Tipo : Guía de Enunciado
Capítulo : ASP NET Core
Duración : 60 minutos

I. OBJETIVO

Crear una aplicación Web Api con ASP NET Core.

II. REQUISITOS

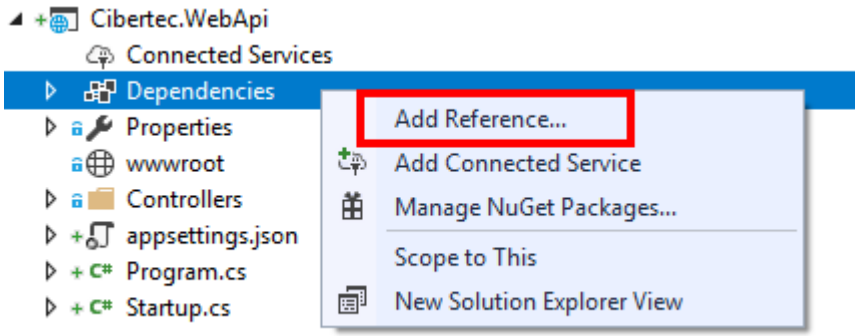
Los siguientes elementos de software son necesarios para la realización del laboratorio:

- Windows 10 (como mínimo Windows 8)
- Visual Studio 2017 como mínimo.

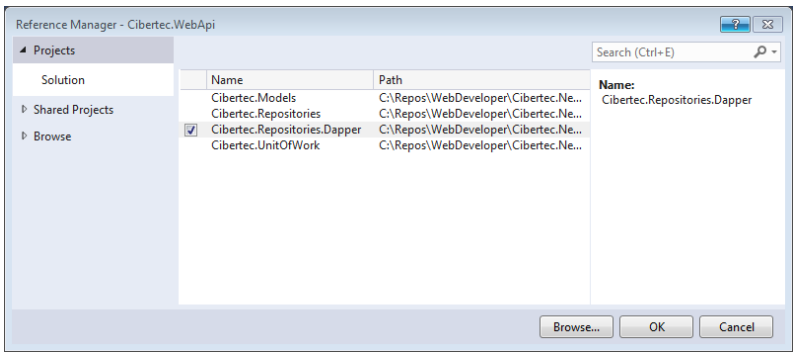
III. EJECUCIÓN DEL LABORATORIO

1. Abrir la solución "**Cibertec.NetCore.sln**" y ubicarse en el proyecto Cibertec.WebApi.

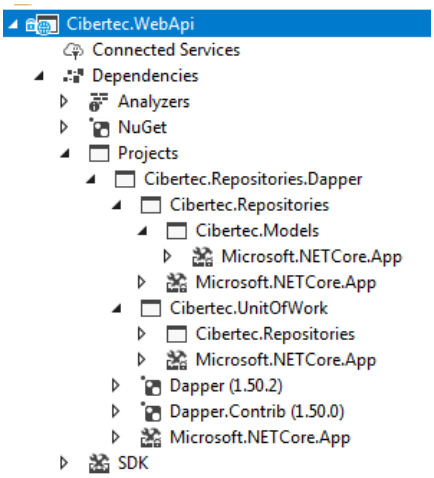
2. Agregamos las dependencias al proyecto "**Cibertec.Repositories.Dapper**", algo interesante de NET Core es que, a diferencia de .NET Framework; solo necesitas referenciar el proyecto principal y este cargara sus dependencias previas.



Y seleccionamos:



3. Podremos notar la siguiente estructura:



4. Eliminamos en controlador por defecto "**ValuesController**" de la carpeta Controllers.

5. Procedemos a crear nuestra clase base llamada "**BaseController**", con el siguiente código:

```
using Cibertec.UnitOfWork;
using Microsoft.AspNetCore.Mvc;

namespace Cibertec.WebApi.Controllers
{
    [Produces("application/json")]
    public class BaseController : Controller
    {
        protected IUnitOfWork _unit;
        public BaseController(IUnitOfWork unit)
        {
            _unit = unit;
        }
    }
}
```

6. Creamos el controlador "CustomerController" con el siguiente código:

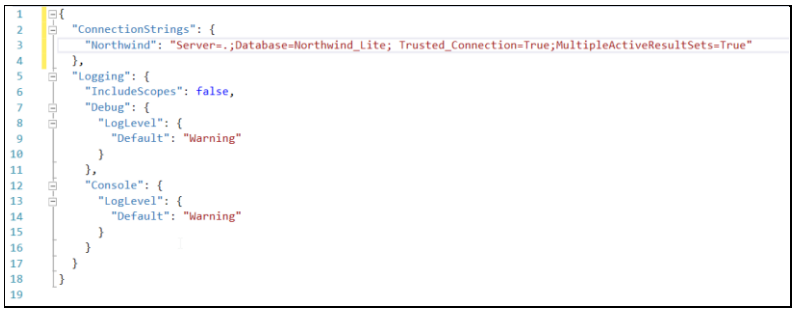
```
using Microsoft.AspNetCore.Mvc;
using Cibertec.UnitOfWork;
using Cibertec.Models;

namespace Cibertec.WebApi.Controllers
{
    [Route("api/Customer")]
    public class CustomerController : BaseController
    {
        public CustomerController(IUnitOfWork unit) : base(unit)
        {
        }

        public IActionResult GetList()
        {
            return Ok(_unit.Customers.GetList());
        }

        [HttpGet]
```

```
        [Route("{id:int}")]
        public IActionResult GetById(int id)
        {
            return Ok(_unit.Customers.GetById(id));
        }

        [HttpPost]
        public IActionResult Post([FromBody] Customer customer)
        {
            if (ModelState.IsValid)
                return Ok(_unit.Customers.Insert(customer));

            return BadRequest(ModelState);
        }

        [HttpPut]
        public IActionResult Put([FromBody] Customer customer)
        {
            if (ModelState.IsValid && _unit.Customers.Update(customer))
                return Ok(new { Message = "The customer is updated" });

            return BadRequest(ModelState);
        }

        [HttpDelete]
        [Route("{id}")]
        public IActionResult Delete(int? id)
        {
            if (id.HasValue && id.Value > 0)
                return Ok(_unit.Customers.Delete(new Customer { Id =
id.Value }));

            return BadRequest(new { Message = "Incorrect data." });
        }

        [HttpGet]
        [Route("count")]
        public IActionResult GetCount()
        {
            return Ok(_unit.Customers.Count());
        }
        [HttpGet]
        [Route("list/{page}/{rows}")]
        public IActionResult GetList(int page, int rows)
        {
            var startRecord = ((page - 1) * rows) + 1;
            var endRecord = page * rows;
            return Ok(_unit.Customers.PagedList(startRecord, endRecord));
        }
    }
}
```

7. Procedemos a agregar nuestra cadena de conexión en el fichero "**appsettings.json**"

```
"ConnectionStrings": {
    "Northwind": "Server=.;Database=Northwind_Lite;
Trusted_Connection=True;MultipleActiveResultSets=True"
  },
```

(Guiarse por la imagen)



```
{
  "ConnectionStrings": {
    "Northwind": "Server=.;Database=Northwind_Lite; Trusted_Connection=True;MultipleActiveResultSets=True"
  },
  "Logging": {
    "IncludeScopes": false,
    "Debug": {
      "LogLevel": {
        "Default": "Warning"
      }
    },
    "Console": {
      "LogLevel": {
        "Default": "Warning"
      }
    }
  }
}
```

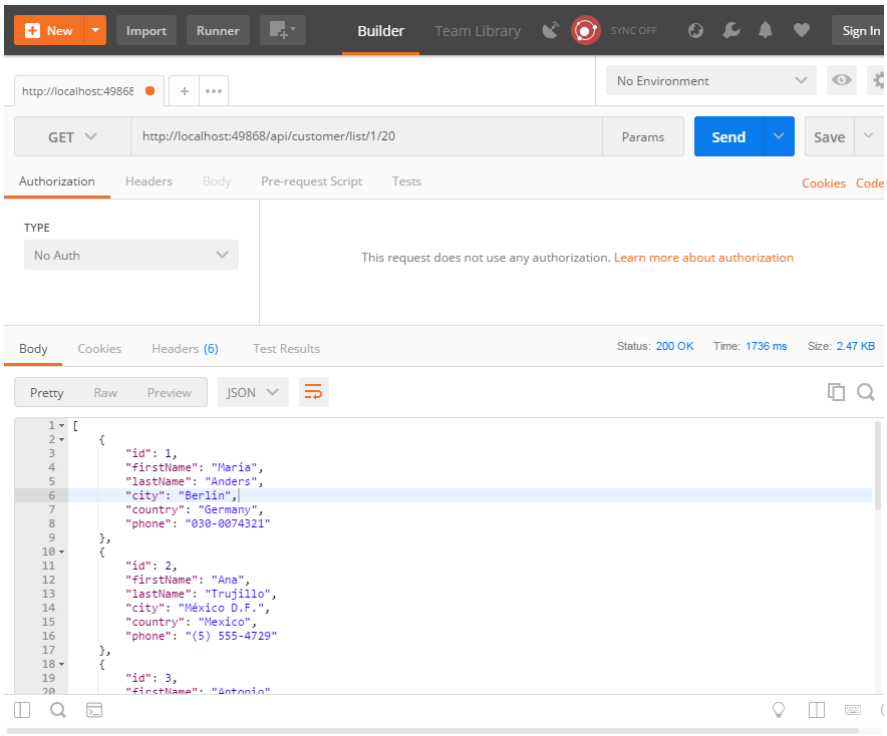
8. Ahora procedemos a realizar la inyección de dependencias usando el propio servicio de ASP NET Core.

   (1) Abrir el "**Startup.cs**" y ubicarse el método "**ConfigureServices**":

   (2) Agregamos la siguientes líneas de código:
   `services.AddSingleton<IUnitOfWork>(option => new NorthwindUnitOfWork(Configuration.GetConnectionString("Northwind")));`
   (Ver Imagen)

```
namespace Cibertec.WebApi
{
    2 references | Cesar Velarde, 1 hour ago | 1 author, 1 change
    public class Startup
    {
        0 references | Cesar Velarde, 1 hour ago | 1 author, 1 change | 0 exceptions
        public Startup(IConfiguration configuration)...

        2 references | Cesar Velarde, 1 hour ago | 1 author, 1 change | 0 exceptions
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 references | Cesar Velarde, 1 hour ago | 1 author, 1 change | 0 exceptions
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddSingleton<IUnitOfWork>(option => new NorthwindUnitOfWork(Configuration.GetConnectionString("Northwind")));
            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references | Cesar Velarde, 1 hour ago | 1 author, 1 change | 0 exceptions
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseMvc();
        }
    }
}
```

9. Si ejecutamos la aplicación podremos probar desde Postman que el endpoint funciona adecuadamente:



10. Proceda a probar los demás endpoints.

IV. EVALUACIÓN

1. ¿Qué diferencia a .NET Core del .NET Framework?
2. ¿Por qué usar Asp Net Core?
3. ¿Cuáles son los beneficios de una Web Api con ASP NET Core?
4. ¿Dónde está el web.config?