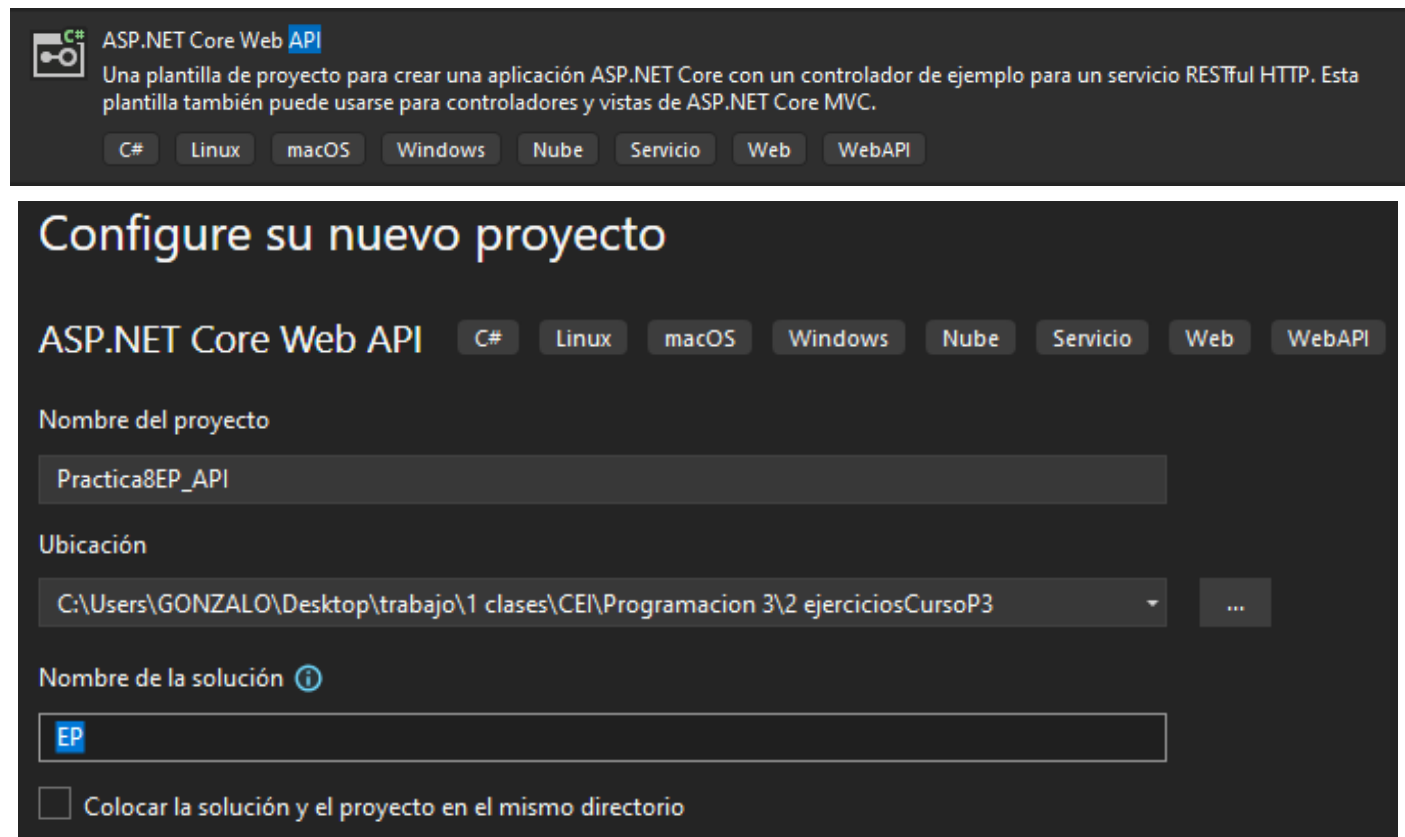


## Práctica 8 Ejercicios de API

## 1- Primer End Point

a) Crear un nuevo proyecto.



ASP.NET Core Web API

Una plantilla de proyecto para crear una aplicación ASP.NET Core con un controlador de ejemplo para un servicio RESTful HTTP. Esta plantilla también puede usarse para controladores y vistas de ASP.NET Core MVC.

C# Linux macOS Windows Nube Servicio Web WebAPI

## Configure su nuevo proyecto

ASP.NET Core Web API C# Linux macOS Windows Nube Servicio Web WebAPI

Nombre del proyecto

Practica8EP\_API

Ubicación

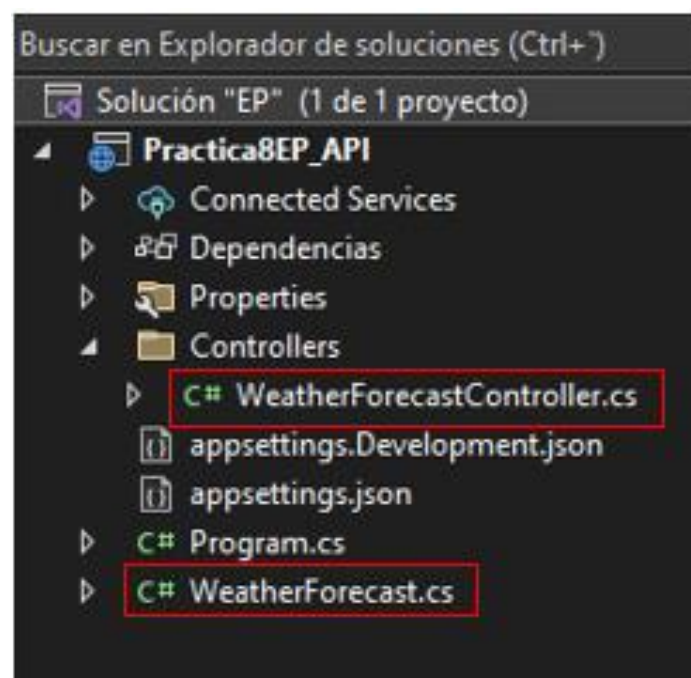
C:\Users\GONZALO\Desktop\trabajo\1 clases\CEI\Programacion 3\2 ejerciciosCursoP3

Nombre de la solución ⓘ

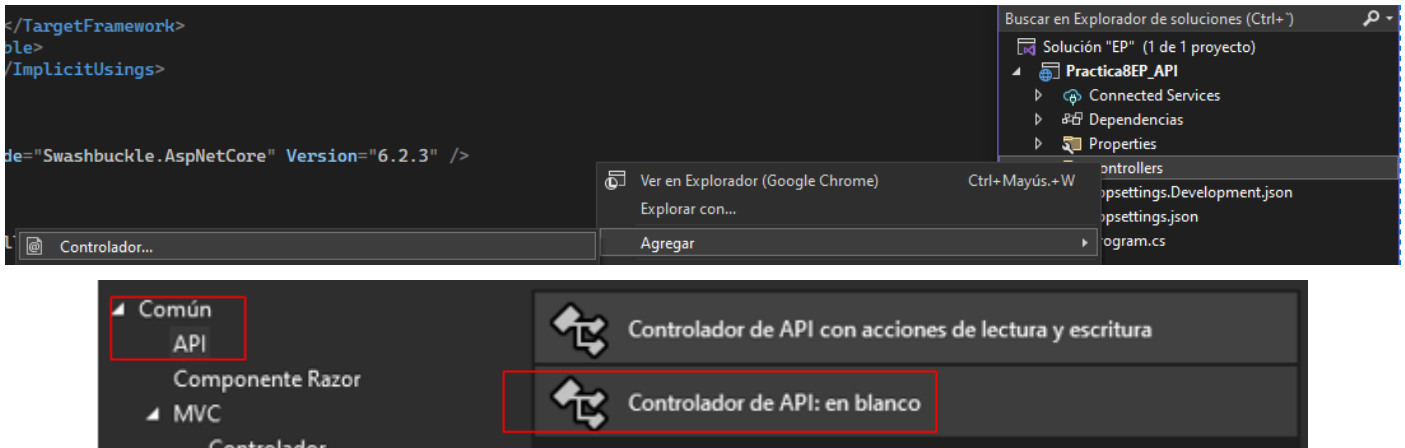
EP

☐ Colocar la solución y el proyecto en el mismo directorio

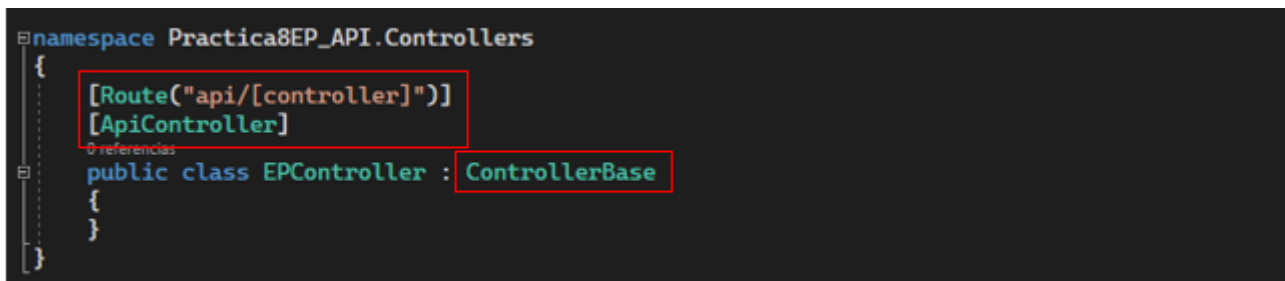
b) Eliminar el controlador y la clase de ejemplo, creados por defecto.



c) Agregar controlador en blanco.

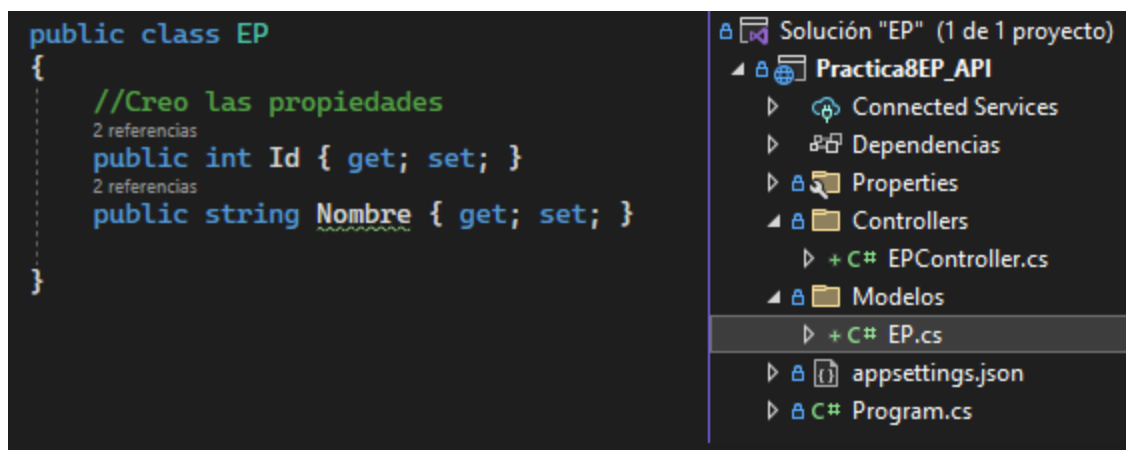


Un controlador es una clase que hereda de ControllerBase.



d) Crear las clases y configurarlas.

- i) Crear un directorio Modelos dentro del proyecto principal.
- ii) Dentro de modelos agregar una nueva clase EP.
- iii) Crear las propiedades dentro de la clase.

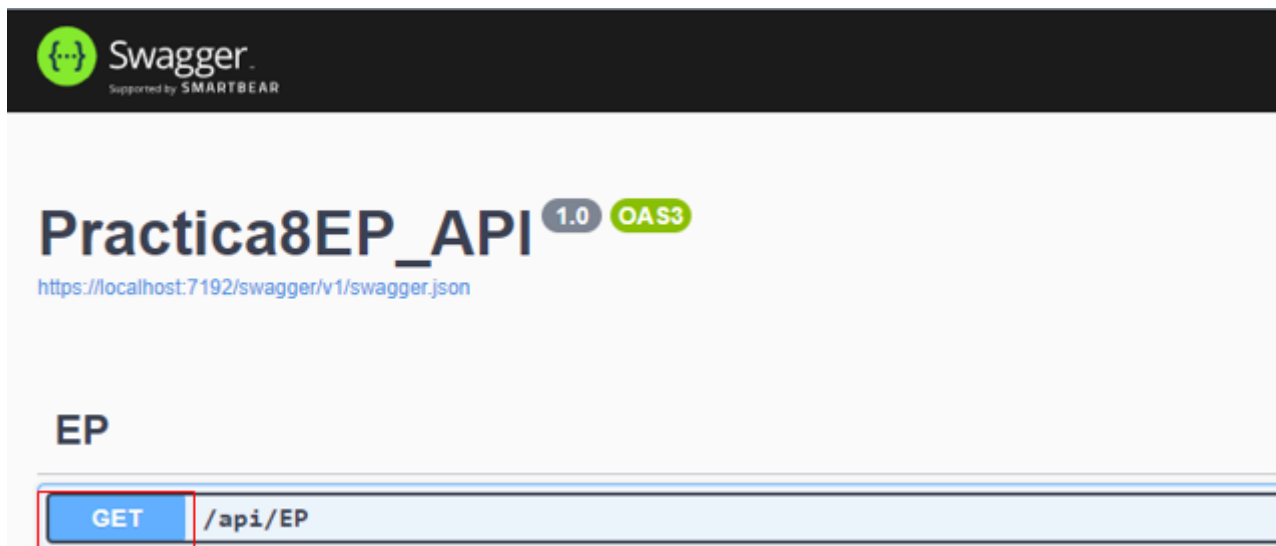


- iv) En el controlador crear el primer End Point, que sea un método GET, llamado GetPD().
- v) GetPD() retorna una lista de EP (List<EP>) con dos instancias.
- vi) Indicar el tipo de verbo http por medio de [HttpGet].

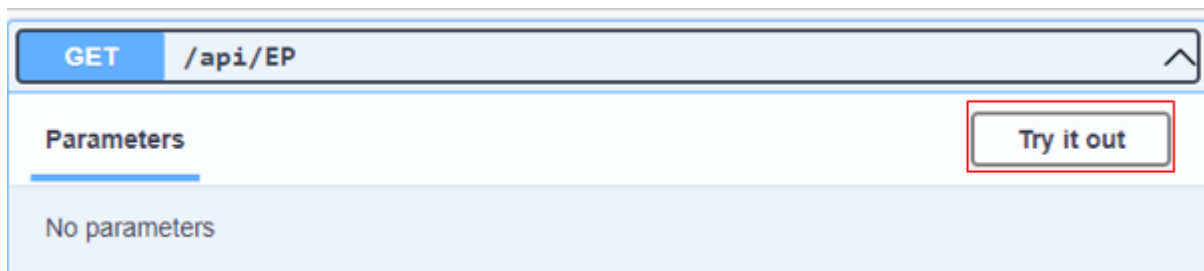
```
{  
    [Route("api/[controller]")]  
    [ApiController]  
    0 referencias  
    public class EPController : ControllerBase  
    {  
        //Creo los End Points  
        [HttpGet]//Tengo que definir el tipo de verbo http  
        0 referencias  
        public IEnumerable<EP> GetEP()  
        {  
            return new List<EP>  
            {  
                new EP{Id=1,Nombre="Vista a la Piscina"},  
                new EP { Id = 2, Nombre = "Vista a la Playa" }  
            };  
        }  
    }  
}
```

e) Se debe abrir automáticamente la documentación de Swagger.

i) Click en GET



ii) Click en Try in out.



iii) Se despliega la url de la respuesta y la respuesta.

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7192/api/EP' \
  -H 'accept: text/plain'
```

Request URL

https://localhost:7192/api/EP

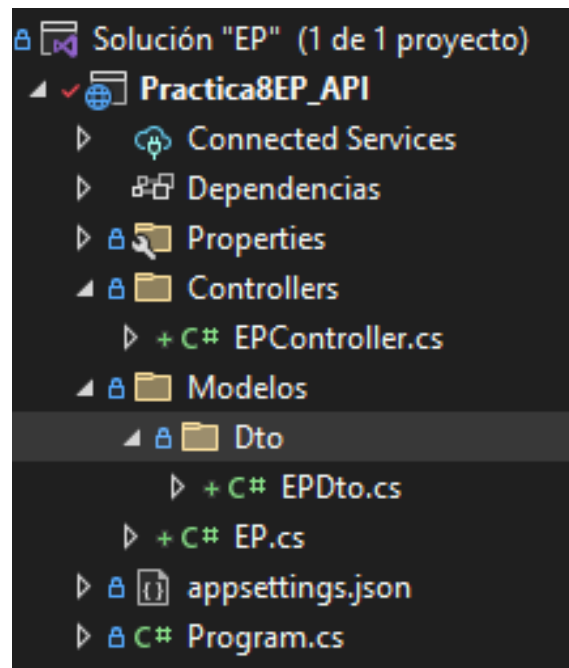
Server response

Code	Details
200	<p>Response body</p> <pre>[   {     "id": 1,     "nombre": "Vista a la Piscina"   },   {     "id": 2,     "nombre": "Vista a la Playa"   } ]</pre>

## 2. Agregar un DTO (Data transfer Object).

En aplicaciones reales no es recomendable trabajar con el modelo directamente. Entonces se crea un DTO para crear una envoltura del modelo.

a) Crear el directorio DTO dentro de Modelos. Crear una nueva clase dentro de DTO, de nombre EPDto con las mismas propiedades que EP.



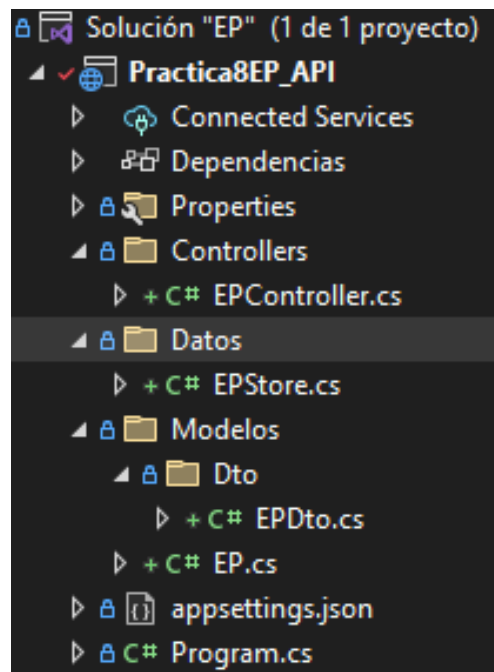
b) Modificar el método GET en el controlador.

```
public IEnumerable<EPDto> GetEPDto()
{
    return new List<EPDto>
    {
        new EPDto{Id=1,Nombre="Vista a la Piscina"},
        new EPDto{ Id = 2, Nombre = "Vista a la Playa" }
    };
}
```

c) Verificar el método desde Swagger.

### 3. Simular un Store donde este almacenado toda la información.

a) Crear el directorio Datos dentro del proyecto principal. Crear una nueva clase dentro de Datos, de nombre EPStore con las mismas propiedades.



b) Crear una lista estática dentro de la clase, con las instancias de EPController.

```
public class EPStore
{
    public static List<EPDto> EPList = new List<EPDto>
    {
        new EPDto{Id=1,Nombre="Vista a la Piscina"},
        new EPDto{ Id = 2, Nombre = "Vista a la Playa" }
    };
}
```

c) Modificar el método GET en el controlador.

```
public IEnumerable<EPDto> GetEPDto()
{
    return EPStore.EPList;
}
```

d) Verificar el método desde Swagger.

#### 4. Segundo End Point

a) Crear un End Point, que seleccione un elemento de la lista según un parámetro.

```
//***** Eercicio3 *****/
[HttpGet]//Retorna una lista
0 referencias
public IEnumerable<EPDto> GetEPDto()
{
    return EPStore.EPList;
}

//***** Eercicio4 *****/
[HttpGet("Id")]//Retorna un objeto. Hay que diferencial los End Point
0 referencias
public EPDto GetEPDto(int Id)
{
    return EPStore.EPList.FirstOrDefault(a => a.Id == Id);
}
```

b) Verificar el método desde Swagger

**EP**

**GET** /api/EP

**GET** /api/EP/Id

**GET** /api/EP/Id

**Parameters**

Name	Description
Id	

integer(\$int32) 1 (query)

**Response body**

```
{
  "id": 1,
  "nombre": "Vista a la Piscina"
}
```

## 5. Definir tipos de retornos.

- Implementar la interfaz IActionResult para ambos End Points. De esta forma se puede obtener el tipo de retorno que sea conveniente.
- Retornar un Objeto "Ok" en cada End Point.
- Agregar validaciones para para en End Point que recibe un parámetro Id.

```
[HttpGet]
0 referencias
public IActionResult GetEPDto()
{
    return Ok(EPStore.EPList); //Responde con un codigo de estado 200
}
[HttpGet("Id")] //Retorna un objeto. Hay que diferencial los End Point
0 referencias
public IActionResult GetEPDto(int Id)
{
    if (Id == 0)
    {
        return BadRequest(); //Responde con un codigo de estado 400
    }

    var Ep = EPStore.EPList.FirstOrDefault(a => a.Id == Id);

    if (Ep == null)
    {
        return NotFound(); //Responde con un codigo de estado 404
    }

    return Ok(Ep);
}
```

El BadRequest se implementa para para validar que no se haga una solicitud incorrecta. Por otro lado, si no se encuentra un registro relacionado al Id se retorna un NotFound.

- Hacer una solicitud correcta y una incorrecta.

**GET** /api/EP/Id

**Parameters**

Name	Description
Id	
integer(\$int32)	1
(query)	

**200** Response body

```
{
  "id": 1,
  "nombre": "Vista a la Piscina"
}
```

GET /api/EP/Id

Parameters

Name	Description
Id	
integer(\$int32)	0
(query)	

400

Undocumented

Error: response status is 400

Response body

```
{
  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
  "title": "Bad Request",
  "status": 400,
  "traceId": "00-a2dbcae6c21e4097b1a4adb74fcd2883-a011cdd9b2934c39-00"
}
```

e) Verificar NotFound.

GET /api/EP/Id

Parameters

Name	Description
Id	
integer(\$int32)	5
(query)	

404

Undocumented

Error: response status is 404

Response body

```
{
  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",
  "title": "Not Found",
  "status": 404,
  "traceId": "00-e5fc3191be9a5cefb75a8162335e7307-496b4eddb3d98846-00"
}
```



## 6. Documentar códigos de estado

La documentación de los códigos de estado en las APIs sirve para proporcionar información clara y concisa sobre el significado de cada código de estado devuelto por la API. Esto es útil para los desarrolladores que utilizan la API, ya que les ayuda a comprender mejor las respuestas que reciben y a tomar las acciones apropiadas en función de cada código de estado.

```
[HttpGet("Id")]//Retorna un objeto. Hay que diferencial los End Point
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 referencias
public ActionResult<EPDto> GetEPDto(int Id)
{
    if (Id == 0)
    {
        return BadRequest();//Responde con un codigo de estado 400
    }

    var Ep = EPStore.EPList.FirstOrDefault(a => a.Id == Id);

    if (Ep == null)
    {
        return NotFound();//Responde con un codigo de estado 404
    }

    return Ok(Ep);
}
```

**GET** /api/EP/Id

**Parameters**

Name	Description
Id	
integer(\$int32)	0
(query)	

400

↑

Error: response status is 400

Response body

```
{
  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
  "title": "Bad Request",
  "status": 400,
  "traceId": "00-d6f70f7861977a3b3d705ba6c58e012e-a641c140e4149cc2-00"
}
```

## 7. Crear un End Point que sea un método POST.

- Crear el método POST con las validaciones correspondientes.
- Documentar los códigos de estado.

- c) Dar un nombre al método GET que recibe un Id por parámetro. Generando una ruta de acceso al mismo.
- d) Utilizar el método "CreatedAtRoute" para llamar a la ruta del metodo GET y pasar los parámetros correspondientes.

```

/***** Ejercicio7 *****/
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
0 referencias
public ActionResult<IEnumerable<EPDto>> GetEPDto()
{
    return Ok(EPStore.EPList);
}

[HttpGet("Id", Name = "GetEP")] // Se agrega un nombre al End Point
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 referencias
public ActionResult<EPDto> GetEPDto(int Id)
{
    if (Id == 0)
    {
        return BadRequest();
    }
    var Ep = EPStore.EPList.FirstOrDefault(a => a.Id == Id);
    if (Ep == null)
    {
        return NotFound();
    }
    return Ok(Ep);
}

[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
0 referencias
public ActionResult<EPDto> CrearEPDto([FromBody] EPDto epdto) // [FromBody] indica recepcion de datos
{
    if (epdto == null)
    {
        return BadRequest(epdto);
    }
    if (epdto.Id > 0)
    {
        return StatusCode(StatusCodes.Status500InternalServerError);
    }
    epdto.Id = EPStore.EPList.OrderByDescending(e => e.Id).FirstOrDefault().Id + 1;
    EPStore.EPList.Add(epdto);
    return CreatedAtRoute("GetEP", new { id = epdto.Id }, epdto);
}

```

- e) Explicar líneas de código que considere necesarias.
- i) EPStore.EPList: Hace referencia a la lista "EPList" dentro de un objeto o una clase "EPStore".
  - ii) OrderByDescending(e => e.Id): Ordena los elementos de la lista en orden descendente según el valor de la propiedad "Id" de cada objeto.
  - iii) FirstOrDefault(): Devuelve el primer elemento de la lista después de aplicar el ordenamiento descendente.
  - iv) Id: Accede a la propiedad "Id" del primer elemento de la lista.

v) + 1: Añade 1 al valor obtenido en el paso anterior, lo que resulta en el siguiente valor disponible para la propiedad "Id" en la lista.

vi) EPStore.EPList.Add(epdto): Agrega un objeto llamado "epdto" a la lista "EPList".

vii) return CreatedAtRoute("GetEP", new {id= epdto.Id},epdto): Devuelve una respuesta HTTP con el código de estado "201 Created" y el objeto "epdto" como contenido.

Además, utiliza el método "CreatedAtRoute" para generar la ubicación de la ruta a la que se puede acceder para obtener el objeto "epdto". La ruta se identifica por el nombre "GetEP" y se pasa un parámetro de ruta "id" con el valor "epdto.Id".

f) Verificar el método Post desde Swagger.

**POST** /api/EP

**Parameters**

No parameters

**Request body**

```
{
  "id": 0,
  "nombre": "nueva villa"
}
```

**201**

**Response body**

```
{
  "id": 3,
  "nombre": "nueva villa"
}
```

g) Implementar DataAnnotation.

h) Validación del ModelState.

i) Validación de que los registros no se creen más de una vez.

```
8 referencias
public class EPDto
{
    8 referencias
    public int Id { get; set; }

    [Required]
    [MaxLength(20)]
    4 referencias
    public string Nombre { get; set; }
}
```

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
0 referencias
public ActionResult<EPDto> CrearEPDto([FromBody] EPDto epdto)
{
    if (!ModelState.IsValid)//Si el modelo es valido
    {
        return BadRequest(ModelState);
    }

    if (EPStore.EPList.FirstOrDefault(v=>v.Nombre.ToLower() == epdto.Nombre.ToLower()) !=null)
    {
        // Validacion personalizada
        ModelState.AddModelError("NombreExiste", "Ese nombre ya existe");
        return BadRequest(ModelState);
    }

    if (epdto ==null)
    {
        return BadRequest(epdto);
    }

    if (epdto.Id>0)
    {
        return StatusCode(StatusCodes.Status500InternalServerError);
    }

    epdto.Id = EPStore.EPList.OrderByDescending(e => e.Id).FirstOrDefault().Id + 1;
    EPStore.EPList.Add(epdto);
    return CreatedAtRoute("GetEP", new {id= epdto.Id},epdto);
}
```

j) Verificar el método desde Swagger.

400

Error: response status is 400

Response body

```
{
  "NombreExiste": [
    "Ese nombre ya existe"
  ]
}
```

## 8) Verbo HttpDelete

- Crear el método Delete con las validaciones correspondientes.
- Hacer validaciones.
- Documentar los códigos de estado.

```
/****** Eercicio8 *****/
[HttpDelete("{id:int}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 referencias
public IActionResult DeleteEP(int id) //A la Interface IActionResult no le hace falta el Modelo,
//ya que devuelve un NoContent()
{
    if (id == 0)
    {
        return BadRequest();
    }
    var ep = EPStore.EPList.FirstOrDefault(v=>v.Id==id);
    if (ep == null)
    {
        return NotFound();
    }
    EPStore.EPList.Remove(ep);
    return NoContent();
}
```

d) Verificar desde Swagger. Consulte los registros con el método GET, luego Agregue un registro con el método POST y finalmente elimine el registro con el método DELETE.

**GET** /api/EP

200

Response body

```
[
  {
    "id": 1,
    "nombre": "Vista a la Piscina"
  },
  {
    "id": 2,
    "nombre": "Vista a la Playa"
  }
]
```

**POST** /api/EP

201

Response body

```
{
  "id": 3,
  "nombre": "vista a la montaña"
}
```

**GET** /api/EP

200

Response body

```
[
  {
    "id": 1,
    "nombre": "Vista a la Piscina"
  },
  {
    "id": 2,
    "nombre": "Vista a la Playa"
  },
  {
    "id": 3,
    "nombre": "vista a la montaña"
  }
]
```

**DELETE** /api/EP/{id}

id \* required  
integer(\$int32)  
(path)

3

**GET** /api/EP

200

Response body

```
[
  {
    "id": 1,
    "nombre": "Vista a la Piscina"
  },
  {
    "id": 2,
    "nombre": "Vista a la Playa"
  }
]
```

## 9) Verbo HttpPut

a) Agregar más propiedades en EPDto y modificar los registros en EPStore.

```
public class EPDto
{
    10 referencias
    public int Id { get; set; }

    [Required]
    [MaxLength(20)]
    6 referencias
    public string Nombre { get; set; }

    4 referencias
    public int Ocupantes { get; set; }
    4 referencias
    public int MetrosCuadrados { get; set; }
}
```



```
public class EPStore
{
    public static List<EPDto> EPList = new List<EPDto>
    {
        new EPDto{Id=1,Nombre="Vista a la Piscina", Ocupantes=3, MetrosCuadrados=50},
        new EPDto{ Id = 2, Nombre = "Vista a la Playa", Ocupantes=2, MetrosCuadrados=100}
    };
}
```

- b) Crear el método Delete con las validaciones correspondientes.  
c) Hacer validaciones y documentar los códigos de estado.

```
/****** Ejercicio9 *****/
[HttpPut("{id:int}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
0 referencias
public IActionResult UpdateEP(int id, [FromBody] EPDto epdto) //Recibe el id y el objeto a actualizar
//ya que devuelve un NoContent()
{
    if (epdto == null)
    {
        return BadRequest();
    }
    var ep = EPStore.EPList.FirstOrDefault(v => v.Id == id);
    ep.Nombre = epdto.Nombre;
    ep.Ocupantes = epdto.Ocupantes;
    ep.MetrosCuadrados = epdto.MetrosCuadrados;
    return NoContent();
}
```

- d) Verificar desde Swagger. Consulte los registros con el método GET, luego actualizar un registro con el método PUT y volver a consultar.

GET

/api/EP

200

Response body

```
[
  {
    "id": 1,
    "nombre": "Vista a la Piscina",
    "ocupantes": 3,
    "metrosCuadrados": 50
  },
  {
    "id": 2,
    "nombre": "Vista a la Playa",
    "ocupantes": 2,
    "metrosCuadrados": 100
  }
]
```

PUT

/api/EP/{id}

## Parameters

Name	Description
<b>id</b> * required	
integer(\$int32)	1
(path)	

Request body

```
{
  "id": 1,
  "nombre": "Prueba de Cambio",
  "ocupantes": 5,
  "metrosCuadrados": 150
}
```

GET

/api/EP

200

Response body

```
[
  {
    "id": 1,
    "nombre": "Prueba de Cambio",
    "ocupantes": 5,
    "metrosCuadrados": 150
  },
  {
    "id": 2,
    "nombre": "Vista a la Playa",
    "ocupantes": 2,
    "metrosCuadrados": 100
  }
]
```

## 10) Verbo HttpPatch

a) Agregar paquetes desde el Administrador de paquetes de Nuget para trabajar con HttpPatch

.NET

**Microsoft.AspNetCore.JsonPatch** por Microsoft, 610M descargas  
ASP.NET Core support for JSON PATCH.

.NET

**Microsoft.AspNetCore.Mvc.NewtonsoftJson** por Microsoft, 262M descarga  
ASP.NET Core MVC features that use Newtonsoft.Json. Includes input and output formatters for JSON and JSON PATCH.



b) Agregar los nuevos servicios en la clase Program

```
builder.Services.AddControllers().AddNewtonsoftJson(); // Agregar
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

c) Hacer validaciones y documentar los códigos de estado.

```

/***** Ejercicio10 *****/
[HttpPatch("{id:int}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
0 referencias
public IActionResult UpdateEP(int id,JsonPatchDocument<EPDto> patchdto) //Hace referencia a la Libreria agrgada
{
    if (patchdto == null || id==0)
    {
        return BadRequest();
    }
    var ep = EPStore.EPList.FirstOrDefault(v => v.Id == id);
    patchdto.ApplyTo(ep, ModelState);
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    return NoContent();
}

```

d) Verificar desde Swagger. Actualizar un registro con el método PATCH y volver a consultar.

PATCH

/api/EP/{id}

Parameters

Name	Description
<b>id * required</b> integer(\$int32) (path)	<input type="text" value="1"/>

Request body

```
[
  {
    "path": "/nombre",
    "op": "replace",
    "value": "nueva vista"
  }
]
```

La "/" indica la propiedad a modificar

Operacion: Reemplazar

Valor: "Nuevo valor"

GET

/api/EP

200

Response body

```
[
  {
    "id": 1,
    "nombre": "nueva vista",
    "ocupantes": 3,
    "metrosCuadrados": 50
  },
  {
    "id": 2,
    "nombre": "Vista a la Playa",
    "ocupantes": 2,
    "metrosCuadrados": 100
  }
]
```