



TI-Capital Humano

Desarrollador .Net

Referencia Rápida

Web API



ASP .NET – Web API.

Tabla de contenido

1	Objetivo.....	3
2	Creación de una Web API.....	4
2.1	Crear Proyecto ASP .NET Core Web API.....	4
2.2	Controlador de Ejemplo	6
2.3	Agregar EntityFramework en ASP .Net Core	6
2.3.1	Instalar paquetes NuGet.....	6
2.3.2	Creación de un modelo con Entity Framework Core	7
2.3.3	Configurar la cadena de conexión	9
2.3.4	Configurar servicio del DbContext.....	11
2.4	Agregar Controladores al Web API	12
2.5	Probar el Controlador.....	14
3	Postman.....	17
3.1.1	Agrupación de requerimientos en colecciones	17
3.1.2	Crear una colección.....	17
3.1.3	Agregar carpetas a una colección	17
3.1.4	Agregar Requerimiento.....	18
3.1.5	Requerimiento tipo Get (Consultar)	19
3.1.6	Requerimiento tipo Post (Agregar)	20
3.1.7	Requerimiento tipo Put (Actualizar)	22
3.1.8	Requerimiento tipo Delete (Eliminar)	23
4	Consumir Web API.....	25
4.1	Consumir una Web API con HttpClient	25
4.1.1	Crear un Proyecto de tipo Aplicación Web ASP .NET (.NET Framework).	25
4.1.2	Crear una clase tipo POCO	25
4.1.3	Crear la clase de Negocio	25
	Crear la clase NEstados con los siguientes métodos, dentro de la Carpeta Models	25
4.1.4	Crear el controlador Estados	31
4.1.5	Implementar los métodos de acción del controlador Estados.....	32
4.1.6	Crear las vistas correspondientes a cada uno de los métodos de acción	35

4.2	Consumir una Web API con AJAX	35
4.2.1	Habilitar CORS con política con nombre y middleware	35
4.2.2	Archivo de Configuración appsettings.json.....	37
4.2.3	Consumir el Web API desde el cliente de la aplicación	37

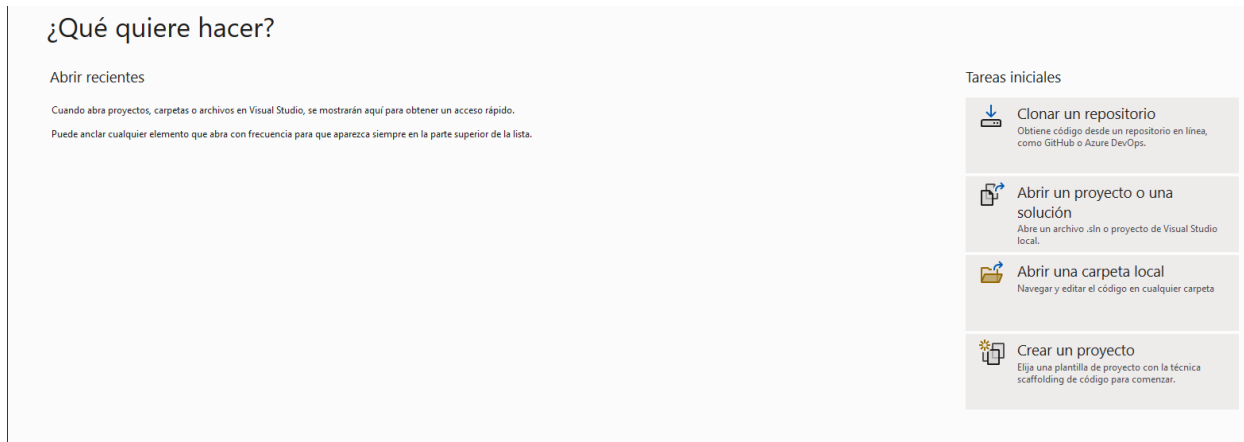
1 Objetivo

En este módulo los participantes aprenderán los conceptos generales necesarios de .Net Core y de Web API, las técnicas y uso de herramientas para el desarrollo de una aplicación Web API con .Net Core y con métodos CRUD; asimismo, probar la Web API con la aplicación Postman, y finalmente consumir la Web API desde el lado del Servidor con HttpClient y desde el lado del Cliente con Ajax.

2 Creación de una Web API

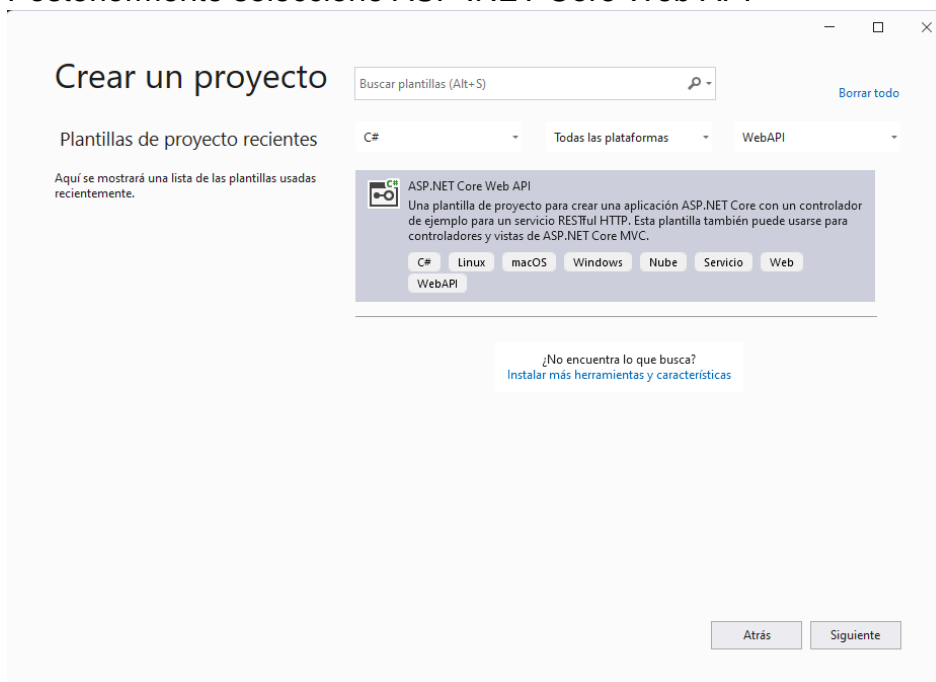
2.1 Crear Proyecto ASP .NET Core Web API

En la ventana de inicio, seleccione Crear un proyecto.



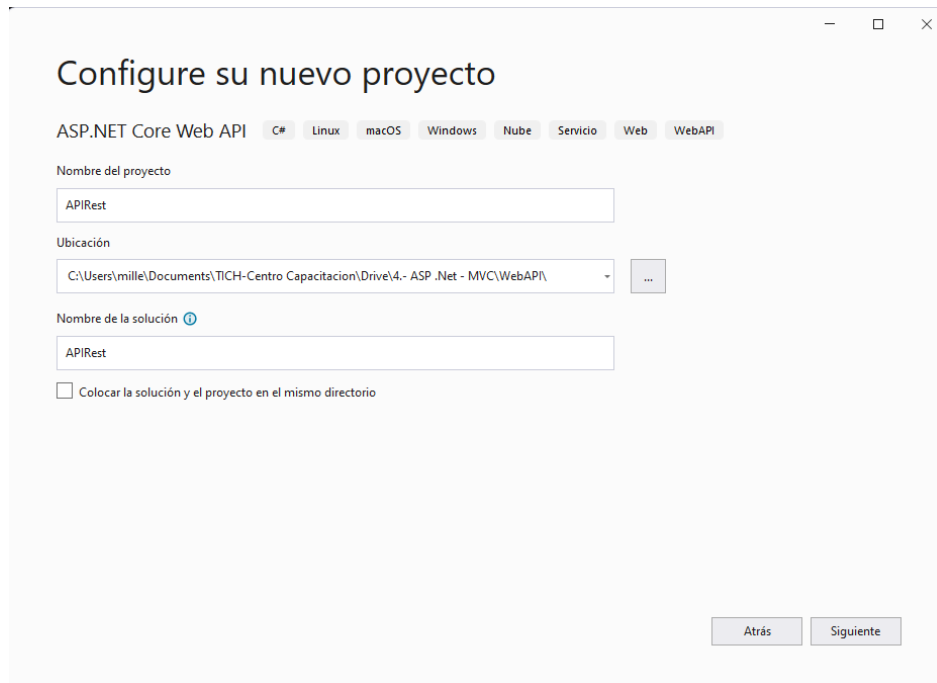
En la ventana Crear un nuevo proyecto

Seleccione en el filtro Lenguaje C#, Todas las Plataformas, Web API
Posteriormente seleccione ASP .NET Core Web API



En la ventana Configurar el nuevo proyecto

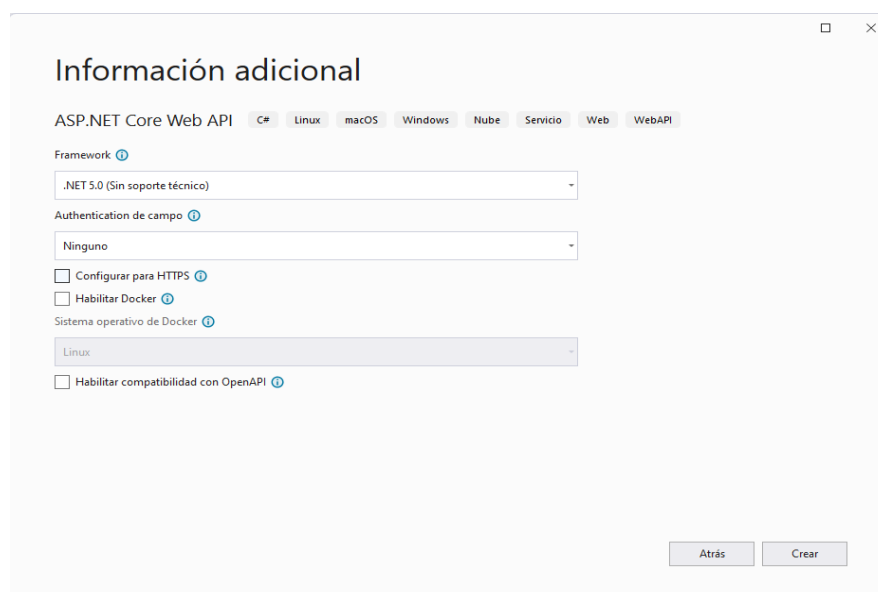
Escriba **APIRest** en el campo Nombre del proyecto, la ubicación donde se almacenará y en el nombre de la solución repetir el nombre del Proyecto y oprimir siguiente



The screenshot shows the 'Configure your new project' dialog box. At the top, there are tabs for different project types: ASP.NET Core Web API, C#, Linux, macOS, Windows, Nube, Servicio, Web, and WebAPI. The 'ASP.NET Core Web API' tab is selected. Below the tabs, there are three input fields: 'Nombre del proyecto' (Project name) with the value 'APIRest', 'Ubicación' (Location) with the value 'C:\Users\mille\Documents\TICH-Centro Capacitacion\Drive4.- ASP .Net - MVC\WebAPI\' and a browse button (...), and 'Nombre de la solución' (Solution name) with the value 'APIRest'. There is a checkbox labeled 'Colocar la solución y el proyecto en el mismo directorio' (Place the solution and the project in the same directory) which is currently unchecked. At the bottom right, there are two buttons: 'Atrás' (Back) and 'Siguiente' (Next).

En la ventana Información Adicional

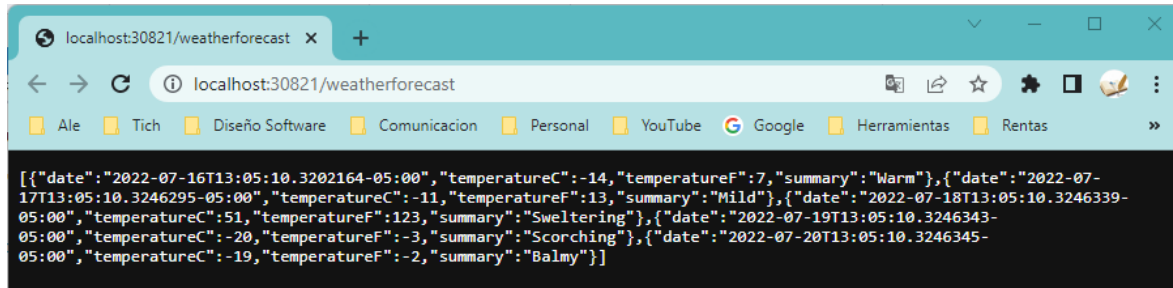
En el framework seleccione **.NET 5.0**; en Autenticación de Campo **Ninguno**, **Desactive las casillas** Configurar HTTPS, Habilitar Docker, Habilitar compatibilidad con OpenAPI



The screenshot shows the 'Additional information' dialog box. At the top, there are tabs for different project types: ASP.NET Core Web API, C#, Linux, macOS, Windows, Nube, Servicio, Web, and WebAPI. The 'ASP.NET Core Web API' tab is selected. Below the tabs, there are several settings: 'Framework' (Framework) with a dropdown menu showing '.NET 5.0 (Sin soporte técnico)'; 'Authentication de campo' (Field authentication) with a dropdown menu showing 'Ninguno'; a checkbox for 'Configurar para HTTPS' (Configure for HTTPS) which is unchecked; a checkbox for 'Habilitar Docker' (Enable Docker) which is unchecked; 'Sistema operativo de Docker' (Docker operating system) with a dropdown menu showing 'Linux'; and a checkbox for 'Habilitar compatibilidad con OpenAPI' (Enable OpenAPI compatibility) which is unchecked. At the bottom right, there are two buttons: 'Atrás' (Back) and 'Crear' (Create).

2.2 Controlador de Ejemplo

La plantilla de Web API crea un controlador de ejemplo llamado `WeatherForecastController`, el cual contiene el método `HttpGet`, mismo que regresa un arreglo en formato json con cinco temperaturas al azar en grados centígrados, grados Fahrenheit, y un indicador. Para efectos del ejemplo utiliza una clase tipo POCO llamada `WeatherForecast` con cuatro propiedades, Fecha y hora, temperatura en grados centígrados, temperatura en grados fahrenheit y un indicador, como se puede ver en la siguiente figura



Se puede observar en la línea de direcciones del navegador como se está invocando al controlador `weathercast`; aunque no se le indica a que acción debe ingresar por default entra a la acción `Get`, que es la que nos regresa los 5 objetos en formato json que se muestra en la figura.

2.3 Agregar EntityFramework en ASP .Net Core

Entity Framework Core (EF Core) se distribuye como paquetes NuGet.

2.3.1 Instalar paquetes NuGet

[Microsoft.EntityFrameworkCore](#)

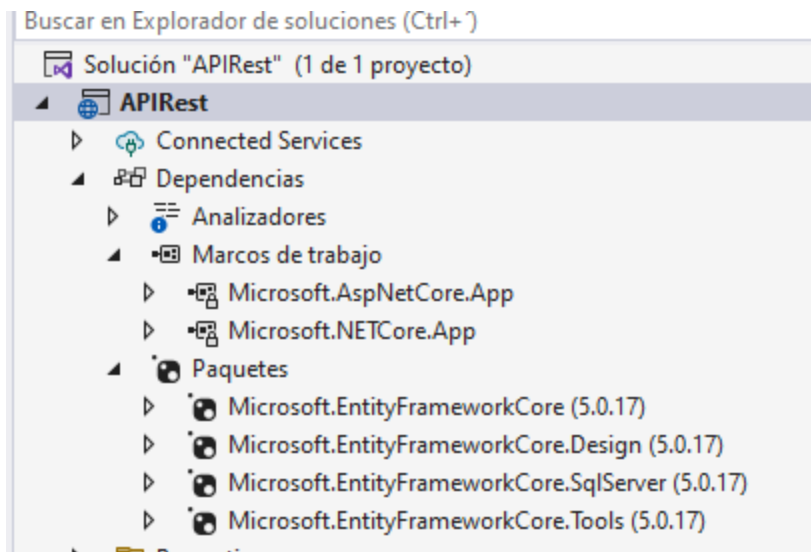
[Microsoft.EntityFrameworkCore.Design](#)

[Microsoft.EntityFrameworkCore.SqlServer](#)

[Microsoft.EntityFrameworkCore.Tools](#)

De preferencia la misma versión del web api que se está trabajando. En este caso seleccionar las versiones 5.0.17

Una vez instalados los paquetes se puede comprobar en la rama de Dependencias



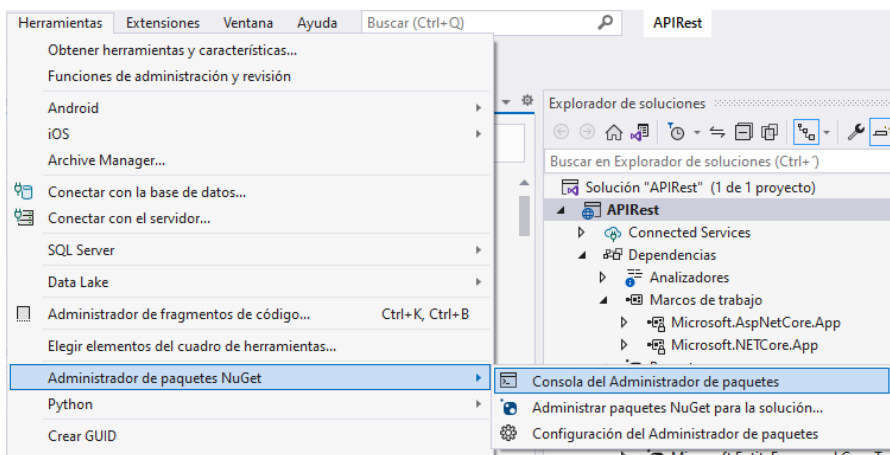
2.3.2 Creación de un modelo con Entity Framework Core

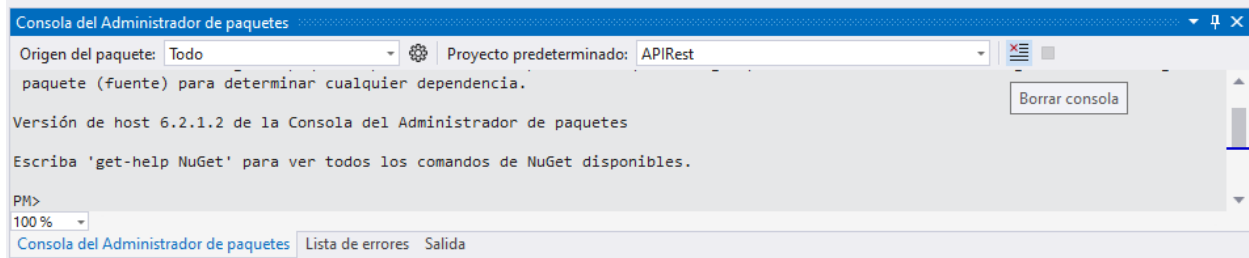
Crear un modelo para una base de datos existente con Entity Framework Core

EF Core no es compatible con el diseñador visual para el modelo de base de datos y el asistente para crear las clases de entidad y contexto similares a EF 6. Por lo tanto, debemos realizar ingeniería inversa con el comando **Scaffold-DbContext**. Este comando de ingeniería inversa crea clases de entidad y contexto (derivando DbContext) según el esquema de la base de datos existente.

Crear clases de entidad “**Estados**” y “**Alumnos**” y contexto desde la base de datos **InstitutoTich**

En Visual Studio, seleccione el menú Herramientas -> Administrador de paquetes NuGet -> Consola del administrador de paquetes





Borre la consola y ejecute el siguiente comando:

```
Scaffold-DbContext "Server=MSI;Database=InstitutoTich;User ID=sa;password=Pass2017;"  
Microsoft.EntityFrameworkCore.SqlServer -Tables "Estados", "Alumnos" -Context ApiContext -ContextDir  
Models/Context -OutputDir Models/Entities -NoPluralize -Force
```

En el comando anterior, el primer parámetro es una cadena de conexión que incluye tres partes: Servidor DB, nombre de la base de datos e información de seguridad.

Servidor=MSI; se refiere al servidor de base de datos desde donde se extraerá la información.

Database=InstitutoTich; especifica el nombre de la base de datos "InstitutoTich" para la que vamos a crear clases.

User Id=sa especifica el usuario para conectarse al SQL Server

Password=Pass2017 especifica la contraseña para conectarse al SQL Server

El segundo parámetro es el nombre del proveedor. Usamos un proveedor para SQL Server, por lo que es Microsoft.EntityFrameworkCore.SqlServer.

El parámetro **-Tables "Estados", "Alumnos"**, es un arreglo de las tablas para generar tipos de entidad, en este caso Estados y Alumnos. Si se omite este parámetro, se incluyen todas las tablas.

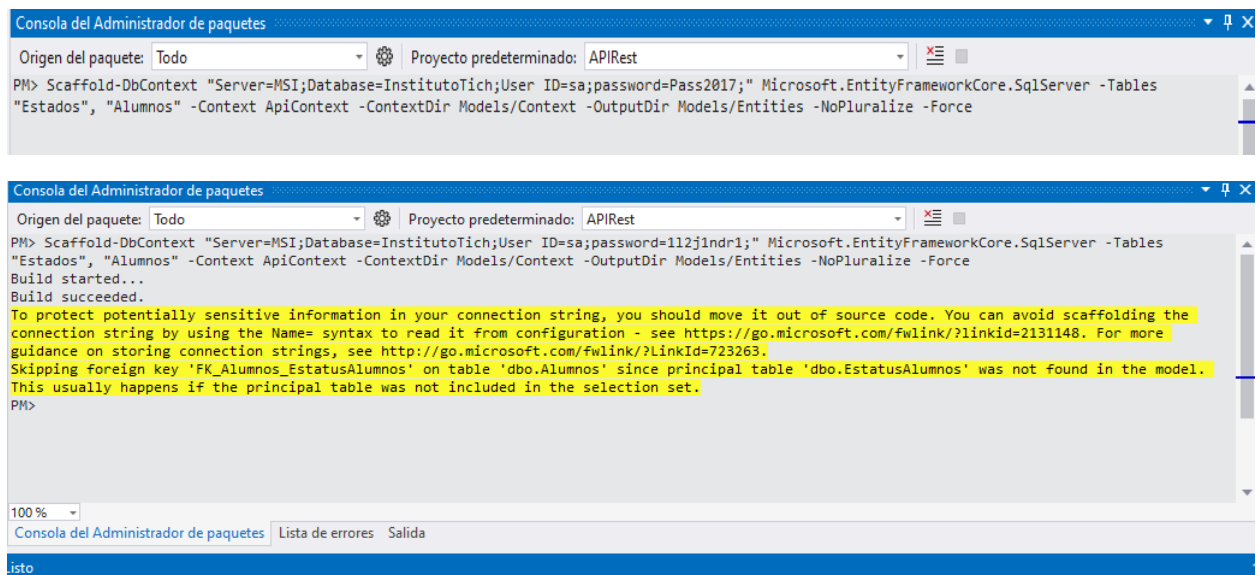
El parámetro **-Context ApiContext** especifica el nombre que se le dará a la clase de contexto

El parámetro **-ContextDir Models/Context** indica el directorio para colocar el archivo DbContext. Las rutas son relativas al directorio del proyecto.

El parámetro **-OutputDir Models/Entities** especifica el directorio donde queremos generar todas las clases, que es la carpeta Models/Entities en este caso.

El parámetro **-NoPluralize** indica que no use el pluralizador, en caso de que no se indica las clases de entidad serán generadas con nombres en singular

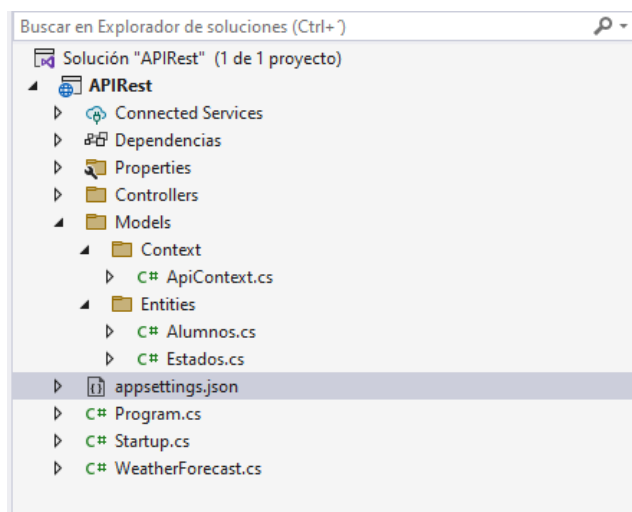
El parámetro **-Force** especifica que sobre escriba los archivos existentes



```
Consola del Administrador de paquetes
Origen del paquete: Todo Proyecto predeterminado: APIRest
PM> Scaffold-DbContext "Server=MSI;Database=InstitutoTich;User ID=sa;password=Pass2017;" Microsoft.EntityFrameworkCore.SqlServer -Tables
"Estados", "Alumnos" -Context ApiContext -ContextDir Models/Context -OutputDir Models/Entities -NoPluralize -Force

Consola del Administrador de paquetes
Origen del paquete: Todo Proyecto predeterminado: APIRest
PM> Scaffold-DbContext "Server=MSI;Database=InstitutoTich;User ID=sa;password=112j1ndr1;" Microsoft.EntityFrameworkCore.SqlServer -Tables
"Estados", "Alumnos" -Context ApiContext -ContextDir Models/Context -OutputDir Models/Entities -NoPluralize -Force
Build started...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the
connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more
guidance on storing connection strings, see http://go.microsoft.com/fwlink/?linkid=723263.
Skipping foreign key 'FK_Alumnos_EstatusAlumnos' on table 'dbo.Alumnos' since principal table 'dbo.EstatusAlumnos' was not found in the model.
This usually happens if the principal table was not included in the selection set.
PM>
```

Al terminar de ejecutarse el comando anterior veremos que nos creó los siguientes elementos en nuestra solución:



La clase **ApiContext** es nuestra clase de contexto de EntityFramework

Y las clases Estados y Alumnos son nuestras clases de entidad que se crearon con el scaffolding

2.3.3 Configurar la cadena de conexión

Configurar la cadena de conexión en el archivo appsetting.json

Con la finalidad de no tener la cadena de conexión dentro del código, se agregará ésta en el archivo de configuración appsettings.json, y posteriormente indicaremos que dicha cadena de conexión se tome desde ahí.

Primeramente agregar la cadena de conexión en el archivo appsettings.json

```
"ConnectionStrings": {
  "InstitutoTich": "Server=MSI;Database=InstitutoTich;User ID=sa;password=Pass2017;"
},
```

```
{
  "ConnectionStrings": {
    "InstitutoTich": "Server=MSI;Database=InstitutoTich;User ID=sa;password=Pass2017;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Recordando el formato json, donde siempre van en parejas llave-valor. El primer dato "InstitutoTich" es el nombre que le estamos dando a nuestra cadena de conexión. Tomando en consideración que podemos tener más de una cadena de conexión.

Continuaremos quitando en el archivo de contexto las líneas que se crearon cuando se ejecutó el scaffolding y se proporcionó la cadena de conexión.

```
public partial class ApiContext : DbContext
{
    // Referencias
    public ApiContext()
    {
    }

    // Referencias
    public ApiContext(DbContextOptions<ApiContext> options)
        : base(options)
    {
    }

    // Referencias
    public virtual DbSet<Alumnos> Alumnos { get; set; }
    // Referencias
    public virtual DbSet<Estados> Estados { get; set; }

    // Referencias
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            To protect potentially sensitive information in your connection string, you should move it out of source code.
            optionsBuilder.UseSqlServer("Server=MSI;Database=InstitutoTich;User ID=sa;password=Pass2017;");
        }
    }
}
```



Eliminar esta línea

2.3.4 Configurar servicio del DbContext

Configurar servicio del DbContext en el archivo Startup.cs

El servicio DbContext se encarga de la tarea de conectarse a la base de datos y asignar objetos a los registros de la base de datos. El contexto de la base de datos se registra en la Configuración de Servicios (ConfigureServices) en Startup.cs:

```
services.AddDbContext<ApiContext>(options =>
{
    options.UseSqlServer(Configuration.GetConnectionString("InstitutoTich"));
});
//
services.AddControllers();

public class Startup
{
    0 referencias
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    2 referencias
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    0 referencias
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<ApiContext>(options =>
        {
            options.UseSqlServer(Configuration.GetConnectionString("InstitutoTich"));
        });
        services.AddControllers();
    }
}
```

Como podemos observar el objeto services es una colección de servicios y es ahí donde agregaremos el servicio del DbContext, el parámetro <ApiContext> es el nombre de nuestra clase de contexto. Como parámetro enviaremos una función donde se le indica que usará SQLServer y como parámetro se le enviará la cadena de conexión, misma que se obtendrá a través del método GetConnectionString del objeto Configuration, que es el objeto que nos maneja el archivo de configuración appsettings.json, similar al ConfigurationManager de otras versiones de .Net. Al método GetConnectionString le pasamos como parámetro el nombre que le dimos en el archivo appsetting.json a nuestra cadena de conexión.

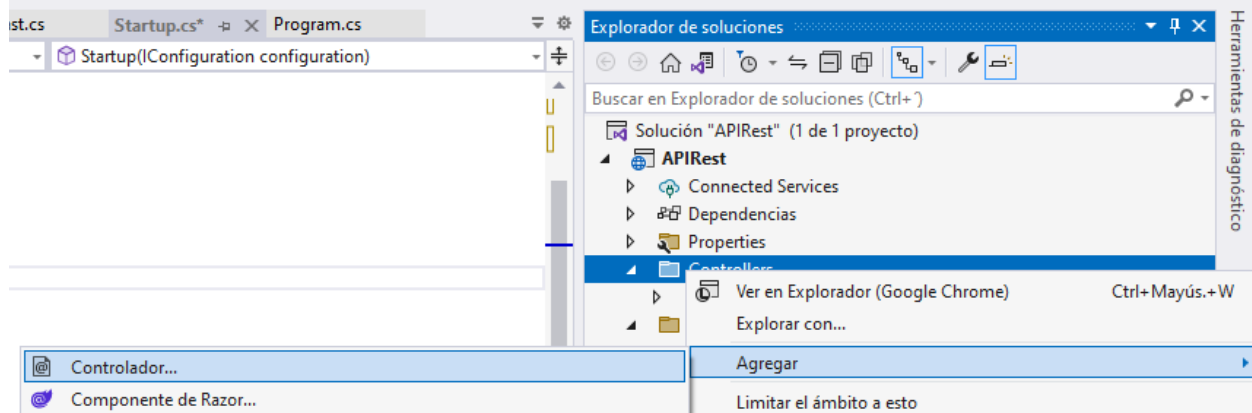
Ya que en esta clase Startup accederemos a la clase ApiContext y UseSqlServer, previamente debemos agregar los correspondientes using de los namespace a los que pertenecen dichas clases:

```
using APIRest.Models.Context;
```

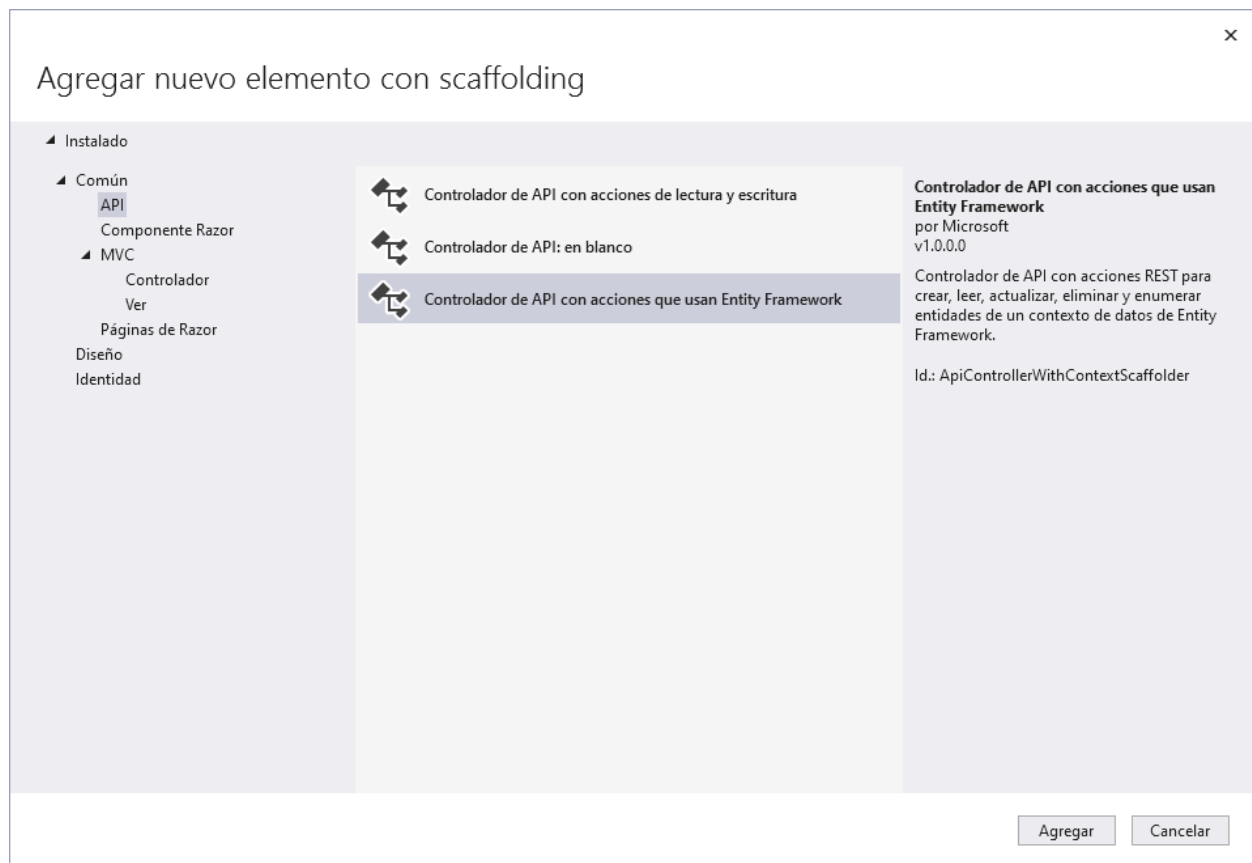
`using Microsoft.EntityFrameworkCore;`

2.4 Agregar Controladores al Web API

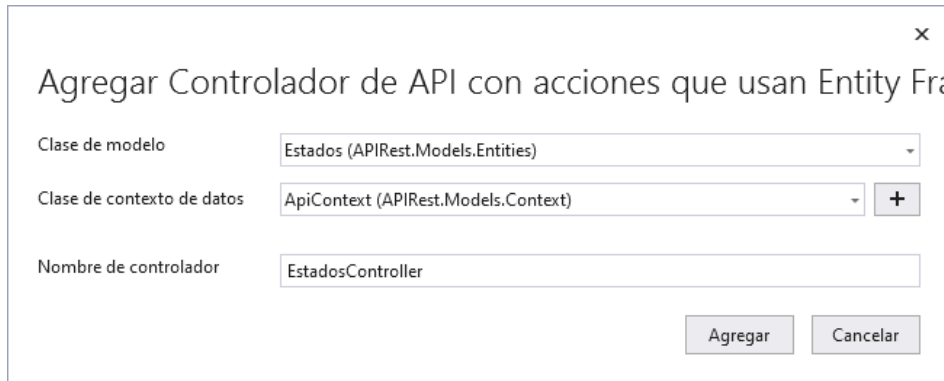
En la carpeta Controllers dar click derecho y seleccionar Agregar/Controlador



En la ventana Agregar un nuevo elemento con scaffolding seleccionar API y Controlador de API con acciones que usan EntityFramework



En la ventana Agregar Controlador de API con acciones que usan Entity Framework Core, en la Clase de Modelo seleccionar “Estados”, en la clase de contexto de datos seleccionar nuestra clase de contexto en este caso “APIContext”, dejamos el nombre que nos da por default del controlador, y oprimimos Agregar



Agregar Controlador de API con acciones que usan Entity Framework Core

Clase de modelo: Estados (APIRest.Models.Entities)

Clase de contexto de datos: ApiContext (APIRest.Models.Context) +

Nombre de controlador: EstadosController

Agregar Cancelar

Con esto se nos crea el controlador con todos los métodos necesarios para realizar un CRUD, en este caso de la tabla de Estados

```

namespace APIRest.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 referencia
    public class EstadosController : ControllerBase
    {
        private readonly ApiContext _context;

        0 referencias
        public EstadosController(ApiContext context)
        {
            _context = context;
        }

        // GET: api/Estados
        [HttpGet]
        0 referencias
        public async Task<ActionResult<IEnumerable<Estados>>> GetEstados()...

        // GET: api/Estados/5
        [HttpGet("{id}")]
        0 referencias
        public async Task<ActionResult<Estados>> GetEstados(int id)...

        // PUT: api/Estados/5
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        0 referencias
        public async Task<IAActionResult> PutEstados(int id, Estados estados)...

        // POST: api/Estados
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        0 referencias
        public async Task<ActionResult<Estados>> PostEstados(Estados estados)...

        // DELETE: api/Estados/5
        [HttpDelete("{id}")]
        0 referencias
        public async Task<IAActionResult> DeleteEstados(int id)...

        2 referencias
        private bool EstadosExists(int id)...
    }
}

```

Como podemos observar en la clase de arriba, los métodos se encuentran con atributos que indican el verbo de http al que corresponde:

HttpGet – Para consultas

HttpPut – Para Actualización

HttpPost – Para Agregar

HttpDelete – Para Eliminar

2.5 Probar el Controlador

Para probar el controlador, vamos a ejecutarlo, y desde la línea de direcciones del navegador, indicamos la ruta con el cual se configuro en la clase del controlador

```
namespace APIRest.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 referencia
    public class EstadosController : ControllerBase
    {
        . . . . .
    }
}
```

Podemos observar el atributo `[Route("api/[controller]")]`

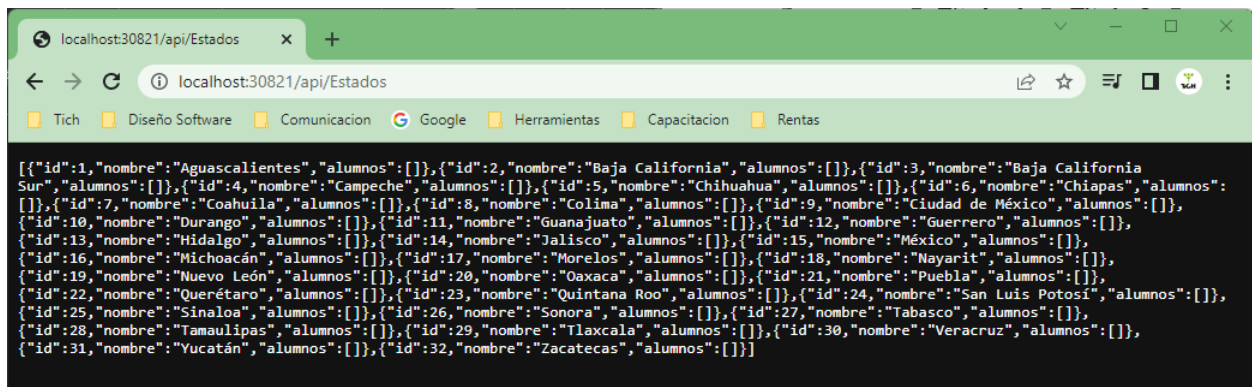
Indicando que la ruta del controlador inicia con api. Misma que se puede eliminar o bien cambiar.

Por lo que dejando el Route tal y como se generó, en la línea de direcciones del navegador deberemos teclear la siguiente dirección:

<http://localhost:30821/api/Estados>

Al indicar solamente el nombre del Controlador, sin ninguna acción, se ejecuta el método por default que es el método Get. Con esto ingresará al Controlador Estados, ejecutará el método constructor donde crear una instancia de la clase de contexto de Entity Framework. Posteriormente ingresa al método invocado desde el navegador y ejecuta la consulta de todos los Estados desde la base de datos a través de la clase de contexto de Entity Framework.

Dando el siguiente resultado:



```
[{"id":1,"nombre":"Aguascalientes","alumnos":[]}, {"id":2,"nombre":"Baja California","alumnos":[]}, {"id":3,"nombre":"Baja California Sur","alumnos":[]}, {"id":4,"nombre":"Campeche","alumnos":[]}, {"id":5,"nombre":"Chihuahua","alumnos":[]}, {"id":6,"nombre":"Chiapas","alumnos":[]}, {"id":7,"nombre":"Coahuila","alumnos":[]}, {"id":8,"nombre":"Colima","alumnos":[]}, {"id":9,"nombre":"Ciudad de México","alumnos":[]}, {"id":10,"nombre":"Durango","alumnos":[]}, {"id":11,"nombre":"Guanajuato","alumnos":[]}, {"id":12,"nombre":"Guerrero","alumnos":[]}, {"id":13,"nombre":"Hidalgo","alumnos":[]}, {"id":14,"nombre":"Jalisco","alumnos":[]}, {"id":15,"nombre":"México","alumnos":[]}, {"id":16,"nombre":"Michoacán","alumnos":[]}, {"id":17,"nombre":"Morelos","alumnos":[]}, {"id":18,"nombre":"Nayarit","alumnos":[]}, {"id":19,"nombre":"Nuevo León","alumnos":[]}, {"id":20,"nombre":"Oaxaca","alumnos":[]}, {"id":21,"nombre":"Puebla","alumnos":[]}, {"id":22,"nombre":"Querétaro","alumnos":[]}, {"id":23,"nombre":"Quintana Roo","alumnos":[]}, {"id":24,"nombre":"San Luis Potosí","alumnos":[]}, {"id":25,"nombre":"Sinaloa","alumnos":[]}, {"id":26,"nombre":"Sonora","alumnos":[]}, {"id":27,"nombre":"Tabasco","alumnos":[]}, {"id":28,"nombre":"Tamaulipas","alumnos":[]}, {"id":29,"nombre":"Tlaxcala","alumnos":[]}, {"id":30,"nombre":"Veracruz","alumnos":[]}, {"id":31,"nombre":"Yucatán","alumnos":[]}, {"id":32,"nombre":"Zacatecas","alumnos":[]}]
```

Donde podemos observar, que se encuentran todos los Estados, en formato json, extraídos de la base de datos

`GetEstados(int id)`

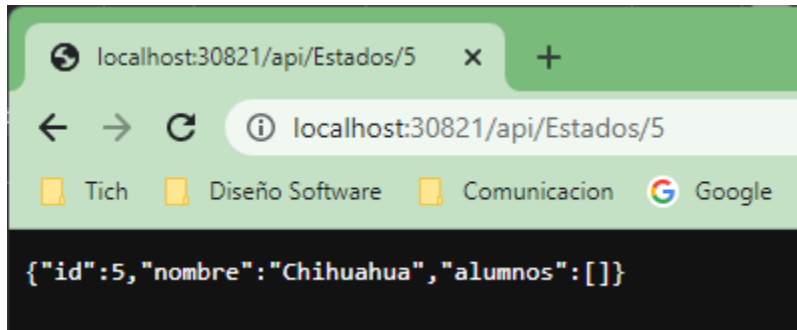
Para probar este método ingresaremos la siguiente dirección en el navegador

<http://localhost:30821/api/Estados/5>

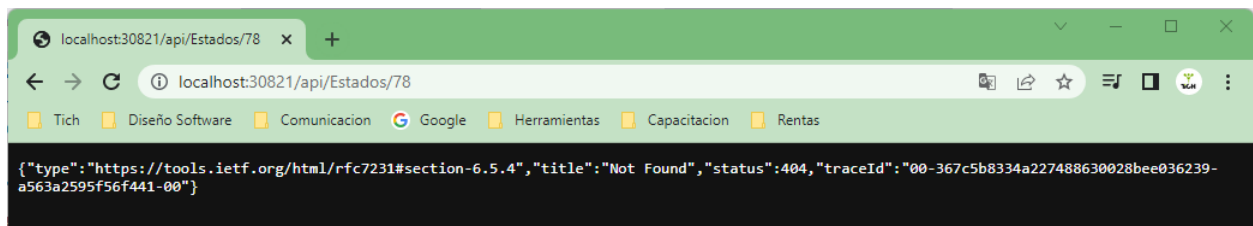
Como podemos observar en este caso se le agrega /5 para indicar el id del estado que queremos consultar. Con esto ingresará al Controlador Estados, ejecutará el método

constructor donde crear una instancia de la clase de contexto de Entity Framework. Posteriormente ingresa al método GetEstados(int id) invocado desde el navegador y ejecuta la consulta del Estados cuyo id se proporciona como parámetro, desde la base de datos a través de la clase de contexto de Entity Framework.

Dando el siguiente resultado:



En caso de que el id no existirá el resultado sería el siguiente:



Podemos observar que dentro del json que regresa se encuentra el código 404 correspondiente al NOTFound

3 Postman

Postman es una plataforma API para construir y usar API. Postman simplifica cada paso del ciclo de vida de la API y agiliza la colaboración para que pueda crear mejores API, más rápido.

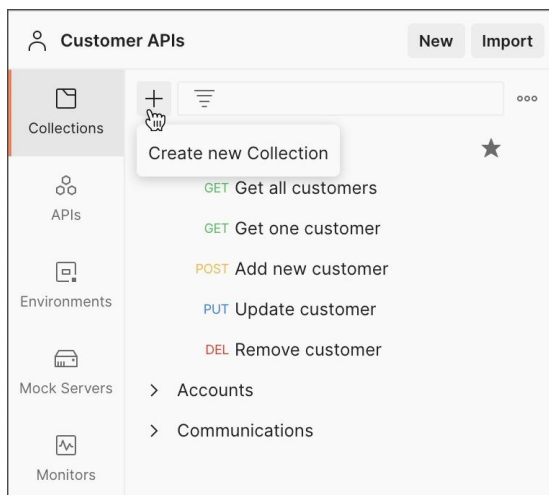
3.1.1 Agrupación de requerimientos en colecciones

Puede agrupar sus solicitudes y ejemplos de Postman en colecciones para mantener su espacio de trabajo organizado, colaborar con compañeros de equipo, generar documentación de API y pruebas de API, y automatizar la ejecución de solicitudes.

3.1.2 Crear una colección

Hay varias formas de crear una nueva colección:

Seleccione Colecciones en la barra lateral, luego seleccione +.



Seleccione Nuevo, luego seleccione Colección, y finalmente escriba el nombre de la Colección, en este caso **DesarrolladorNet**

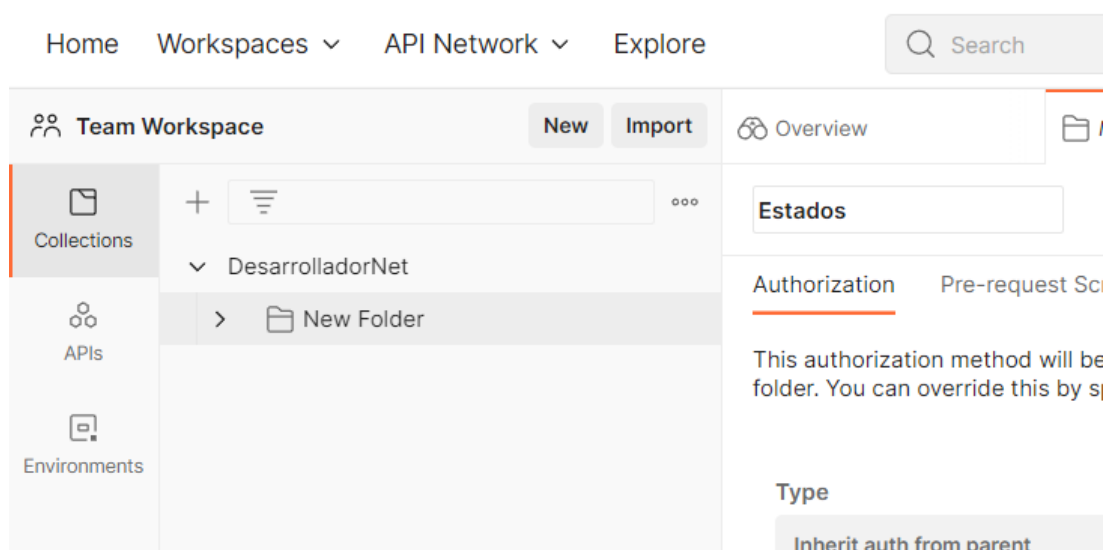
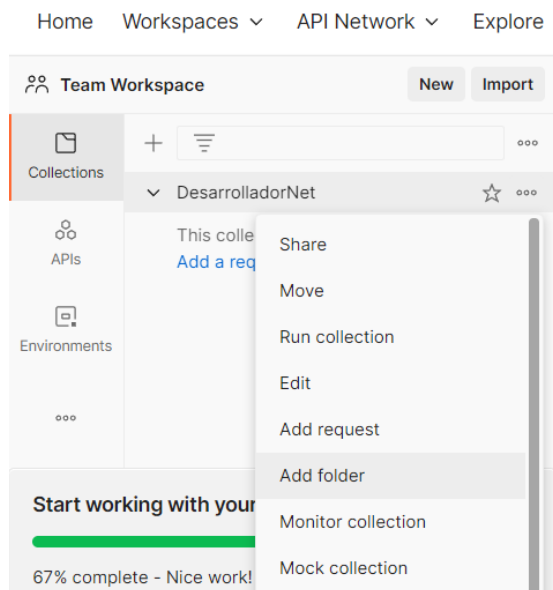
3.1.3 Agregar carpetas a una colección

Para organizar todavía de mejor manera sus requerimientos, se pueden crear carpetas y subcarpetas dentro de una colección.

Para agregar una carpeta a su colección:

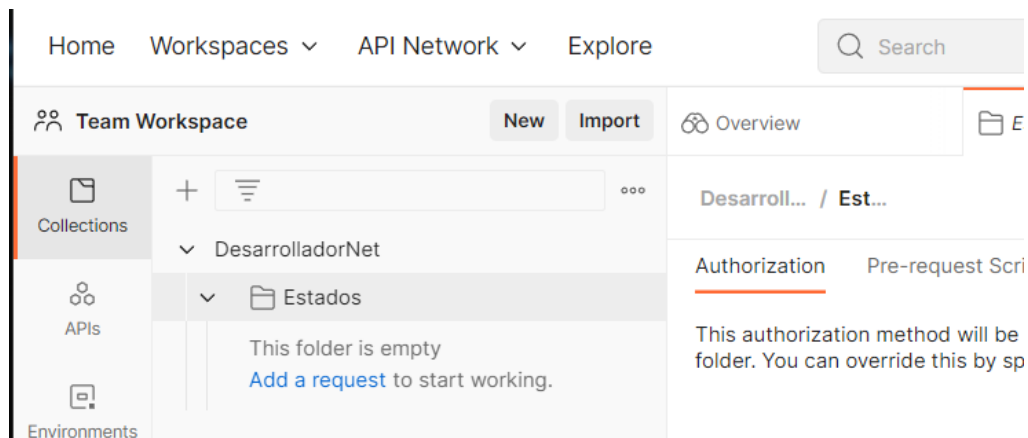
Seleccione el ícono de más acciones a la derecha del nombre de la colección.

Seleccione Agregar carpeta, finalmente escriba el nombre de la carpeta **Estados**

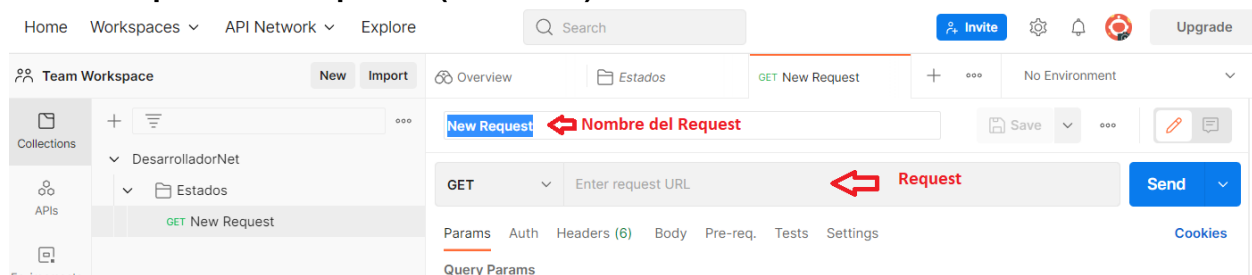


3.1.4 Agregar Requerimiento

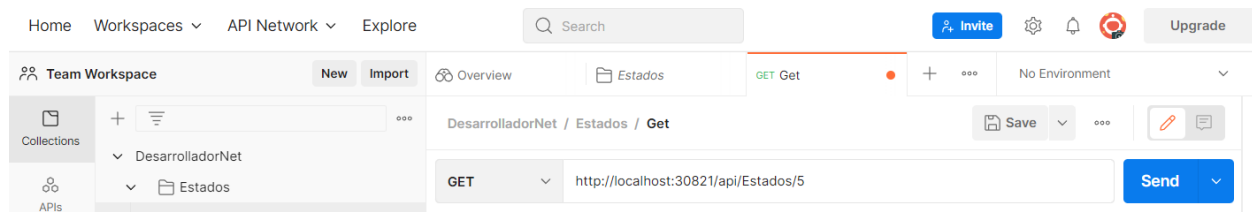
Seleccione Add request, que se encuentra bajo la carpeta, en este caso Estados, bien seleccione más opciones al lado derecho de la carpeta para finalmente seleccionar Add Folder



3.1.5 Requerimiento tipo Get (Consultar)



Le damos el nombre de Get, y en la URL le damos la url que daríamos en el navegador <http://localhost:30821/api/Estados/5>



Oprimos Send, para que haga la llamada y podamos observar que ingresa a nuestra Web API. Si ponemos puntos de interrupción en nuestra web api, podemos depurarla

DesarrolladorNet / Estados / Get

GET http://localhost:30821/api/Estados/5 Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body

200 OK 9.23 s 224 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 5,
3   "nombre": "Chihuahua",
4   "alumnos": []
5 }
```

← Respuesta

↑ Código de Retorno

↑ Tiempo de ejecución

↑ Tamaño de la respuesta

3.1.6 Requerimiento tipo Post (Agregar)

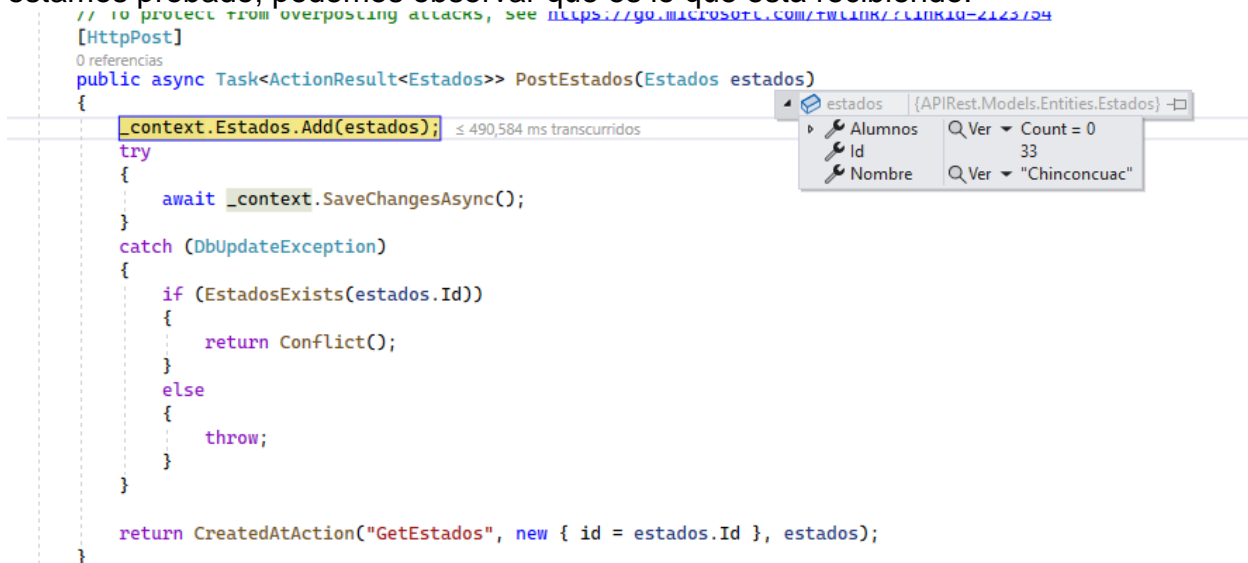
Para poder probar el método de agregar, necesitaremos utilizar un verbo POST desde el postman. Para ello agregaremos otro requerimiento al cual le llamaremos Agregar, con el verbo POST, y dentro del cuerpo de la petición enviaremos en formato Json el objeto que queremos agregar. Considerando que el objeto lo agregaremos en una base de datos y si la tabla se definió la llave primaria como incremental, no deberemos enviar ese dato.

Las solicitudes HTTP para verbos como POST, PUT y PATCH opcionalmente envían un payload, un conjunto de datos necesarios para el requerimiento, como parte de la solicitud. Estos datos pueden tener el formato de pares clave-valor o algún otro formato de serialización, como JSON y XML.

En nuestro caso, la url será la que corresponde al controlador, considerando que en una web API, el método se seleccionará con base en el verbo http con que se hace el requerimiento.



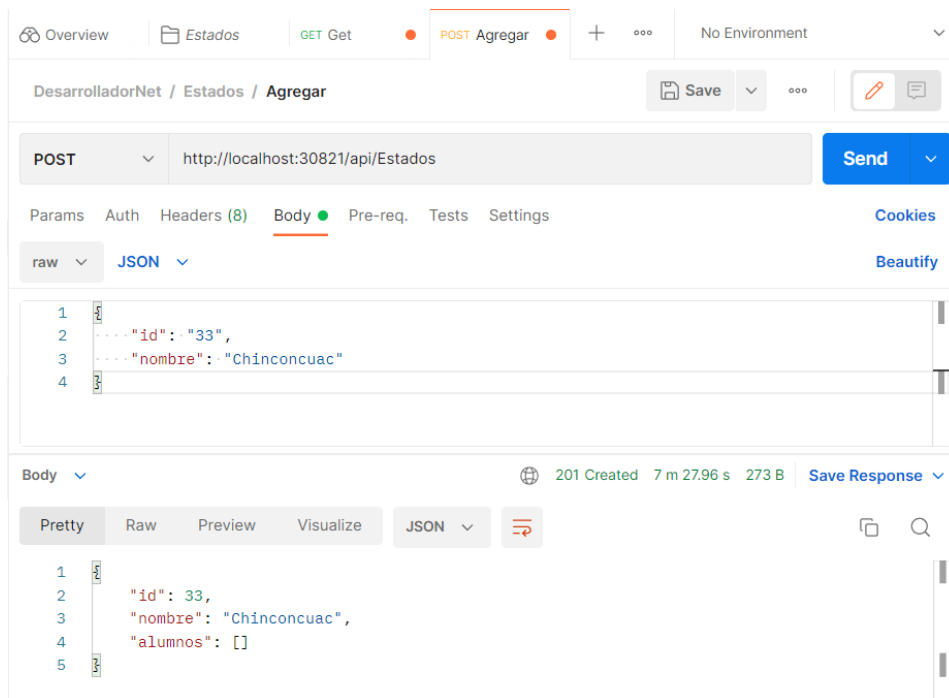
Al oprimir Send y habiendo colocado un punto de interrupción en el método que estamos probado, podemos observar que es lo que está recibiendo.



Como podemos observar está recibiendo los datos que se enviaron en el body desde la solicitud http realizada desde postman.

Al final del método observamos que se está invocando al método CreatedAtAction, el cual realizará un llamado al método cuyo nombre que se pasa como primer parámetro, y el segundo parámetro son los datos enviados a dicho método.

En este caso se está invocando al método GetEstados, como parámetro se le está el id del objeto agregado, y el resultado es el mismo que regresara el método PostEstado, el cual no es otra cosa que el objeto agregado en formato json.



Podemos observar que el código http que nos regresa es el 201 que significa que un objeto fue creado.

3.1.7 Requerimiento tipo Put (Actualizar)

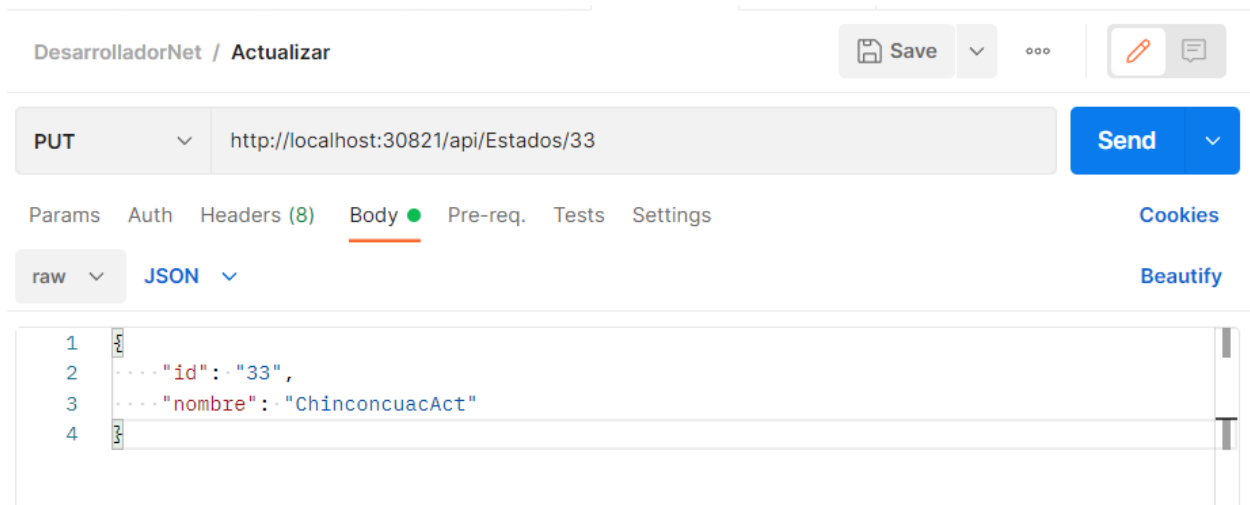
Para poder probar el método de Actualizar, necesitaremos utilizar un verbo PUT desde el postman. Para ello agregaremos otro requerimiento al cual le llamaremos Actualizar, con el verbo PUT, y dentro del cuerpo de la petición enviaremos en formato Json el objeto que queremos actualizar con sus nuevos datos.

De la misma manera que en el caso POST, la url será la que corresponde al controlador, sin embargo si vemos nuestro código

```
// PUT: api/Estados/5
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=21
[HttpPut("{id}")]
public async Task<IActionResult> PutEstados(int id, Estados estados)
{
    if (id != estados.Id)
    {
        return BadRequest();
    }
}
```

Observamos que cuando se define el verbo al cual responderá el método se está indicando que se debe de ingresar el parámetro id a través de la propia url.

Por lo que nuestro requerimiento quedará de la siguiente manera



Hay que tener cuidado que el id que se envía a través de la url es exactamente igual al que tiene en el Body. Ya que como podemos observar en nuestro código de nuestra web api que generó la plantilla se hace la validación de que deben ser iguales y en caso de que no se cumpla regresa un código de Bad Request

```
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> PutEstados(int id, Estados estados)
{
    if (id != estados.Id)
    {
        return BadRequest();
    }
}
```

Finalmente, este método, si no hubo problemas nos regresa un código 204, indicado que la solicitud fue realizada sin ningún problema.

3.1.8 Requerimiento tipo Delete (Eliminar)

Para poder probar el método de Eliminar, necesitaremos utilizar un verbo DELETE desde el postman. Para ello agregaremos otro requerimiento al cual le llamaremos Eliminar, con el verbo DELETE, y la url será la misma que utilizamos para consultar un solo elemento, ya que si vemos el código así nos lo indica.

Finalmente, este método, si no hubo problemas nos regresa un código 204, indicado que la solicitud fue realizada sin ningún problema.

<

Estado

GET Get

POST Agri

PUT Acti

DEL Elimi

>

+

...

No Environment



▼

DesarrolladorNet / Estados / Eliminar

Save

▼

...



DELETE

▼

http://localhost:30821/api/Estados/33

Send

▼

Params

Auth

Headers (6)

Body

Pre-req.

Tests

Settings


Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body

▼

 204 No Content 3 m 37.44 s 115 B

Save Response

▼

Pretty


Raw



Preview

Visualize

Text

▼





4 Consumir Web API

4.1 Consumir una Web API con HttpClient

Para consumir un Web API en el lado del servidor ASP.NET MVC, podemos usar HttpClient en el controlador MVC. HttpClient envía una solicitud a la API web y recibe una respuesta. Luego, debemos convertir los datos de respuesta que provienen de la API web en un modelo y posteriormente representarlos en una vista.

4.1.1 Crear un Proyecto de tipo Aplicación Web ASP .NET (.NET Framework)

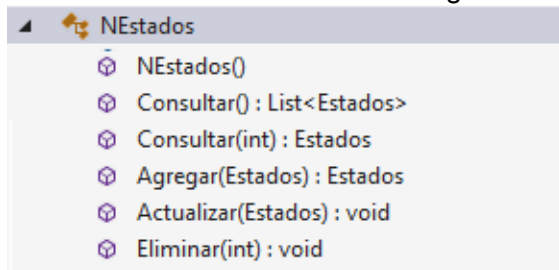
4.1.2 Crear una clase tipo POCO

Crear la clase `Estados` dentro de la carpeta Models

```
public class Estados
{
    public int id { get; set; }
    public string nombre { get; set; }
}
```

4.1.3 Crear la clase de Negocio

Crear la clase `NEstados` con los siguientes métodos, dentro de la Carpeta Models



Desde esta clase accedaremos la Web API creada mediante la clase HttpClient, para lo cual debemos agregar las siguientes usings

```
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using System.Text;
```

Además, necesitaremos serializar los objetos a enviar al Web API así como deserializar los objetos recibidos desde la misma API, por lo que requerimos utilizar Newtonsoft,

```
using Newtonsoft.Json;
```

Clase HttpClient

Métodos

GetAsync (String)	Envía una solicitud GET al URI especificado como una operación asincrónica.
PostAsync (String, HttpContent)	Envía una solicitud POST al URI especificado como una operación asincrónica.
PutAsync (Uri, HttpContent)	Envía una solicitud PUT al URI especificado como una operación asincrónica.
DeleteAsync (String)	Envía una solicitud DELETE al URI especificado como una operación asincrónica.

Estos cuatro métodos nos devolverán un objeto

Task<HttpResponseMessage>, el cual representa una operación que devuelve un valor y que normalmente se ejecuta de forma asincrónica.

Al ser una operación asíncrona deberemos esperar a que se complete la ejecución del objeto **Task**, invocando su método **Wait()**;

Toda vez que ha terminado la ejecución de la tarea, en nuestro caso la invocación al web API, podemos acceder al objeto devuelto por el llamado a la Web API, esto lo haremos accediendo a la propiedad **Result** del objeto **Task<HttpResponseMessage>**, y como podemos ver en su parámetro de tipo el objeto que obtendremos será del tipo **HttpResponseMessage**

Este objeto **HttpResponseMessage**, entre otras, tiene las siguientes propiedades que nos permitirán conocer si la operación se realizó correctamente y en su caso obtener el mensaje de respuesta:

Propiedad	Descripción
IsSuccessStatusCode	Obtiene un valor que indica si la respuesta HTTP se realizó correctamente.
StatusCode	Obtiene o establece el código de estado de la respuesta HTTP
Content	Obtiene o establece el contenido de un mensaje de respuesta HTTP

Ahora bien, la propiedad **Content** es un objeto de tipo **HttpContent**, la cual representa un cuerpo de entidad HTTP y encabezados de contenido, y para obtener el resultado debemos invocar el método **ReadAsStringAsync()**, el cual serializa el contenido HTTP en una cadena como una operación asincrónica. Al ser una operación asíncrona debemos invocar el método **Wait()** para que se espere a que se complete la ejecución

del objeto. Finalmente, como se realizó el objeto de respuesta tendremos un json del objeto resultado.

Método Consultar()

Con lo anteriormente expuesto, podemos implementar el método Consultar() para que regrese una lista de estados

```
public List<Estados> Consultar()
{
    var estados = new List<Estados>();
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Invoca el método GetAsync del objeto HttpClient, el cual envía una solicitud GET al
            //URI especificado como parámetro, como una operación asincrónica
            Task<HttpResponseMessage> responseTask = client.GetAsync("http://localhost:30821/api/Estados");

            // Se invoca al método Wait a fin de esperar a que se complete la operación asincrona
            responseTask.Wait();

            //Obtenemos el objeto HttpResponseMessage a través de la propiedad Result del objeto Task<HttpResponseMessage>
            HttpResponseMessage result = responseTask.Result;

            // Verificamos que la operación haya sido ejecutada con éxito, para proceder a obtener el resultado enviado
            // desde la web api, en caso contrario enviamos una excepción
            if (result.IsSuccessStatusCode)
            {
                //Invocamos al método ReadAsStringAsync del objeto HttpContent el cual serializa
                //el contenido HTTP en una cadena como una operación asincrónica.
                Task<string> readTask = result.Content.ReadAsStringAsync();

                // Se invoca al método Wait a fin de esperar a que se complete la operación asincrona
                readTask.Wait();

                //Obtenemos el string en formato json del objeto recibido
                string json = readTask.Result;

                //Deserializamos el objeto recibido, en este caso una lista de estados
                estados = JsonConvert.DeserializeObject<List<Estados>>(json);
            }
            else //web api sent error response
            {
                throw new Exception($"WebAPI. Respondio con error.{result.StatusCode}");
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
    return estados;
}
```

De la misma manera, podemos implementar el método Consultar(int id) para que regrese un estado. Lo único que habría que considerar es que hay que enviarle al Web API el id como parámetro dentro del URI.

Método Consultar(int id)

```
public Estados Consultar(int id)
{
    Estados estados = null;
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Invoca el método GetAsync del objeto HttpClient, el cual envía una solicitud GET al
            //URI especificado como parámetro, como una operación asíncronica
            var responseTask = client.GetAsync(urlWebAPI + $"/{id}");

            // Se invoca al método Wait a fin de esperar a que se complete la operación asincrona
            responseTask.Wait();

            //Obtenemos el objeto HttpResponseMessage a través de la propiedad Result del objeto Task<HttpResponseMessage>
            var result = responseTask.Result;

            // Verificamos que la operación haya sido ejecutada con éxito, para proceder a obtener el resultado enviado
            // desde la web api, en caso contrario enviamos una excepción
            if (result.IsSuccessStatusCode)
            {
                //Invocamos al método ReadAsStringAsync del objeto HttpContent el cual serializa
                //el contenido HTTP en una cadena como una operación asíncronica.
                var readTask = result.Content.ReadAsStringAsync();

                // Se invoca al método Wait a fin de esperar a que se complete la operación asincrona
                readTask.Wait();

                //Obtenemos el string en formato json del objeto recibido
                string json = readTask.Result;

                //Deserealizamos el objeto recibido, en este caso un estado
                estados = JsonConvert.DeserializeObject<Estados>(json);
            }
            else //web api envió error de respuesta
            {
                throw new Exception($"WebAPI. Respondio con error.{result.StatusCode}");
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
    return estados;
}
```

Método Agregar(Estados estado)

El método agregar se implementa más o menos de la misma forma, solo que hay que considerar que en el contenido de la petición HTTP debemos enviarle el objeto que queremos que la web API agregue, y asimismo en el encabezado hay que indicarle el tipo de contenido.

```

public Estados Agregar(Estados estado)
{
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Creamos un objeto HttpContent instanciando un objeto StringContent, la cual es una clase derivada de HttpContent.
            //Este contenido se crea con el objeto Estado que se está recibiendo
            HttpContent httpContent = new StringContent(JsonConvert.SerializeObject(estado), Encoding.UTF8);

            //Asignamos a la propiedad ContentType del encabezado de HttpContent
            httpContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            //Invoca el método PostAsync del objeto HttpClient, el cual envía una solicitud POST al
            //URI especificado como parámetro, como una operación asincrónica, asimismo le envía el contenido (objeto estado) dentro del httpContent
            var postTask = client.PostAsync(urlWebAPI, httpContent);

            // Se invoca al método Wait a fin de esperar a que se complete la operación asincrónica
            postTask.Wait();

            //Obtenemos el objeto HttpResponseMessage a través de la propiedad Result del objeto Task<HttpResponseMessage>
            var result = postTask.Result;

            // Verificamos que la operación haya sido ejecutada con éxito, para proceder a obtener el resultado enviado
            // desde la web api, en caso contrario enviamos una excepción
            if (result.IsSuccessStatusCode)
            {
                // Verificamos que la operación haya sido ejecutada con éxito, para proceder a obtener el resultado enviado
                // desde la web api, en caso contrario enviamos una excepción
                var readTask = result.Content.ReadAsStringAsync();
                // Se invoca al método Wait a fin de esperar a que se complete la operación asincrónica
                readTask.Wait();
                //Obtenemos el string en formato json del objeto recibido
                string json = readTask.Result;
                //Deserealizamos el objeto recibido, en este caso un estado
                estado = JsonConvert.DeserializeObject<Estados>(json);
            }
            else //web api envió error de respuesta
            {
                throw new Exception($"WebAPI. Respondio con error.{result.StatusCode}");
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
    return estado;
}

```

Método Actualizar(Estados estado)

El método Actualizar se implementa más o menos de la misma forma que el método agregar ya que también hay que enviarle el body el objeto que vamos a actualizar, además hay que recordar que en PutEstados(int id, Estados estados) recibe dos parámetros, uno en la url y otro en el body del body del http.

```

public void Actualizar(Estados estado)
{
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Creamos un objeto HttpContent instanciando un objeto StringContent, la cual es una clase derivada de HttpContent.
            //Este contenido se crea con el objeto Estado que se está recibiendo
            HttpContent httpContent = new StringContent(JsonConvert.SerializeObject(estado), Encoding.UTF8);

            //Asignamos a la propiedad ContentType del encabezado de HttpContent
            httpContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            //Invoca el método PutAsync del objeto HttpClient, el cual envía una solicitud PUT al
            //URI especificado como parámetro, incluyendo el id del estado, como una operación asincrónica,
            //asimismo le envía el contenido (objeto estado) dentro del httpContent
            var postTask = client.PutAsync(urlWebAPI + $"{estado.id}", httpContent);

            // Se invoca al método Wait a fin de esperar a que se complete la operación asincrónica
            postTask.Wait();

            //Obtenemos el objeto HttpResponseMessage a través de la propiedad Result del objeto Task<HttpResponseMessage>
            var result = postTask.Result;

            // Verificamos que si la operación no fue exitosa se levanta una excepción
            if (!result.IsSuccessStatusCode)
            {
                throw new Exception($"WebAPI. Respondio con error.{result.StatusCode}");
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
}

```

Método Eliminar(int id)

El método Eliminar es el método más sencillo, ya que solamente se le envía el id dentro del URL al invocar el método DeleteAsync, y no recibe respuesta.

```

public void Eliminar(int id)
{
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Invoca el método DeleteAsync del objeto HttpClient, el cual envía una solicitud DELETE al
            //URI especificado como parámetro, incluyendo el id del estado, como una operación asíncronica,
            var responseTask = client.DeleteAsync(urlWebAPI + $"/{id}");

            // Se invoca al método Wait a fin de esperar a que se complete la operación asíncrona
            responseTask.Wait();

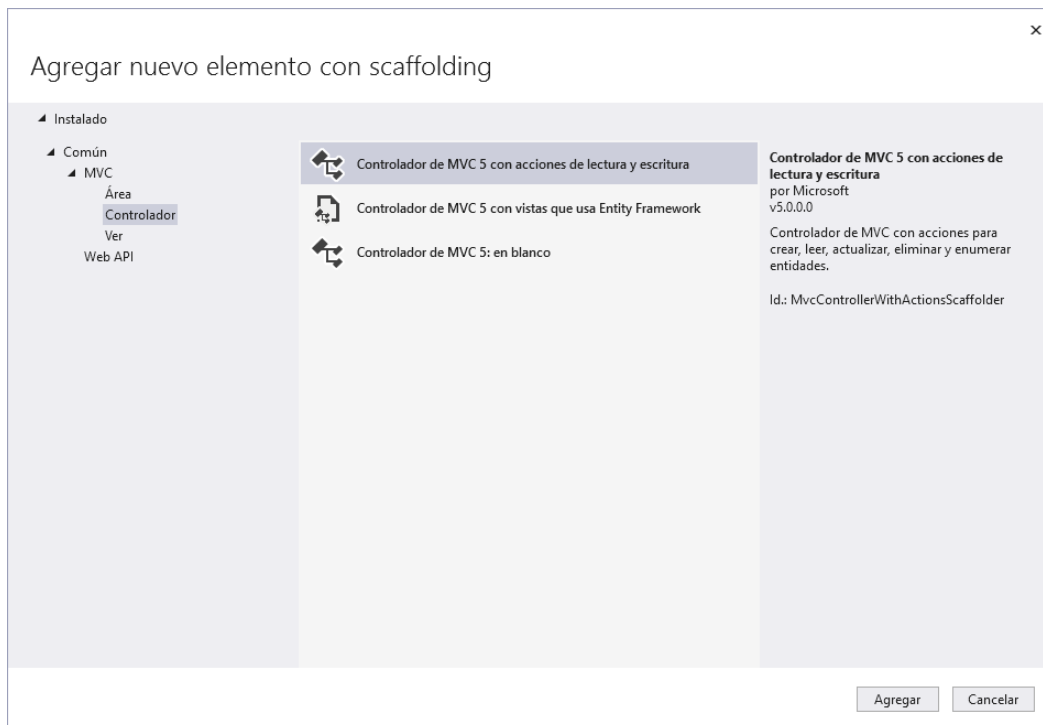
            //Obtenemos el objeto HttpResponseMessage a través de la propiedad Result del objeto Task<HttpResponseMessage>
            var result = responseTask.Result;

            // Verificamos que si la operación no fue exitosa se levanta una excepción
            if (!result.IsSuccessStatusCode)
            {
                throw new Exception($"WebAPI. Respondio con error.{result.StatusCode}");
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
}

```

4.1.4 Crear el controlador Estados

Crear el controlador Estados dentro de la carpeta de Controllers, con la plantilla Controlador de MVC 5 con acciones de lectura y escritura



4.1.5 Implementar los métodos de acción del controlador Estados

Considerando que se tiene implementada la clase NEstados con los métodos necesarios para realizar el CRUD a través del Web API, implementar los diferentes métodos de Acción del Controlador creado utilizando esta clase.

```
public class EstadosController : Controller
{
    private NEstados nEstados = new NEstados();

    public ActionResult Index()
    {
        IEnumerable<Estados> estados = null;
        try
        {
            estados = nEstados.Consultar();
        }
        catch (Exception ex)
        {
            estados = Enumerable.Empty<Estados>();
            ModelState.AddModelError(string.Empty, $"Server error. Mensaje: {ex.Message}");
        }
        return View(estados);
    }

    public ActionResult Details(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }

        Estados estado = new Estados();
        try
        {
            estado = nEstados.Consultar(Convert.ToInt32(id));
        }
        catch (Exception ex)
        {
            ModelState.AddModelError(string.Empty, $"Server error. Mensaje: {ex.Message}");
        }
        return View(estado);
    }
}
```



```
public ActionResult Create()
{
    return View();
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
```

0 referencias

```
public ActionResult Create([Bind(Include = "id,nombre")] Estados estado)
{
```

```
    if (ModelState.IsValid)
    {
        try
        {
            nEstados.Agregar(estado);
            return RedirectToAction("Index");
        }
        catch (Exception ex)
        {
            ModelState.AddModelError(string.Empty, $"Server Error. Mensaje: {ex.Message}");
        }
    }
}
```

```
    return View(estado);
}
```

```
public ActionResult Edit(int? id)
```

```
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
}
```

```
    Estados estado = new Estados();
```

```
    try
    {
        estado = nEstados.Consultar(Convert.ToInt32(id));
    }
```

```
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, $"Server Error. Mensaje: {ex.Message}");
    }
}
```

```
    return View(estado);
```

```
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public ActionResult Edit([Bind(Include = "id,nombre")] Estados estado)
{
    if (ModelState.IsValid)
    {
        try
        {
            nEstados.Actualizar(estado);
            return RedirectToAction("Index");
        }
        catch (Exception ex)
        {
            ModelState.AddModelError(string.Empty, $"Server Error. Mensaje: {ex.Message}");
        }
    }

    return View(estado);
}

public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Estados estado = new Estados();
    try
    {
        estado = nEstados.Consultar(Convert.ToInt32(id));
        if (estado == null)
        {
            return HttpNotFound();
        }
    }
    catch (Exception)
    {
        throw;
    }
    return View(estado);
}
```

```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 referencias
public ActionResult DeleteConfirmed(int id)
{
    try
    {
        nEstados.Eliminar(Convert.ToInt32(id));
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, $"Server Error. Mensaje: {ex.Message}");
    }

    return RedirectToAction("Index");
}

```

4.1.6 Crear las vistas correspondientes a cada uno de los métodos de acción

4.2 Consumir una Web API con AJAX

Se puede acceder a la Web API directamente desde la interfaz de usuario en el lado del cliente utilizando las capacidades AJAX de cualquier framework de JavaScript como JQuery, AngularJS, KnockoutJS, Ext JS, etc.

4.2.1 Habilitar CORS con política con nombre y middleware

CORS Middleware maneja solicitudes de origen cruzado. El siguiente código aplica una política CORS a todos los puntos finales de la aplicación con los orígenes especificados:

En la Web API realizar las siguientes acciones:

Paso 1.- Instalar el paquete Nuget **Microsoft.AspNetCore.Cors**

Paso 2.- En el método **ConfigureServices** del **startup.cs**, agregue los servicios CORS.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApiContext>(options =>
    {
        options.UseSqlServer(Configuration.GetConnectionString("InstitutoTich"));
    });
    //
    services.AddCors();
    //
    services.AddControllers();
}

```

```

//
    services.AddCors();
//

```

Paso 3.- En el método **Configure** de su **startup.cs**, agregue lo siguiente:

```

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseAuthorization();
    //
    string urlAppCaller = Configuration["URLAppCaller"];
    app.UseCors(
        options => options.WithOrigins(urlAppCaller).AllowAnyMethod().AllowAnyHeader()
    );
    //
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}

```

```

//
string urlAppCaller = Configuration["URLAppCaller"];
app.UseCors(
    options => options.WithOrigins(urlAppCaller).AllowAnyMethod().AllowAnyHeader()
    );
//

```

Con la primera línea se está obteniendo la URL, desde el archivo de configuración, a la que se le dará permisos de hacer invocaciones.

4.2.2 Archivo de Configuración appsettings.json

La configuración de aplicaciones en ASP.NET Core el archivo de configuración es el appsettings.json, este archivo contiene una cadena en formato json con pares Key-value, al cual se le pueden agregar los diferentes parámetros dentro de esta cadena.

En este caso agregaremos el key "URLAppCaller" y como value la url de la aplicación que consumirá el web api.

```
{
  "ConnectionStrings": {
    "InstitutoTich": "Server=MSI;Database=InstitutoTich;User ID=sa;password=Pass2017;"
  },
  "URLAppCaller": "http://localhost:53846",
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}

//
"URLAppCaller": "http://localhost:53846",
//
```

Para recuperar los valores de los parámetros dentro de la aplicación, se puede hacer mediante el objeto de tipo IConfiguration que se recibe en el constructor del Startup.

A este objeto tiene una propiedad de tipo arreglo que almacena todos los pares key-value y que proporcionando el key se obtiene el valor

```
string urlAppCaller = Configuration["URLAppCaller"];
```

4.2.3 Consumir el Web API desde el cliente de la aplicación

Agregar un botón Eliminar Ajax a la vista Index

Ya que se ha configurado el Web API para que pueda recibir peticiones de otro dominio, agregar un botón a la vista Index, con la finalidad de realizar la eliminación de un registro haciendo llamadas al Web API a través de AJAX.

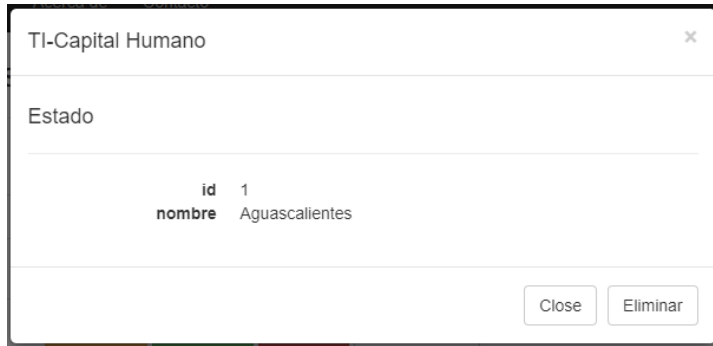
Catálogo de Estados

Crear

nombre				
Aguascalientes	Actualizar	Consultar	Eliminar	Eliminar Ajax
Baja California	Actualizar	Consultar	Eliminar	Eliminar Ajax
Baja California Sur	Actualizar	Consultar	Eliminar	Eliminar Ajax
Campeche	Actualizar	Consultar	Eliminar	Eliminar Ajax
Chihuahua	Actualizar	Consultar	Eliminar	Eliminar Ajax

```
<a href="javascript:void(0)" name="btnEliminarAjax" id="btnEliminarAjax" class="btn btn-default" onclick="eliminar(@item.id)">Eliminar Ajax</a>
```

Cuando se haga click al botón, desde el cliente se hará la solicitud Ajax al web API de consulta del Estado correspondiente. Con la respuesta se llenará la ventana modal previamente definida.



Definición de la Ventana Modal

```
<div class="container">
  <!-- Modal -->
  <div class="modal fade" id="eliminarModal" role="dialog">
    <div class="modal-dialog">
      <!-- Modal content -->
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal">&times;</button>
          <h4 class="modal-title">TI-Capital Humano</h4>
        </div>
        <div class="modal-body">
          <h4>Estado</h4>
          <hr />
          <dl class="dl-horizontal">
            <dt>
              id
            </dt>
            <dd id="id">
            </dd>
            <dt>
              nombre
            </dt>
            <dd id="nombre">
            </dd>
          </dl>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
          <a id="btnEliminarModal" name="submit" title="Guardar" class="btn btn-default left-align">Eliminar</a>
        </div>
      </div>
    </div>
  </div>
</div>
```

Asignación de manejador de evento click al botón “btnEliminarModal”

```
function load() {  
    var btnEliminar = document.getElementById("btnEliminarModal");  
    btnEliminar.addEventListener("click", eliminarConfirmado, false);  
}
```

Función eliminar para hacer el llamado AJAX con verbo GET al Web API y levantar la ventana modal

```
function eliminar(id) {  
    var url = 'http://localhost:30821/api/Estados/' + id;  
    $.ajax({  
        type: 'GET',  
        url: url,  
        contentType: "application/json; charset=utf-8",  
        dataType: "json",  
        success: function (estado) {  
            $("#id").html(estado.id);  
            $("#nombre").html(estado.nombre);  
        },  
        failure: function (data) {  
            alert(data.responseText);  
        },  
        error: function (data) {  
            alert(data.responseText);  
        }  
    });  
    $("#eliminarModal").modal();  
}
```

Al oprimir el botón “Eliminar de la ventana modal hace el llamado AJAX con verbo DELETE al Web API para eliminar el elemento

```
function eliminarConfirmado() {  
    var url = 'http://localhost:30821/api/Estados/' + $("#id").html();  
    return $.ajax({  
        type: 'DELETE',  
        url: url,  
        contentType: "application/json; charset=utf-8",  
        dataType: "json",  
        success: function (data) {  
            alert("Operación Exitosa");  
        },  
        failure: function (data) {  
            alert(data.responseText);  
        },  
        error: function (data) {  
            alert(data.responseText);  
        }  
    });  
}
```