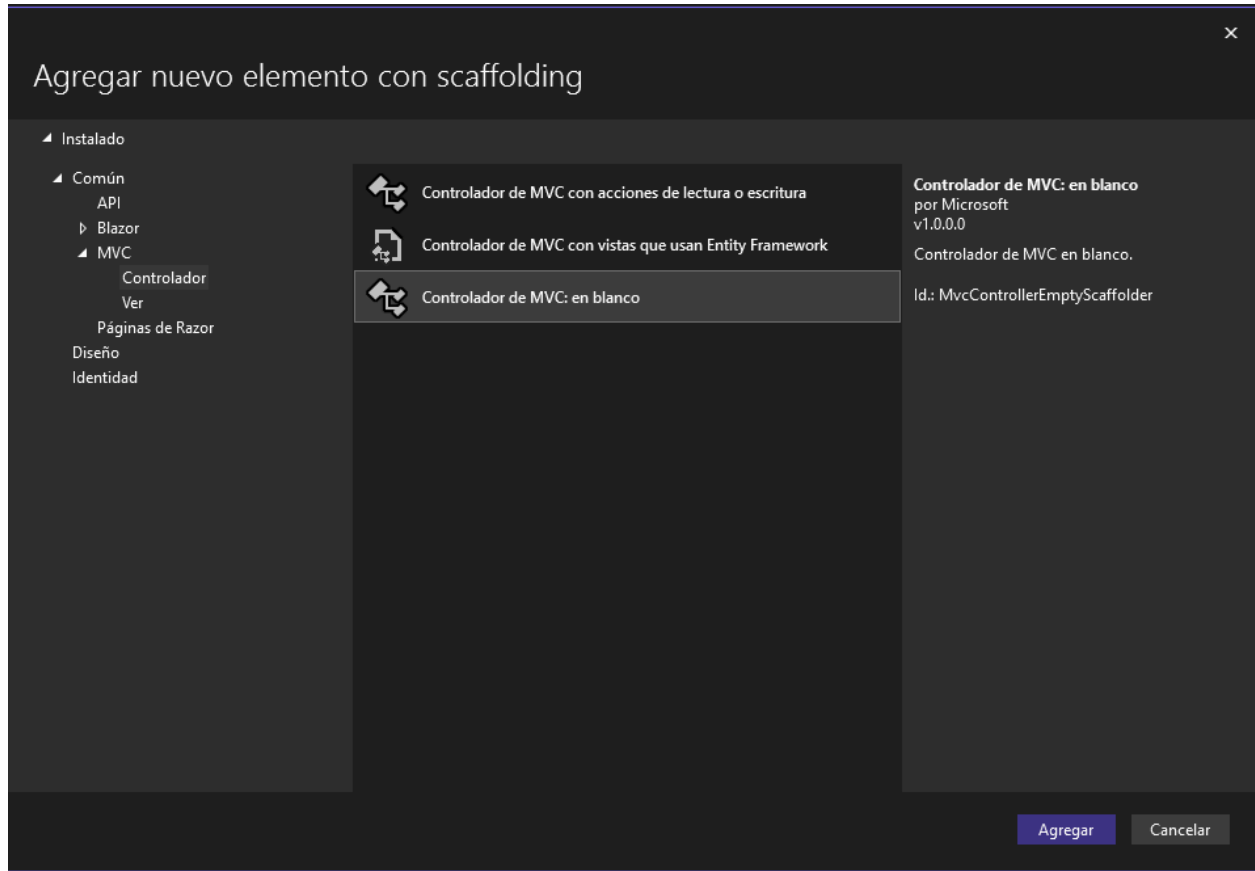
Y las insertamos en este caso ya los esta insertas y se puede ver en el mensaje que dice que el objeto ya tiene el nombre de Pacientes

Entonces ahorita volvemos a nuestro visual estudio y creamos los controles correspondientes de cada tabla que hicimos anteriormente

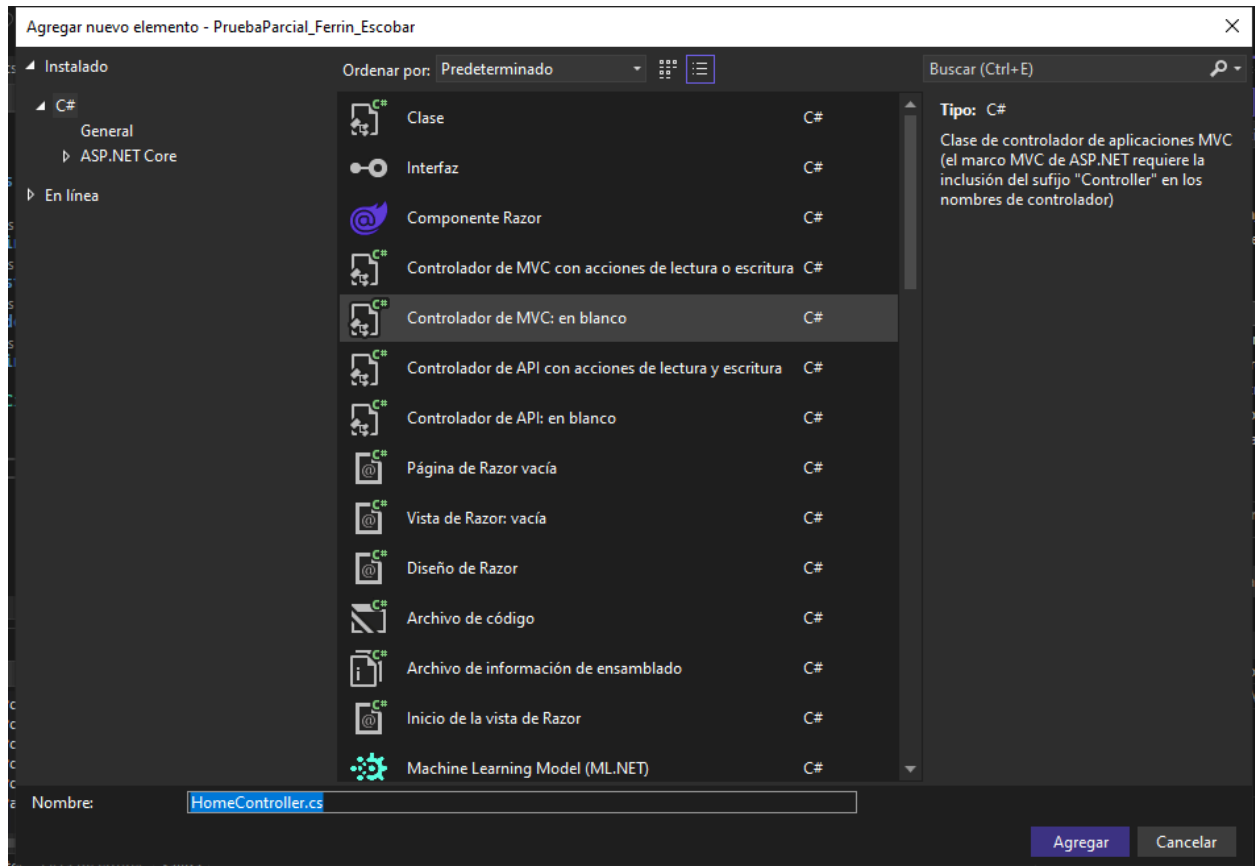
Para crearlo necesitamos dar click derecho en controllers y después irnos a la opción de agregar y seleccionar controlador



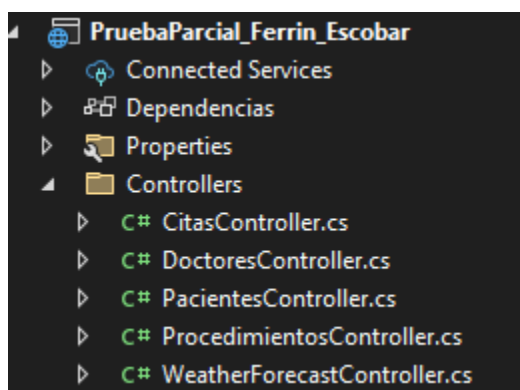
Una vez que ya elegimos controlador nos salió esta pantalla y elegimos



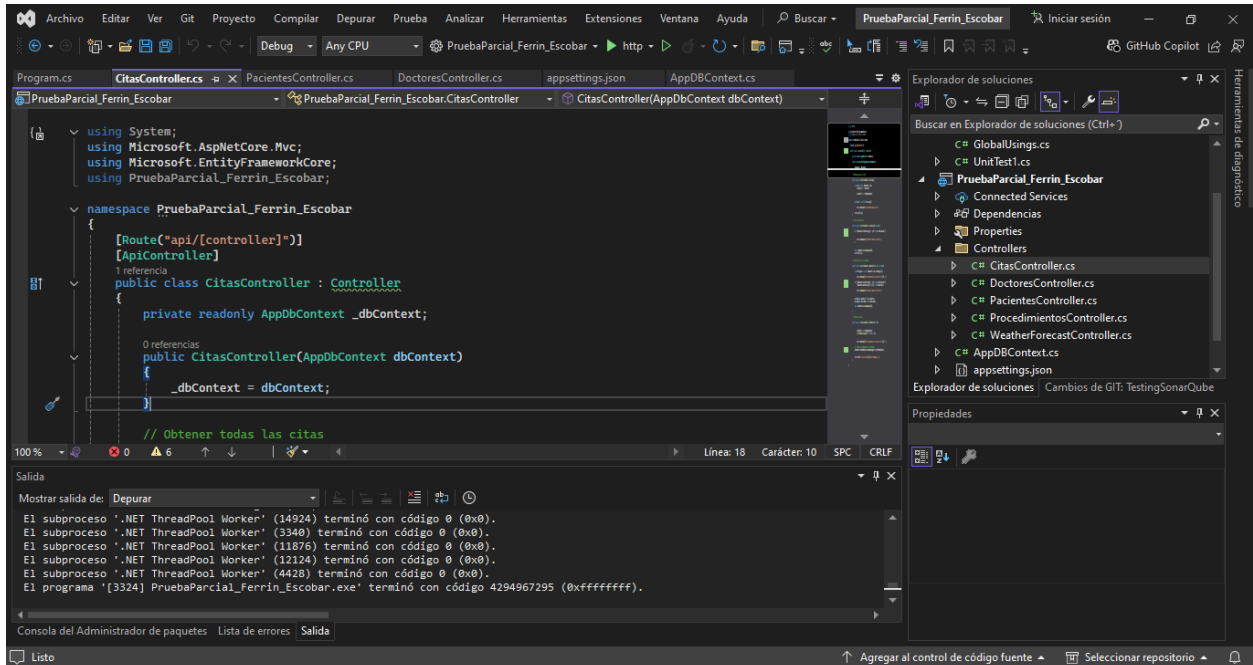
Una vez que hayamos elegido esa opción nos manda para poner nombre a los controladores y se crear con un .cs



Y así vamos creando cada controlador y repetimos 3 veces mas el mismo proceso y dándole nombre a cada controlador y quedo de la siguiente manera:

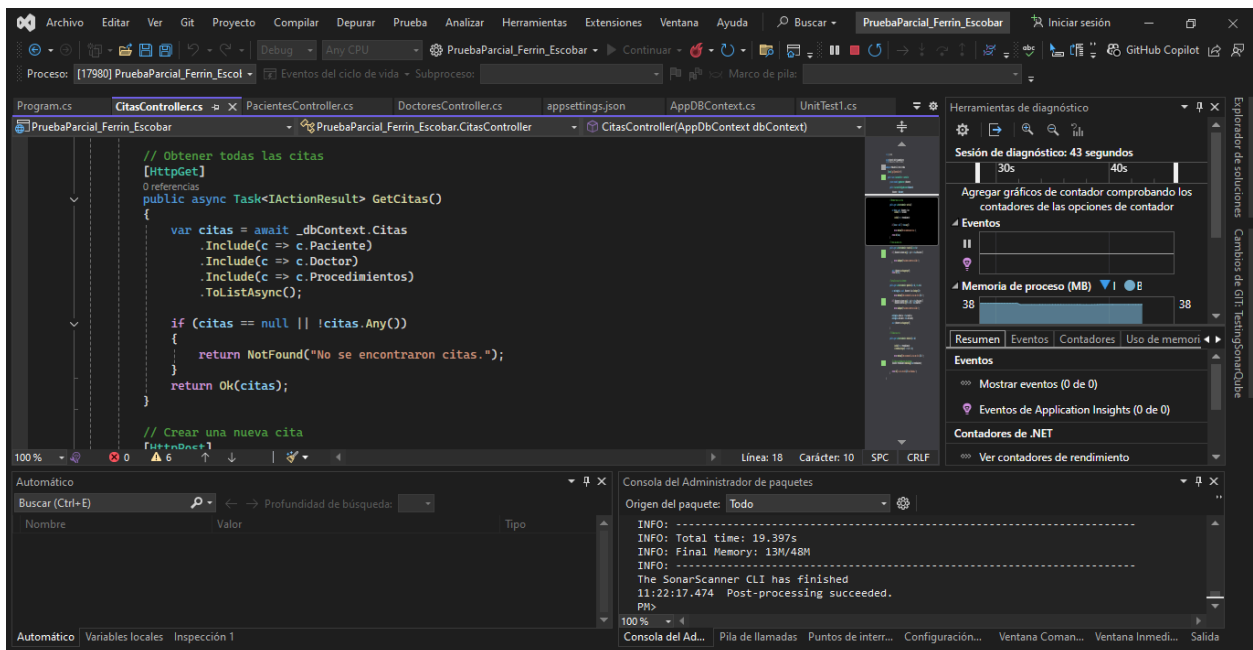


Podemos ver que ya hemos creado los controles de cada uno, nos aseguramos de poner cada parámetro como teníamos en las tablas

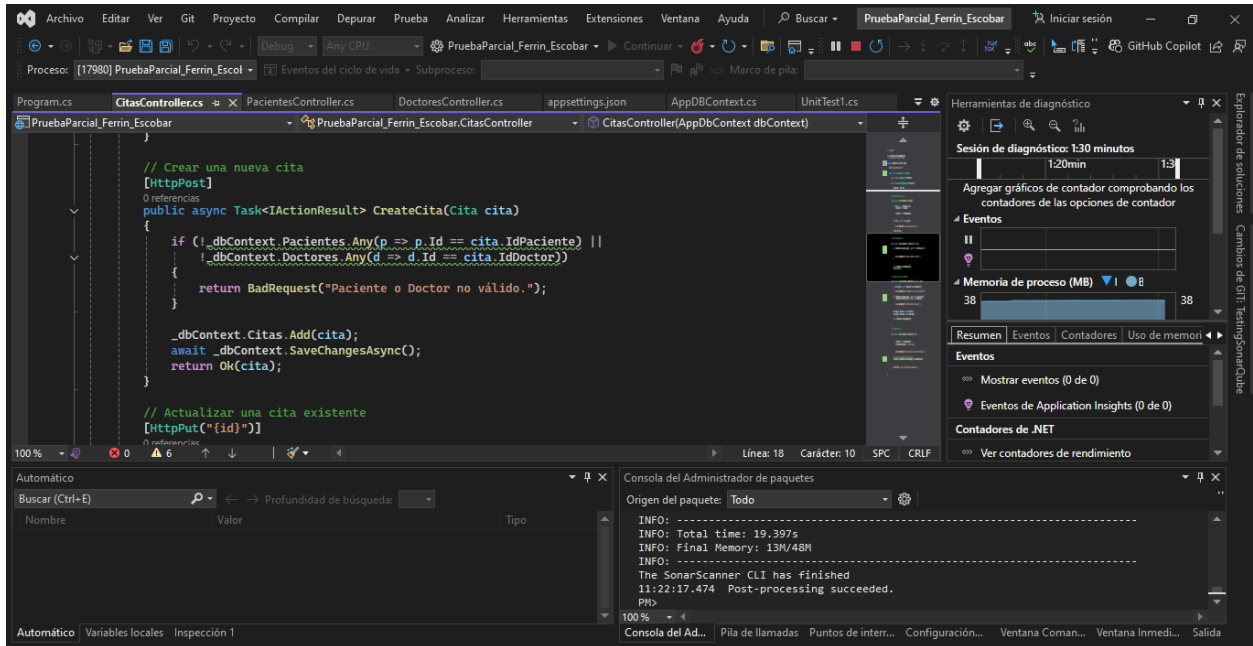


Una vez creados procedimos a crear a cada uno su post, get, delete, put

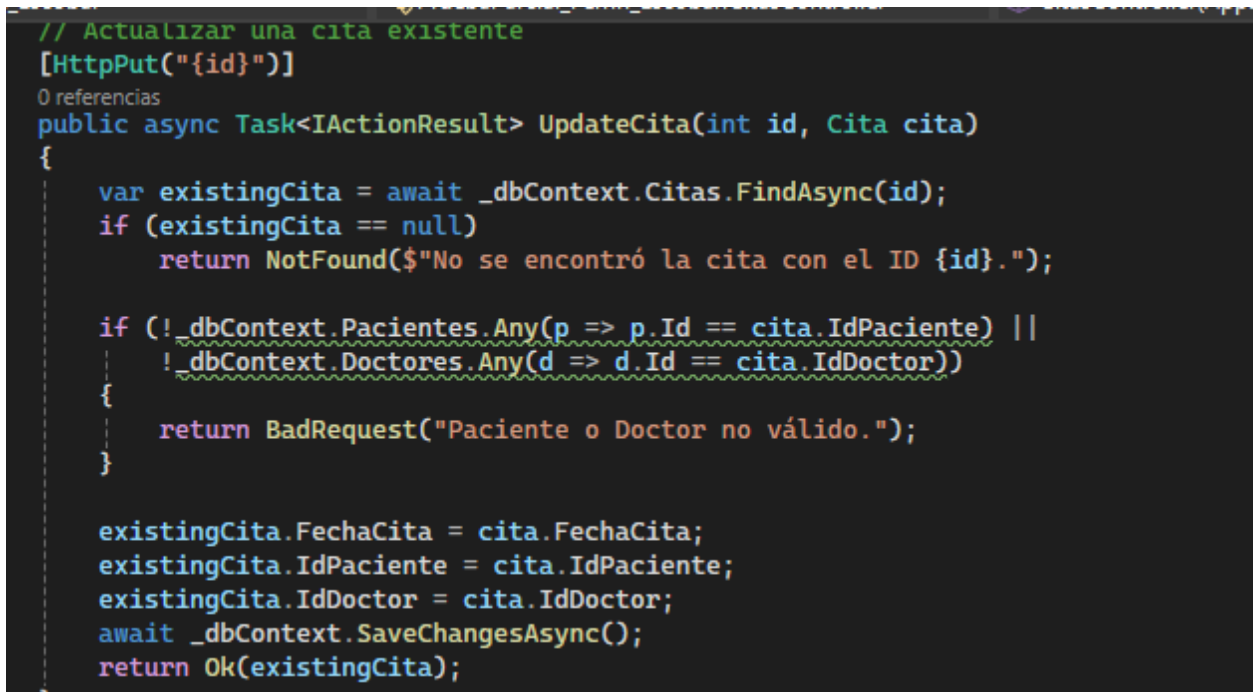
Aquí tenemos el Get



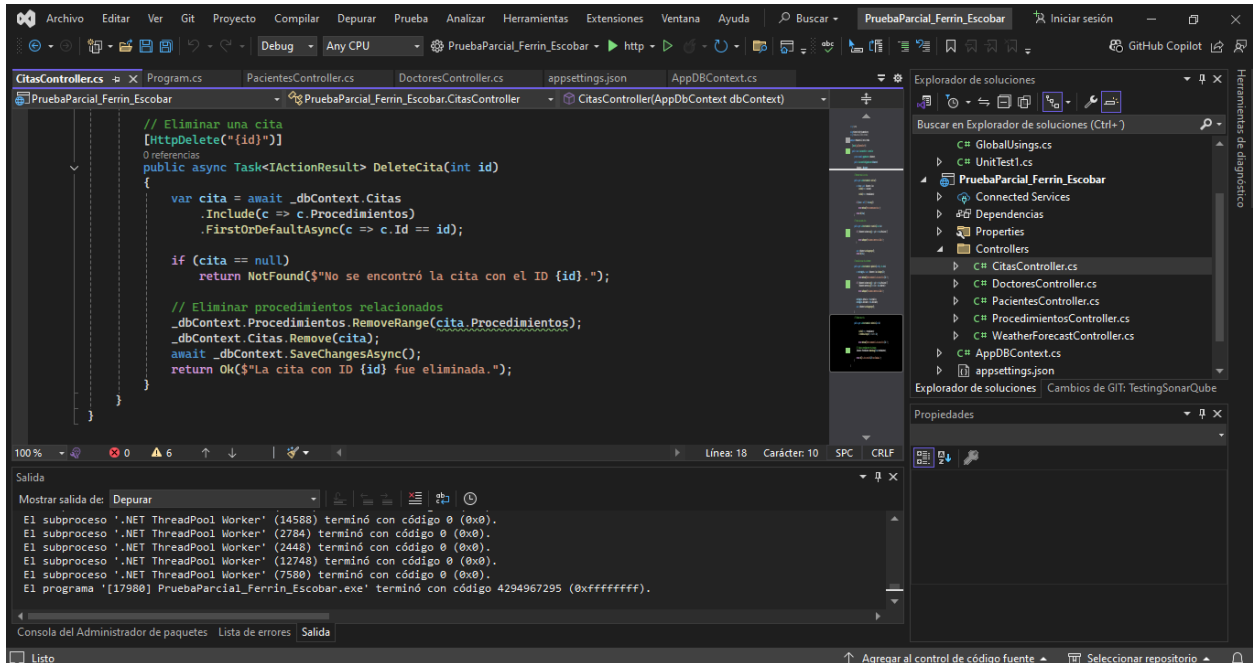
Aquí tenemos nuestro POST



Aquí tenemos nuestro PUT

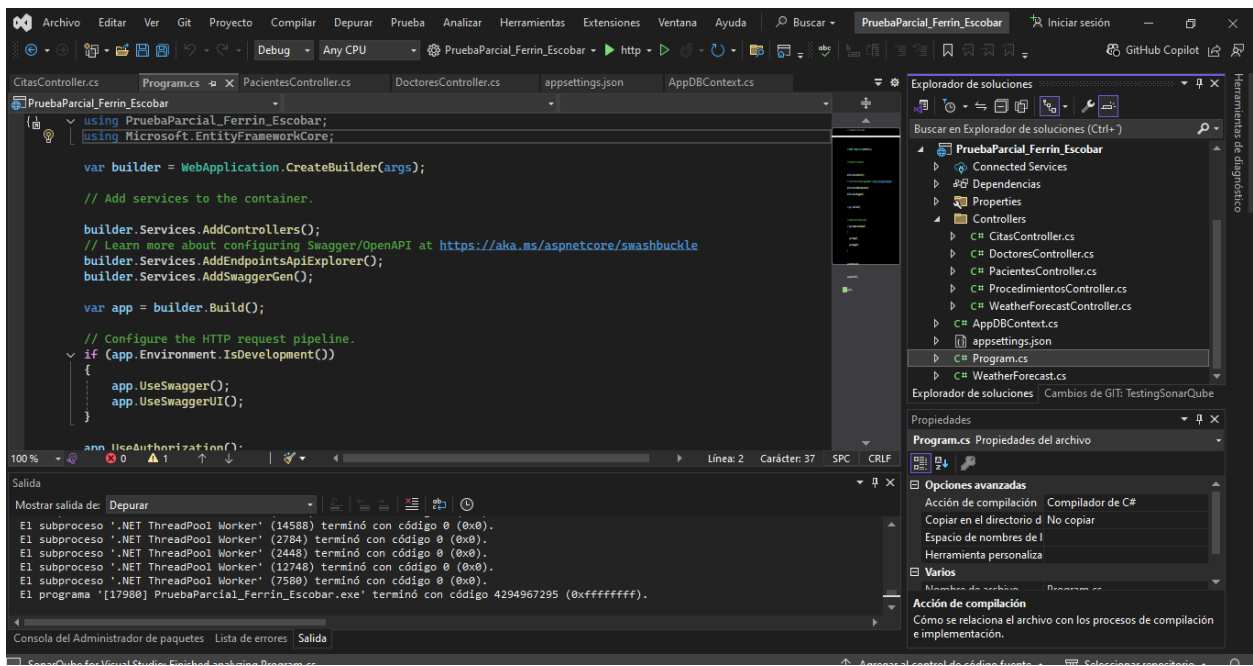


Aquí tenemos nuestro DELETE

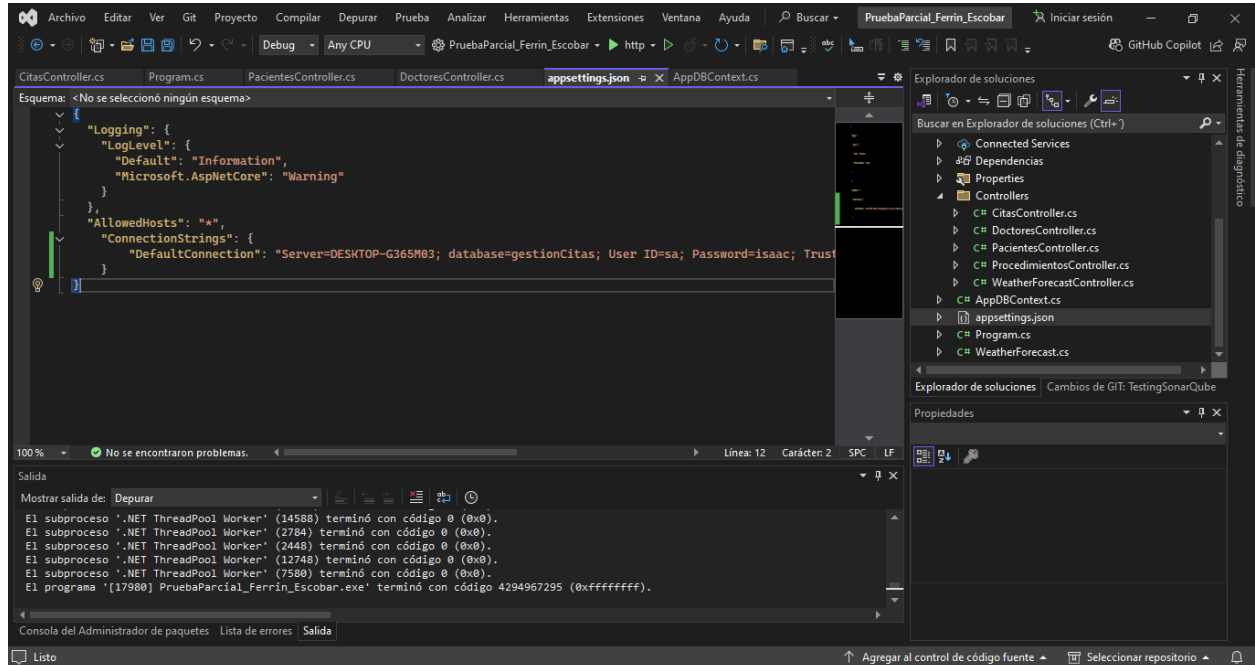


Repetimos el mismo proceso para los demás controladores y funcionan correctamente

Ahora para que nos funcione correctamente añadimos las librerías faltantes en el program.cs para que funcione de mejor manera

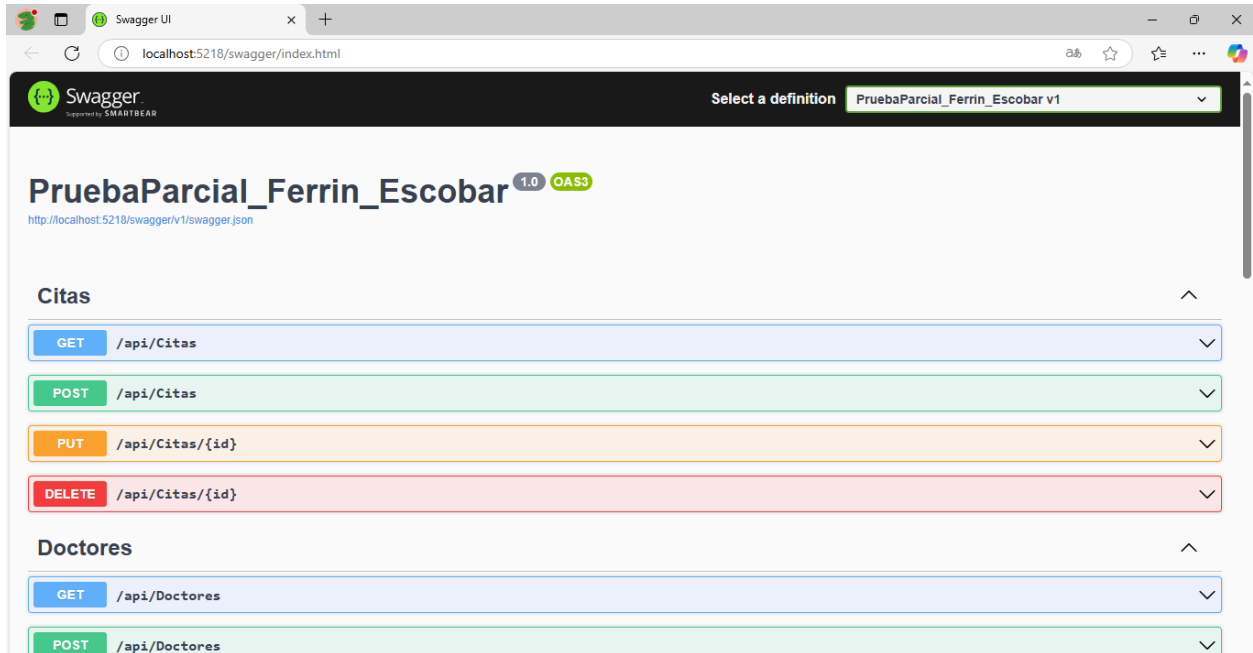


Y por el ultimo para que funcione correctamente configuramos nuestro appsetting. json para que se pueda conectar a nuestra base de datos y pueda mostrar los datos

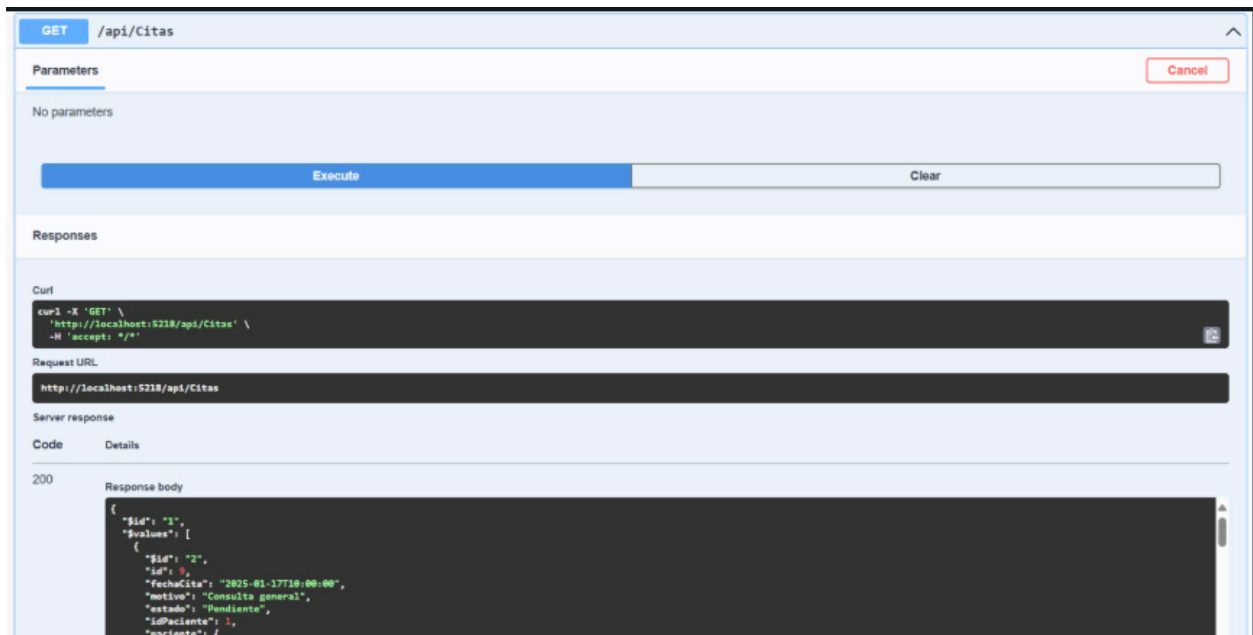


Eso seria para tener todo listo para que nuestro proyecto funcione correctamente, nos muestra nuestra api creada con todo lo que pusimos

Esta es nuestra interfaz



Y así se ve cada uno de nuestras tablas



GET /api/Doctores

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5218/api/Doctores' \
  -H 'accept: */*'
```

Request URL

http://localhost:5218/api/Doctores

Server response

Code Details

200

Response body

```
{
  "id": "2",
  "values": [
    {
      "id": "2",
      "id": 1,
      "nombre": "Dr. Laura",
      "apellido": "Ramirez",
      "especialidad": "Pediatría",
      "telefono": "555-4321",
      "citas": null
    }
  ]
}
```

GET /api/Pacientes

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5218/api/Pacientes' \
  -H 'accept: */*'
```

Request URL

http://localhost:5218/api/Pacientes

Server response

Code Details

200

Response body

```
{
  "id": "1",
  "values": [
    {
      "id": "2",
      "id": 1,
      "nombre": "Juan",
      "apellido": "Pérez",
      "fechaNacimiento": "1985-03-15T00:00:00",
      "genero": "Masculino",
      "telefono": "555-1234",
      "citas": null
    }
  ]
}
```

GET /api/Procedimientos

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  http://localhost:5218/api/Procedimientos \
  -H 'accept: */*'
```

Request URL

http://localhost:5218/api/Procedimientos

Server response

Code Details

200

Response body

```
{
  "id": "1",
  "values": [
    {
      "id": "2",
      "id": 1,
      "descripcion": "Examen de sangre",
      "costo": 200,
      "idCita": 9,
      "cita": {
        "id": 9
      }
    }
  ]
}
```

Code Details

200

Response body

```
{
  "id": "1",
  "id": 1,
  "nombre": "María",
  "apellido": "García",
  "fechaNacimiento": "2025-01-16T17:23:38.804Z",
  "genero": "femenino",
  "telefono": "0989112225",
  "direccion": "Avenida Albondiga",
  "citas": {
    "id": "2",
    "values": []
  }
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Thu, 16 Jan 2025 17:27:32 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	Success	No links

POST /api/Doctores

Parameters

Cancel

Reset

No parameters

Request body

application/json

```
{  "nombre": "Dr. Steven",  "apellido": "Albuja",  "especialidad": "Fisioterapeuta",  "telefono": "0989112236",  "citas": []}
```

Execute

Clear

Responses

Request URL

http://localhost:5218/api/Doctores

Server response

Code	Details
200	<div>Response body<div><pre>{ "id": "1", "id": 1, "nombre": "Dr. Steven", "apellido": "Albuja", "especialidad": "Fisioterapeuta", "telefono": "0989112236", "citas": { "id": "2", "\$values": [] }}</pre></div><div><div>Download</div></div></div> <div>Response headers<div>content-type: application/json; charset=utf-8 date: Thu, 16 Jan 2025 17:33:45 GMT server: Kestrel transfer-encoding: chunked</div></div>

Responses

Code	Description	Links
200	Success	No links

POST /api/Pacientes

Parameters Cancel Reset

No parameters

Request body application/json

```
{  "nombre": "Maria",  "apellido": "Garcia",  "fechaNacimiento": "2025-01-16T17:23:38.804Z",  "genero": "femenino",  "telefono": "0989112225",  "direccion": "Avenida Albondiga",  "citas": []}
```

Execute Clear

Responses

POST /api/Citas

Parameters Cancel Reset

No parameters

Request body application/json

```
{  "tipo": "PERMANENTE",  "idPaciente": 4,  "paciente": {    "nombre": "Maria",    "apellido": "Garcia",    "fechaNacimiento": "1990-01-01T00:00:00Z",    "genero": "femenino",    "telefono": "0989112225",    "direccion": "Avenida Albondiga"  },  "idDoctor": 3,  "doctor": {    "nombre": "Dr. Steven",    "apellido": "Albuja",    "especialidad": "Fisioterapeuta",    "telefono": "0989112236"  },  "procedimientos": []}
```

Execute Clear

The screenshot shows a REST client interface with a 'Code' tab selected. The status is 200. The response body is a JSON object representing a medical appointment and patient information. The response headers include content-type, date, server, and transfer-encoding.

Response body

```
{
  "id": "1",
  "id": 16,
  "fechaCita": "2025-01-16T17:12:57.849Z",
  "motivo": "Dolor de Cabeza",
  "estado": "Pendiente",
  "idPaciente": 4,
  "paciente": {
    "id": "2",
    "id": 5,
    "nombre": "María",
    "apellido": "García",
    "fechaNacimiento": "1990-01-01T00:00:00Z",
    "genero": "femenina",
    "telefono": "8989112225",
    "direccion": "Avenida Alameda",
    "citas": {
      "id": "3",
      "values": [
        {
          "id": "1"
        }
      ]
    }
  },
  "idDoctor": 3,
  "doctor": {
    "id": "1"
  }
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Thu, 16 Jan 2025 17:51:23 GMT
server: Kestrel
transfer-encoding: chunked
```

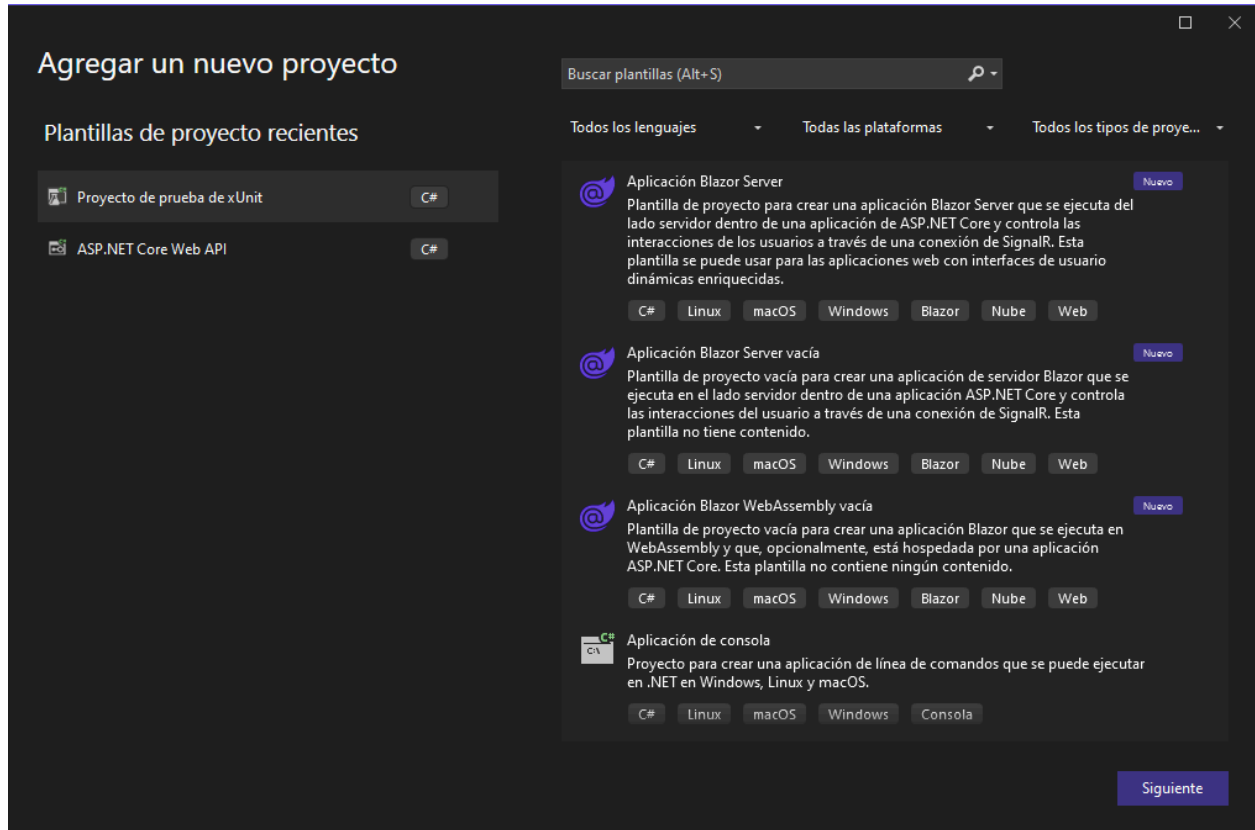
Responses

Code	Description	Links
200	Success	No links

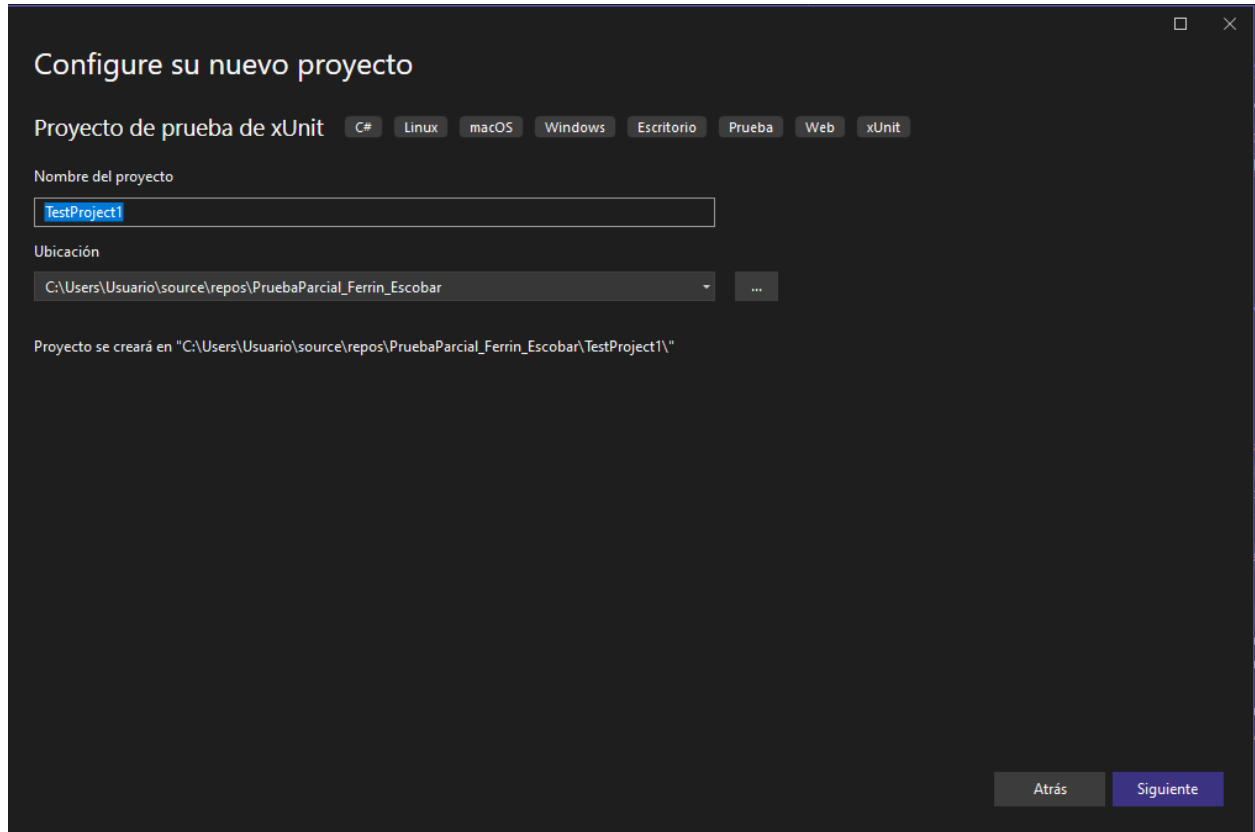
Funciono correctamente nuestro proyecto mostrándonos todo lo que hemos incluido en cada tabla con los POST, GET, DELETE, PUT

Después nos enfocamos en hacer las pruebas para ver si todo esta correcto hacemos un Tester con SonarQube, pero antes de entrar con SonarQube vamos a crear pruebas unitarias en nuestro proyecto

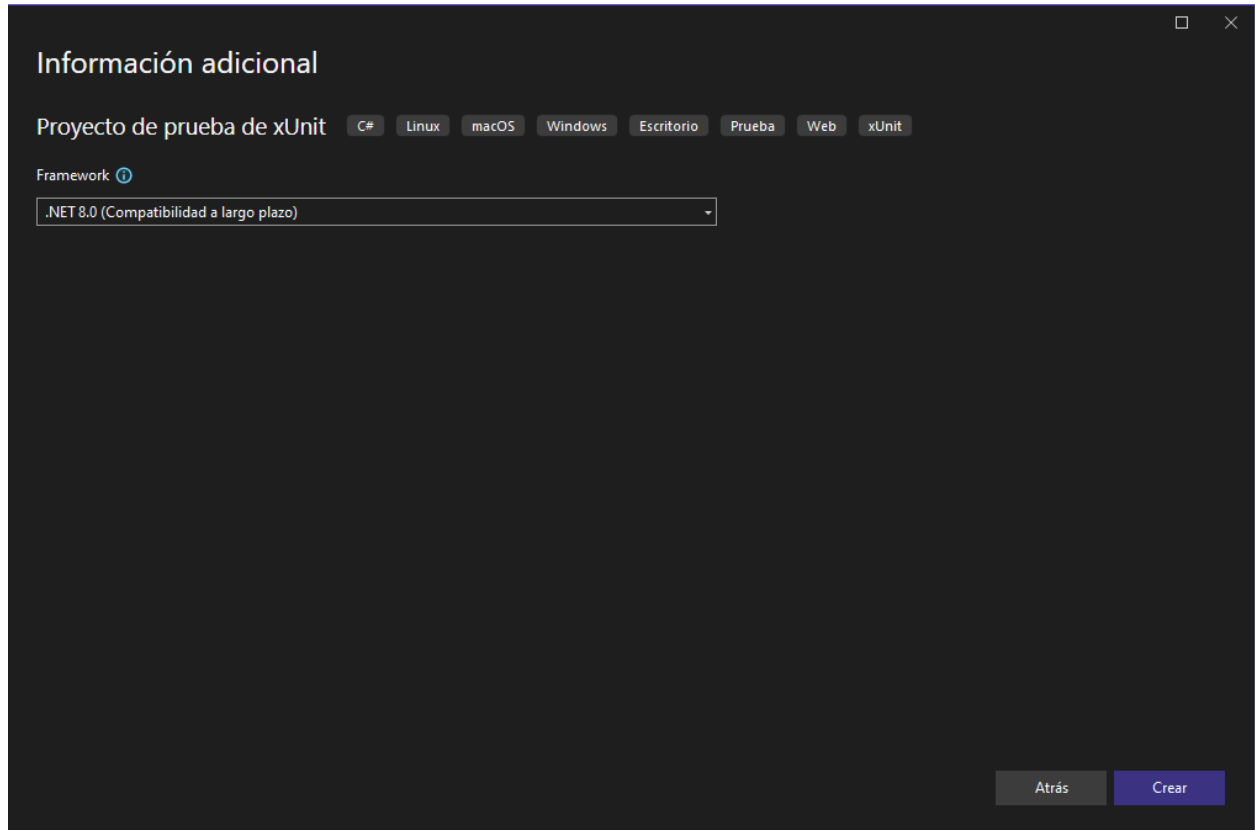
Elegimos el proyecto xUnix que nos sirve para poder hacer las pruebas



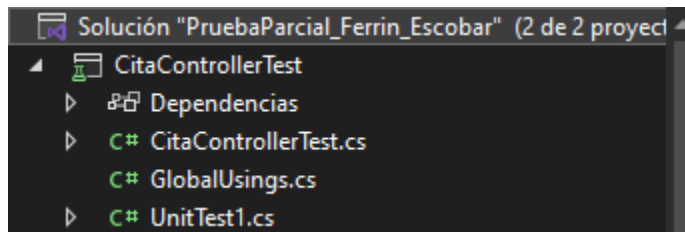
Le damos un nombre



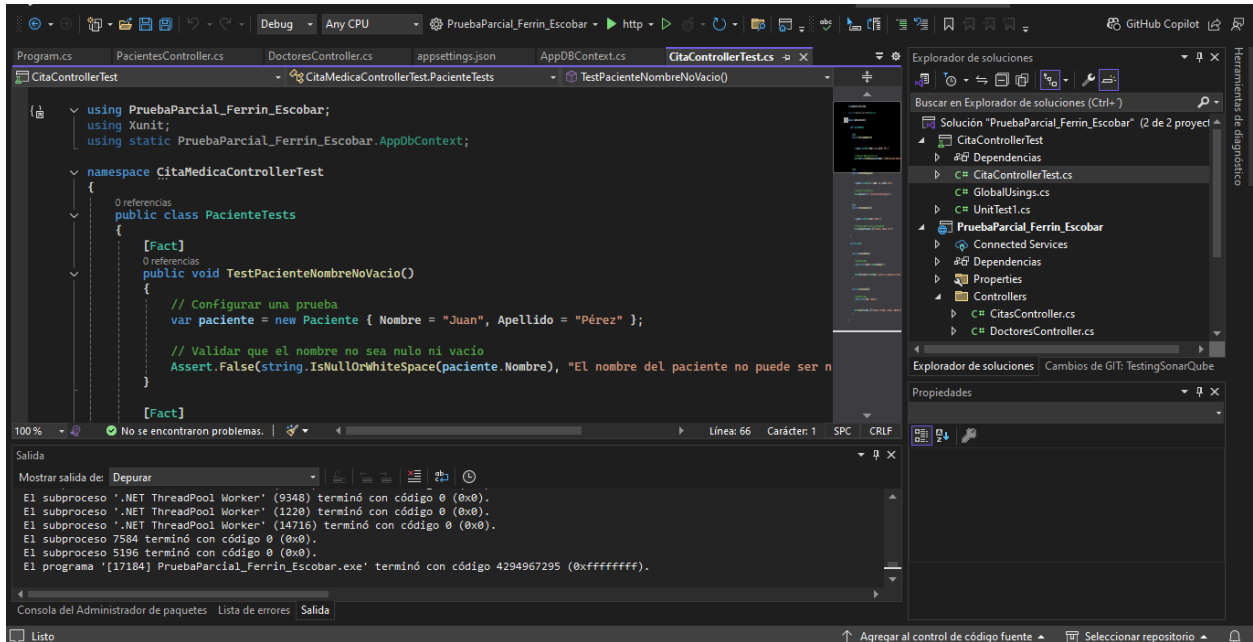
Mantenemos las configuraciones del Framework



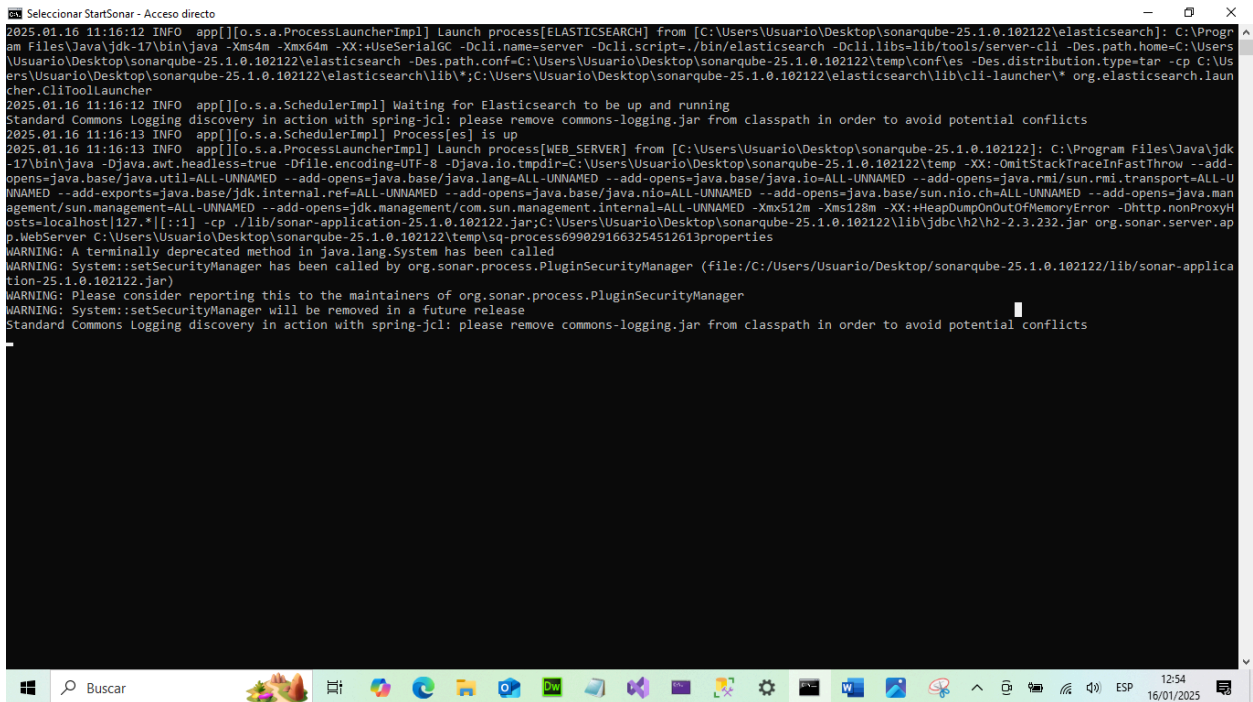
Una vez creado creamos nuestra propia clase dentro del proyecto de Test



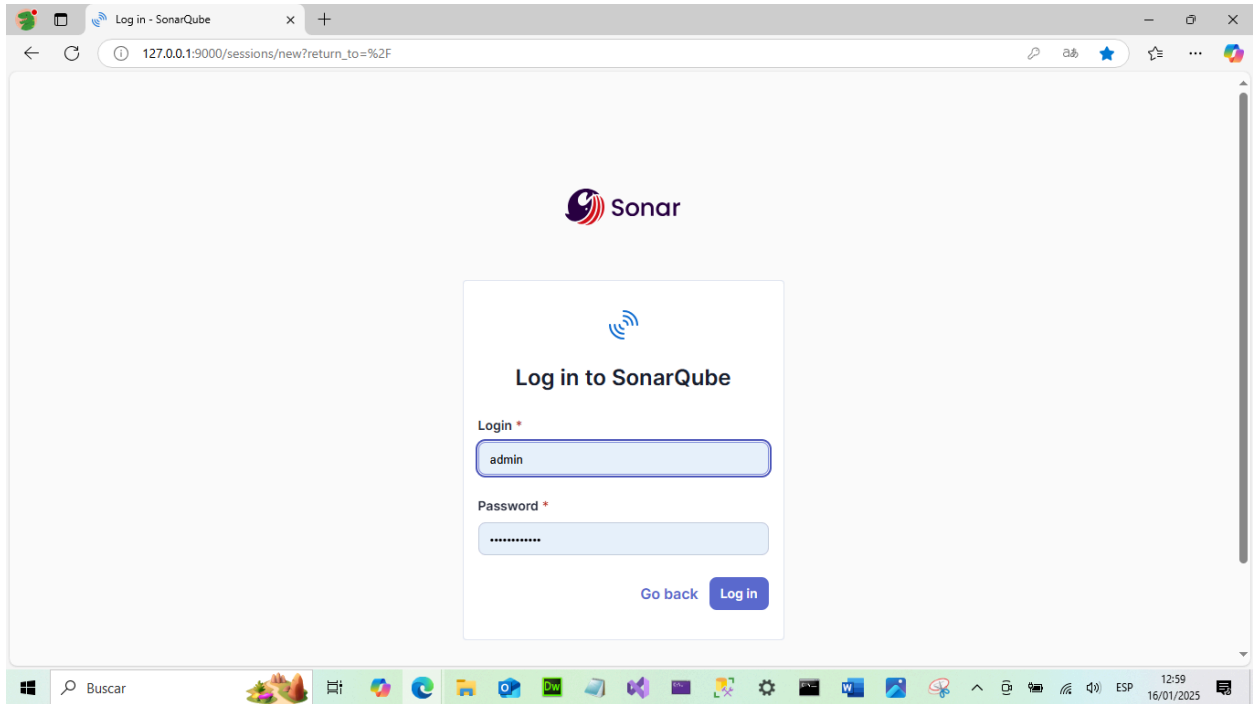
Esto nos permite hacer el código correspondiente para que se pueda conectar con SONARQUBE



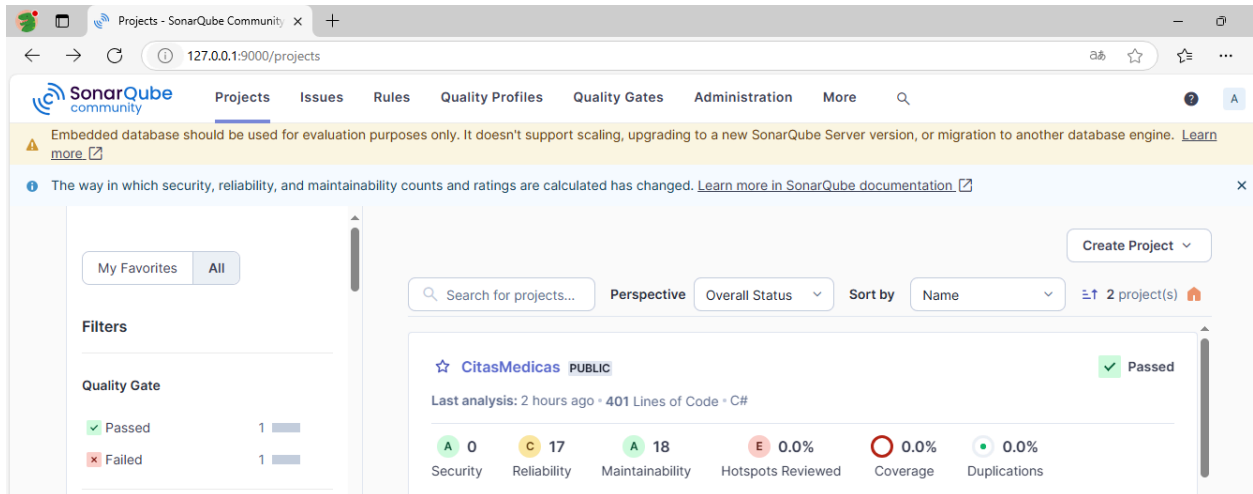
Lo creamos procedemos a correr el cmd para que se pueda abrir Sonarqube



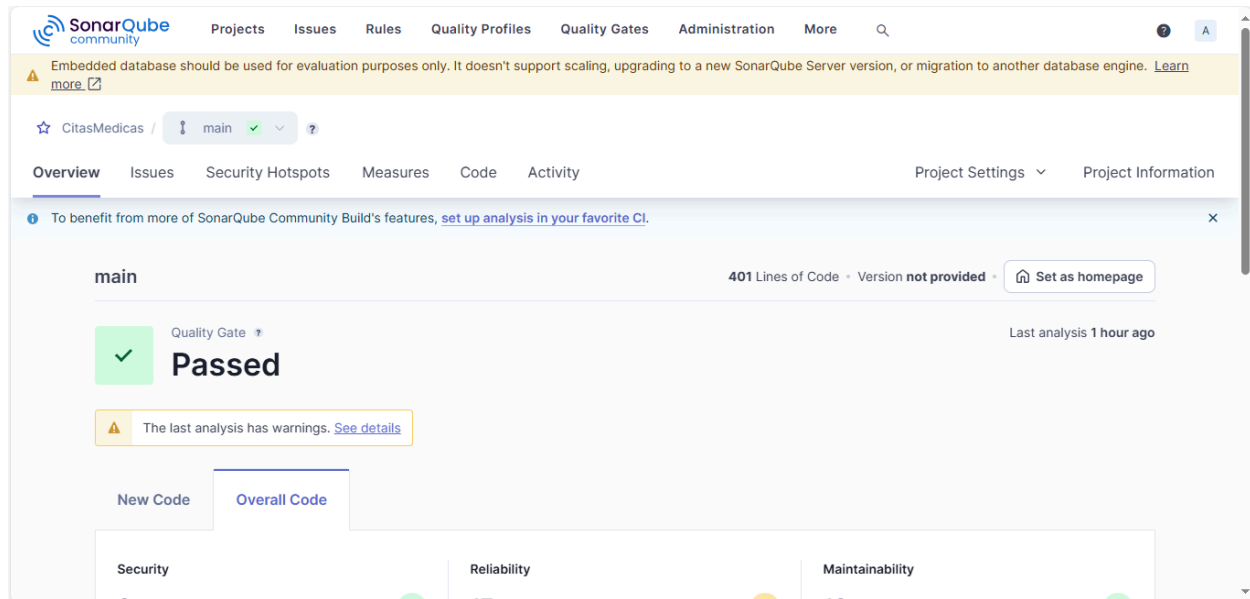
Una vez abierto nos vamos a la página de SonarQube



Iniciamos Sesión nos permite ver todas las pruebas que hemos hecho



Para poder hacer la prueba creamos el proyecto y siempre tenemos que tener los tokens a la mano porque nos permite hacer las pruebas



Conclusión

En conclusion, en esta prueba pudimos volver a recrear un proyecto que habíamos hecho anteriormente solo que si tuvimos nuestras dificultades para poder hacerlo nuevamente.

Teníamos que estar atentos a las variables que habíamos puesto en la base de datos, ya que esto nos dio muchas complicaciones para hacer funcionar el proyecto que en este caso era de Medicos.

Tuvimos que tomar en cuenta como son las citas medicas y que se necesita para que se pueda llevar acabo una cita medica y que es lo que usan o hacen para poder agendar o registrar la cita.

