**CREAR Y CONSUMIR WEB API ASP**

<https://vbpuntonet.blogspot.com/2018/06/api-rest-creacion-de-un-api-rest-crud.html>

Qué es un Rest?

Es una arquitectura que se apoya en el estándar HTTP, permite crear aplicaciones y servicios que pueden ser usadas por cualquier dispositivo o cliente que utilice HTTP, algunas alternativas son SOAP y XML.

Los datos obtenidos pueden se XML o Json que suele ser más rápido.

Las operaciones más importantes en cualquier sistema Rest (CRUD) son:

GET (Leer y consultar los registros)

POST (Crear nuevos registros)

PUT (Editar y modificar los registros)

DELETE (Eliminar los registros)

Las ventajas de usar protocolo Rest es separar totalmente la interfaz de usuario del servidor y el almacenamiento de datos, esto contribuye a una mejora en la portabilidad de la plataforma y un aumento en la escalabilidad, así como aumentar la seguridad del mismo.

**Creamos un proyecto nuevo Web ASP NET Framework**

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Una vez creado el proyecto anexaremos un nuevo proyecto donde realizaremos la conexión a la base de datos.**

Botón derecho sobre el Solución “MiWebApi” Agregar nuevo proyeto:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Agregamos biblioteca de clases .NET Framework

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Se crea el proyecto ConectarDatos:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Eliminar class1

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Usaremos la base de datos UsuariosDB:**

**En sql Server**

Le damos en adjuntar base de datos

Texto

Descripción generada automáticamente

En agregar- ubicamos el archivo UsuariosDB.mdf

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Le damos en aceptar:

**NOTA:**

**Si no tenemos el archivo .ldf**

**Da el error de: No se encontró el archivo de registro .ldf**

**Lo podemos omitir para que se adjunte le base de datos**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tenemos montada la base de datos:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Conexión a la base de datos:**

**Adicionamos una nuevo Ítem en el nuevo proyecto.**

Captura de pantalla de un celular

Descripción generada automáticamente

El tipo será Visual C#, Data, ADO.NET Entity Data Model, tecleamos el nombre y lo adicionamos.

UsuariosDatosModel

Captura de pantalla de computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Buscamos las credenciales del servidor y las agregamos:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Agregamos las credenciales:**

**Contraseña: Sqlexp**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Le damos en siguiente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

siguiente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Seleccionamos las tablas a utilizar

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Finalizamos con la conexión a datos.

Una vez finalizado se visualizara el diagrama con la tabla de la base de datos donde se indica que nuestro campo primario es int\_id.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Ingresaremos al archivo app.Config, seleccionamos y copiamos la cadena de conexión a la base de datos.**

**Captura de pantalla de computadora

Descripción generada automáticamente**

Editamos el archivo web.Config de nuestro proyecto principal y pegamos la cadena de conexión que hemos copiado de app.Config.

**Texto

Descripción generada automáticamente**

**Agregar referencia a la base de datos:**

En nuestro proyecto principal agregamos una Referencia.

MiWebApi-Referencias-clic derecho-agregar Referencia -proyectos-ConectarDatos

aceptamos

Captura de pantalla de computadora

Descripción generada automáticamente

La nueva referencia se adicionará en el apartado References.

Captura de pantalla de un celular

Descripción generada automáticamente

**Agregar controladores:**

En nuestro proyecto principal agregamos un Controller.

Seleccionaremos el tipo Web Api 2 Controller-Empty.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**En caso de visualizar el siguiente error.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Seleccionar Build/Rebulid Solution.

Recompilamos la solución

Captura de pantalla de un celular

Descripción generada automáticamente

**Agregar de nuevo el controlador:**

Agregamos un controlador web API2 – ne blanco

Interfaz de usuario gráfica, Texto

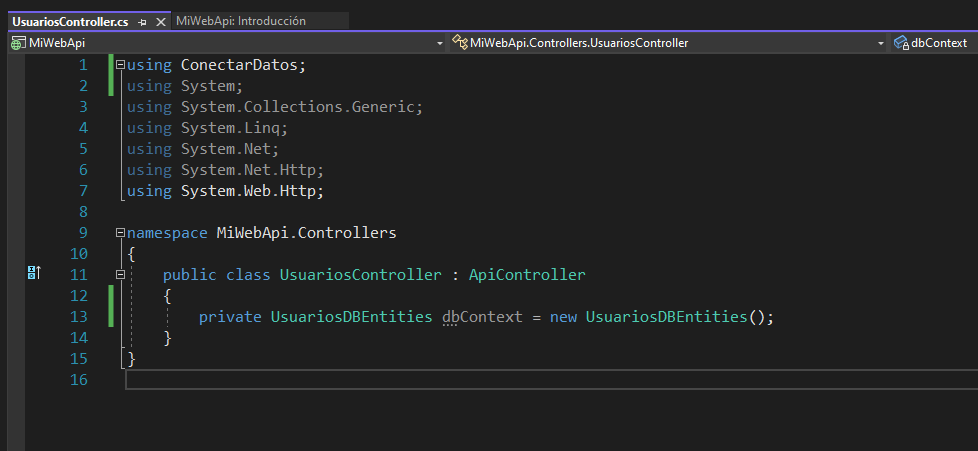
Descripción generada automáticamente

Lo llamamos: UsuariosController

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

**Agregaremos la referencia de la conexión de la base datos y teclearemos la siguiente línea para el DbContext.**

****

**Los datos los tomaremos de la referencia donde realizamos la conexión a la base de datos.**

**Captura de pantalla con letras y números

Descripción generada automáticamente**

**Ahora debemos agregar el paquete EntityFramework, seleccionamos nuestro proyecto principal, Manager NuGet Packages…**

**Clic derecho**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**Instalamos:**

**EntityFramework**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Instalamos:**

**EntityFramework.SqlServerCompact**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**Los nombres de las tablas los podemos ver haciendo doble clic sobre la referencia ConectarDatos:**

**Texto

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**UsuariosController**

En el controller agregamos un método para consultar todos los usuarios de la base de datos:

Texto

Descripción generada automáticamente

**Ejecutaremos nuestra aplicación y seleccionaremos el apartado API.**

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

De momento tenemos el metodo Get para usuarios:

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Sitio web

Descripción generada automáticamente**

Podemos ver los campos y tipos que podemos utilizar

**Interfaz de usuario gráfica

Descripción generada automáticamente**

también vemos los formatos de la repuesta

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Ver en el navegador la respuesta de la api:**

Consumir la api:

[**https://localhost:44355/api/Usuarios**](https://localhost:44355/api/Usuarios)

nos muestra el resultado en formato XML:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Salida en JSON:**

**La salida de la información esta en formato XML para generarla en formato Json debemos editar el archivo WebApiConfig.cs que se ubica en el proyecto principal.**

Teclearemos el siguiente código que realizar la salida en formato Json y remueve la salida en formato XML.

**App\_Start-WebApiConfig**

Texto

Descripción generada automáticamente

**WebApiConfig salida en json:**

Texto

Descripción generada automáticamente

Código:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

namespace MiWebApi

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Modifica para que la repuesta de la api sea en formato Json remueve la salida en Xml:

var json = config.Formatters.JsonFormatter;

json.SerializerSettings.PreserveReferencesHandling =

Newtonsoft.Json.PreserveReferencesHandling.Objects;

config.Formatters.Remove(config.Formatters.XmlFormatter);

// Rutas de Web API

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

}

**Vemos la salida en formato json:**

<https://localhost:44355/api/Usuarios>

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

<https://vbpuntonet.blogspot.com/2018/06/api-rest-creacion-de-un-api-rest-crud.html>

**CREAR CRUD PARA LA API USURIOS:**

using ConectarDatos;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using System.Data.Entity;

namespace MiWebApi.Controllers

{

public class UsuariosController : ApiController

{

private UsuariosDBEntities dbContext = new UsuariosDBEntities();

//Mostrar todos los registros (api/usuarios)

[HttpGet]

public IEnumerable<usuario> Get()

{

using (UsuariosDBEntities db = new UsuariosDBEntities())

{

return db.usuarios.ToList();

}

}

//Obtener un registro por su Id:

[HttpGet]

public usuario Get(int id)

{

using(UsuariosDBEntities db = new UsuariosDBEntities()){

return db.usuarios.FirstOrDefault(u => u.int\_id == id);

}

}

//Crear nuevos Registros

[HttpPost]

public IHttpActionResult AgregaUsuario([FromBody]usuario user)

{

if (ModelState.IsValid)

{

dbContext.usuarios.Add(user);

dbContext.SaveChanges();

return Ok(user);

}

else

{

return BadRequest();

}

}

//Actualizar Registros

[HttpPut]

public IHttpActionResult ActulizarUsuario(int id, [FromBody]usuario user)

{

if (ModelState.IsValid)

{

var UsuarioExiste = dbContext.usuarios.Count(c => c.int\_id == id) > 0;

if (UsuarioExiste)

{

dbContext.Entry(user).State = EntityState.Modified;

dbContext.SaveChanges();

return Ok();

}

else { return NotFound(); }

}

else { return BadRequest(); }

}

//Eliminar Registros

[HttpDelete]

public IHttpActionResult EliminarUsuario(int id)

{

var user = dbContext.usuarios.Find(id);

if(user != null)

{

dbContext.usuarios.Remove(user);

dbContext.SaveChanges();

return Ok(user);

}

else { return NotFound(); }

}

}

}

**Corremos el proyecto y vemos que se crearon todos los métodos para la api:**

<https://localhost:44355/Help>

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Consumir la api:**

Ver el usuario con id 1:

<https://localhost:44355/api/Usuarios/1>

Texto, Carta

Descripción generada automáticamente

Ver todos los usuarios:

<https://localhost:44355/api/Usuarios/>

Texto

Descripción generada automáticamente

**Postman:**

Para utilizar los métodos de Post, Put, Delete. Usaremos Postman para consumir la web api.

Descargamos postman:

<https://www.postman.com/downloads/?utm_source=postman-home>

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

User:

[eliezzer.jmz@gmail.com](mailto:eliezzer.jmz@gmail.com)

Post+-20\_22\*\*

**Entramos a Workspaces My Workspace**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Creamos una nueva colección para la API USUARIOS CON ASP.NET:**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Realizamos el GET para todos los usuarios y lo guardamos en la colección que acabamos de crear API USUARIOS CON ASP.NET:**

Le damos en save

Interfaz de usuario gráfica, Aplicación

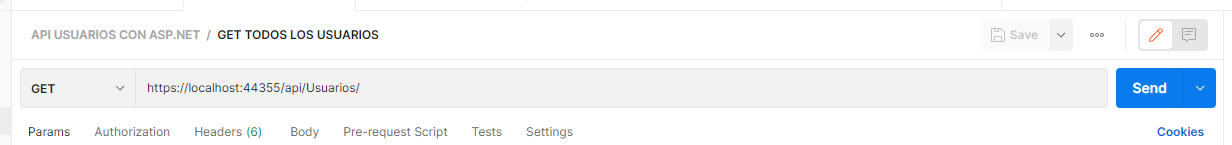
Descripción generada automáticamente con confianza media

**Se guardo en la colección el GET A TODOS LOS USUARIOS:**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

ENVIAMOS LA CONSULTA Y NOS DEVUELVE EL JOSN CON TODOS LOS USUARIOS:



Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

**GET**

**OBTENER USUARIO POR MEDIO DE SU ID:**

Creamos new HTTP Request

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

**Enviamos la petición y le damos a guardar como GET USURIO POR ID**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**POST**

**CREAR UN NUEVO REGISTRO:**

En la api nos indica el formato a utilizar:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Le damos en new HTTP Request

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Usamos <https://localhost:44355/api/Usuarios> con el metodo post:

**METODO POST (Agregar un nuevo registro)**

Seleccionaremos el método POST, Body, Raw y teclearemos los nuevo datos en formato **Json, al presionar SEND el nuevo registro se visualizara en la parte inferior.**

**POST**

**Enviamos un nuevo registro POST**

**Body—Raw--JSON**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**REGISTRO ENVIADO EN FORMATO JSON:**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

RESPUESTA:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**GET POR ID**

**VERRIFICAMOS QUE SE HAYA INSERTADO EL REGISTRO HACEMOS UN GET CON EL int\_id: 34**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

**DELETE**

**ENVIAMOS EL ID:**

<https://localhost:44355/api/Usuarios/35>

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Nos muestra el registro por id eliminado:

Texto

Descripción generada automáticamente

* **CONSUMIER AL WEB API DESDE OTRA APLICACIÓN:**

<https://vbpuntonet.blogspot.com/2018/06/api-rest-consumiendo-un-api-rest-en-c.html>

creamos un proyecto nuevo web asp framework con MVC

Captura de pantalla de un celular

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

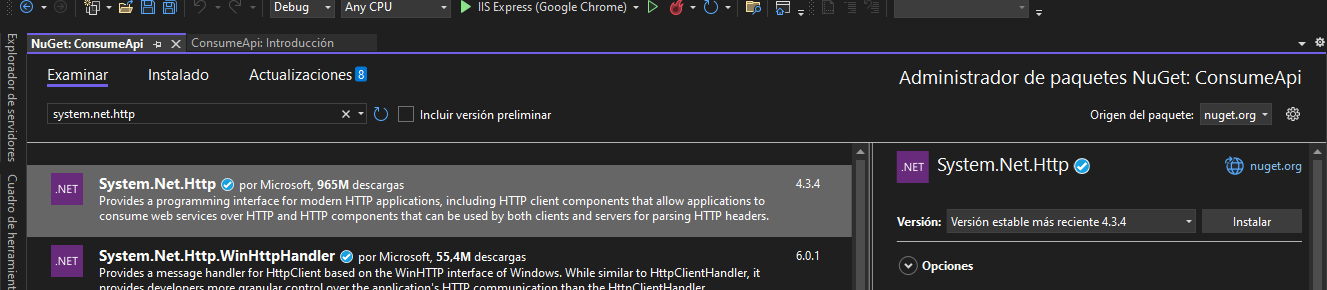
Peoyecto vacio

Marcamos MVC

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Instalamos los siguientes paquetes:**



Captura de pantalla de computadora

Descripción generada automáticamente

**Agregar el modelo basado en el json de la api:**

**Nos apoyamos en esta web:**

[**https://json2csharp.com/**](https://json2csharp.com/)

**copias el json que devuelve la api suficiente con un registro:**

[**https://localhost:44355/api/Usuarios/1**](https://localhost:44355/api/Usuarios/1)

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Listo ya tenemos nuestro modelo:**

**Solo eliminamos la propiedad jsonProperty y el campo string id, que no es parte de nuestro modelo original:**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Agregar el modelo para Usuarios:**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Copiamos pegamos las propiedades que nos genero json2c#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace ConsumeApi.Models

{

public class Usuarios

{

public int int\_id { get; set; }

public string str\_status { get; set; }

public DateTime dat\_fecha { get; set; }

public string str\_nombre { get; set; }

public string str\_direccion { get; set; }

public string str\_departamento { get; set; }

public int int\_telefono { get; set; }

}

}

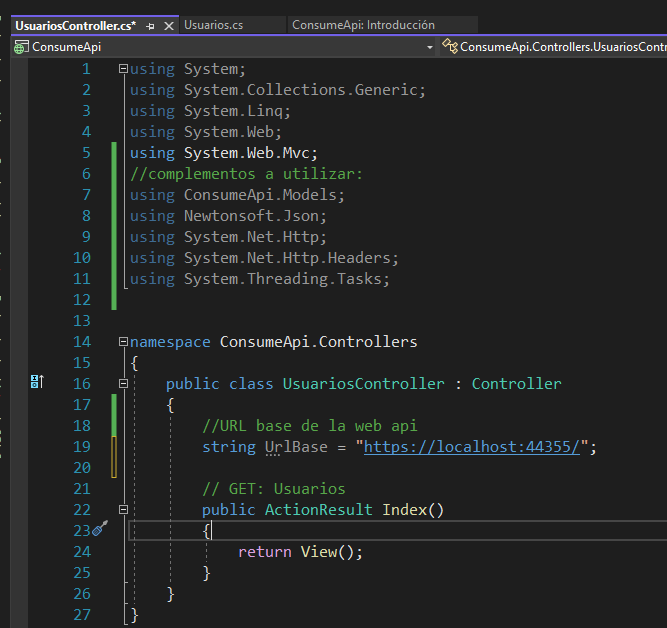
Agregamos un controlador en blanco UsuariosController:

**Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente**

**Controller Usuarios:**

Indicamos los complementos a utilizar y la UrlBase:



**El código:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

//complementos a utilizar:

using ConsumeApi.Models;

using Newtonsoft.Json;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Threading.Tasks;

namespace ConsumeApi.Controllers

{

public class UsuariosController : Controller

{

//URL base de la web api

string UrlBase = "https://localhost:44355/";

// GET: Usuarios

public ActionResult Index()

{

return View();

}

}

}

**Método GET**

**index**

**para retornar un lista de todos los usuarios**

Consumir la dirección de la api <https://localhost:44355/api/usuarios>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

//complementos a utilizar:

using ConsumeApi.Models;

using Newtonsoft.Json;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Threading.Tasks;

namespace ConsumeApi.Controllers

{

public class UsuariosController : Controller

{

//URL base de la web api, variable global

string UrlBase = "https://localhost:44355/";

// GET: Usuarios

//Usa un metodo asyncrono y una tarea de tipo AcntionResult

public async Task<ActionResult> Index()

{

//creamos una lista donde guardaremos la informacion de los usuarios

List<Usuarios> InfoUsuarios = new List<Usuarios>();

//realizamos la conexion utilizanod HttpClient()

using (var client = new HttpClient())

{

//URL base de la api

client.BaseAddress = new Uri(UrlBase);

client.DefaultRequestHeaders.Clear();

//indicamos el encabezado de la api que seran json

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

//hacemos una peticion await a la parte de la api que lista a tosos ls usuarios usando Httpclient

HttpResponseMessage Respuesta = await client.GetAsync("api/usuarios");

//validamos si obtenemos una Respuesta con exito

if (Respuesta.IsSuccessStatusCode)

{

//si Respuesta = true entra y asigna los datos a la varaible UsuariosRespuesta

var UsuariosRespuesta = Respuesta.Content.ReadAsStringAsync().Result;

//Deserializar el api y asigna los datos de UsuariosRespuesta a la lista InfoUsuarios

InfoUsuarios = JsonConvert.DeserializeObject<List<Usuarios>>(UsuariosRespuesta);

}

//Retorna la lista de todos los usuarios

return View(InfoUsuarios);

}

}

}

}

**VISTA INDEX QUE MUESTRA TODOS LOS USUARIOS:**

Interfaz de usuario gráfica

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**CORREMOS LA WEB API Y CORREMOS NUESTRO PROYECTO:**

Estamos consumiendo la web api

**Tabla

Descripción generada automáticamente**

**instalar bootrap y confirugar los bundles y render para que se vea bonita la vista :**

* Instalar Bootstrap

**Captura de pantalla con la imagen de una pantalla

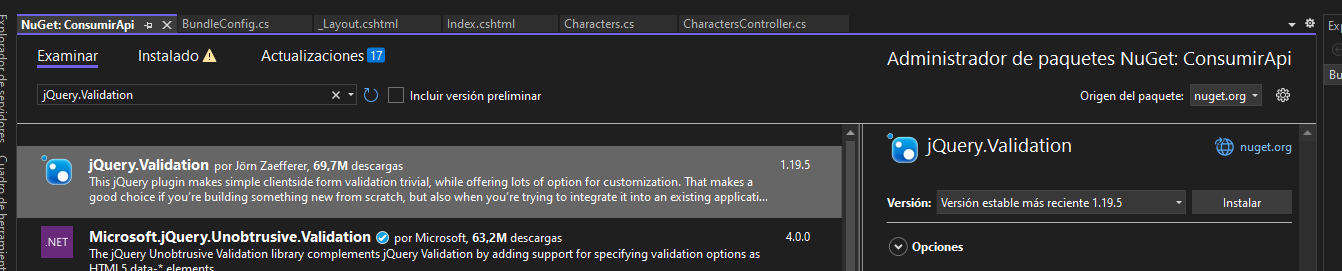
Descripción generada automáticamente**

* Instalar Microsoft.AspNet.Web.Optimizatio

Captura de pantalla de computadora

Descripción generada automáticamente

* Instalar jQuery.Validation



* Instalar ModernizrCaptura de pantalla de un celular

  Descripción generada automáticamente
* Agregar clase BundleConfig
* Texto

  Descripción generada automáticamente

Texto

Descripción generada automáticamente

* **Modificar el archivo BundleConfig.cs**

Usamos System.Web.Optimization; y el //Metodo para registrar los Bundles

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Optimization;

namespace ConsumeApi.App\_Start

{

public class BundleConfig

{

//Metodo para registrar los Bundles

public static void RegisterBundles(BundleCollection bundles)

{

//Nombre del Bundle: "~/bundles/jquery"

//Ubicacion del archivo: "~/Scripts/jquery-{version}.js"

//{version}: version tomara la version del archivo en este caso jquery-3.0.0.js

//Nota: los Bundles utilizan los archivos .min cuando estan en produccion

//Podemos agregar cunatas ubicacines necesitemos y utilizar solo un nombre para el Bundle

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

// Utilice la versión de desarrollo de Modernizr para desarrollar y obtener información sobre los formularios. De esta manera estará

// para la producción, use la herramienta de compilación disponible en https://modernizr.com para seleccionar solo las pruebas que necesite.

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js"));

//StyleBundle

//ubicacion: carpeta Content

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"));

}

}

}

* **WebConfig: agregar <add namespace="System.Web.Optimization"/>**

En el web config de las vistas

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Texto

Descripción generada automáticamente

* **Modificar el archivo Global.asax**

Agregamos las dependencias y BundleConfig.RegisterBundles(BundleTable.Bundles);

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

**//agregamos dependencias**

**using ConsumeApi.App\_Start;**

**using System.Web.Optimization;**

namespace ConsumeApi

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

RouteConfig.RegisterRoutes(RouteTable.Routes);

**BundleConfig.RegisterBundles(BundleTable.Bundles);**

}

}

}

**Actualizar el \_layout**

**Texto

Descripción generada automáticamente**

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - Mi aplicación ASP.NET</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">

@Html.ActionLink("Inicio", "Index", "Usuarios", new { area = "" }, new { @class = "navbar-brand" })

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="navbar-collapse collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

@Html.ActionLink("Inicio", "Index", "Usuarios", null, new { @class = "nav-link" })

</li>

<li class="nav-item">

@Html.ActionLink("Acerca de", "About", "Home", null, new { @class = "nav-link" })

</li>

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-expanded="false">

Dropdown

</a>

<div class="dropdown-menu" aria-labelledby="navbarDropdown">

@Html.ActionLink("Contacto", "Contact", "Home", null, new { @class = "dropdown-item" })

<a class="dropdown-item" href="#">Another action</a>

<div class="dropdown-divider"></div>

<a class="dropdown-item" href="#">Something else here</a>

</div>

</li>

<li class="nav-item">

<a class="nav-link disabled">Disabled</a>

</li>

</ul>

@{

//Html.Partial("\_LoginPartial")

}

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" type="search" placeholder="Search" aria-label="Search">

<button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>

</form>

</div>

</nav>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - Mi aplicación ASP.NET</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

**Listo corremos la api y el proyecto:**

Estamos consumiendo la web api y configuramos Bootstrap y los bundle, render y \_layout

**Una captura de pantalla de una computadora

Descripción generada automáticamente**

**METODO POST**

**CREATE**

//Crea nuevo usuario

//carga el formulario y llama al metodo HttpPost Create

public ActionResult Create()

{

return View();

}

//POST Usuarios

[HttpPost]

public ActionResult Create(Usuarios usuarios)

{

string create = "api/usuarios";

//Usamos una conexion hacia la api

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(UrlBase + create);

//aqui indicamos al metodo post el registro de tipo usuarios que vamos a crear

var postTask = client.PostAsJsonAsync<Usuarios>("usuarios", usuarios);

postTask.Wait();

var result = postTask.Result;

if (result.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

}

ModelState.AddModelError(String.Empty, "Error, al crear un registro.");

return View(usuarios);

}

**Crear la vista:**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Ingresamos los datos en el formulario:

**Interfaz de usuario gráfica

Descripción generada automáticamente**

Se ha creado el registro

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**METODO POST**

**EDITAR:**

//EDIT

//Primero realizamos una busqueda del registro a editar por medio de su ID, usamos un GET

public ActionResult Edit(int id)

{

string api = "api/usuarios/";

Usuarios usuarios = new Usuarios();

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(UrlBase);

//HTTP GET

var responseTask = client.GetAsync(api + id.ToString());

responseTask.Wait();

var result = responseTask.Result;

if (result.IsSuccessStatusCode)

{

var readTask = result.Content.ReadAsAsync<Usuarios>();

readTask.Wait();

usuarios = readTask.Result;

}

}

return View(usuarios);

}

[HttpPost]

public ActionResult Edit(Usuarios usuarios)

{

string api = "api/usuarios/";

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(UrlBase);

//HTTP POST

var putTask = client.PutAsJsonAsync(api + usuarios.int\_id.ToString(), usuarios);

putTask.Wait();

var result = putTask.Result;

if (result.IsSuccessStatusCode)

{

//si la actualizacion fue correcta nos redirecciona al index

return RedirectToAction("Index");

}

}

return View(usuarios);

}

**VISTA:**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

**INDEX:**

Debemos indicar en el index que le enviaremos un parámetro :

<**td>**

**<!--le envia el id al metodo edit para que realice una busqueda por id-->**

**@Html.ActionLink("Edit", "Edit", new { id=item.int\_id })**

**Podemos editar los usuarios:**

Interfaz de usuario gráfica

Descripción generada automáticamente

POST

DELETE:

//DELETE

//BUSCAR REGISTRO A ELIMINAR POR ID

public ActionResult Delete(int id)

{

//api

string api = "api/usuarios/";

//modelo

Usuarios usuarios = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(UrlBase);

//HTTP GET

var responseTask = client.GetAsync(api + id.ToString());

responseTask.Wait();

var result = responseTask.Result;

if (result.IsSuccessStatusCode)

{

var readTask = result.Content.ReadAsAsync<Usuarios>();

readTask.Wait();

usuarios = readTask.Result;

}

}

return View(usuarios);

}

[HttpPost]

public ActionResult Delete(Usuarios usuarios, int id)

{

//api

string api = "api/usuarios/";

using (var cliente = new HttpClient())

{

cliente.BaseAddress = new Uri(UrlBase);

//HTTP DELETE

var deleteTask = cliente.DeleteAsync(api + id.ToString());

deleteTask.Wait();

var result = deleteTask.Result;

if (result.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

}

return View(usuarios);

}

VIEW

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

INDEX:

**En el index indicamos en el actionlink que enviaremos como parámetro el int\_id**

<td>

<!--le envia el id al metodo edit para que realice una busqueda por id-->

@Html.ActionLink("Edit", "Edit", new { id=item.int\_id }) |

@Html.ActionLink("Details", "Details", new { /\* id=item.PrimaryKey \*/ }) |

**@Html.ActionLink("Delete", "Delete", new { id=item.int\_id})**

</td>

Corremos el proyecto:

Nos muestra el detalle del registro a eliminar:

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

**PATCH:**

<https://geeks.ms/jorge/2020/12/23/json-patch-en-asp-net-core-5-web-api/#:~:text=JSON%20Patch%20es%20un%20formato,array%20o%20matriz%20de%20operaciones>.

Actualizaciones parciales, la api recibe un json, únicamente con los capos que se van a actualizar, por lo que no es necesario enviar todo el registro.

Mas ejemplos:

<https://gavilanch.wordpress.com/2019/04/05/asp-net-core-2-2-actualizaciones-parciales-con-http-patch-json-patch/>

<https://www.youtube.com/watch?v=EOOJRyQuTdo>

CONTROLLER:

//Instalar el complemento:

using Microsoft.AspNetCore.JsonPatch;

//Actuaizacion Parcial

//recibe un json unicamente con los campos a actualizar, no todo el registro

[HttpPatch]

public IHttpActionResult ActualizacionParcialUsuario(int id, [FromBody] JsonPatchDocument<usuario> patchUser)

{

if (patchUser != null)

{

var userDeLaDB = dbContext.usuarios.FirstOrDefault(x => x.int\_id == id);

if (userDeLaDB != null)

{

patchUser.ApplyTo(userDeLaDB);

dbContext.SaveChanges();

return Ok(userDeLaDB);

}

}

return NotFound();

}

**Formato de un JSON Patch**

De acuerdo al ejemplo de **JSON Patch** anterior, tenemos que tener en cuenta que un **JSON Patch** estará formado por tantas operaciones atómicas o individuales como consideremos, y que el formato de un **JSON Patch** está formado por las siguientes partes generales:

**op**  
Para indicar el tipo de operación.

Las posibles operaciones son:

* add
* copy
* move
* remove
* replace
* test

**path**  
Para indicar el elemento a actualizar.  
Si nuestro objeto tiene niveles, dividiremos esos niveles con el carácter /  
Imaginemos que tenemos una clase **Person**, que dentro de ésta tiene una clase **Address**, y esa clase una propiedad **Street**.  
El **path** en este caso será algo similar a **/address/street**

**value**  
Para indicar el nuevo valor del elemento a actualizar.

Enviamos un json indicando el tipo de operación. El path en donde indicamos el nombre del campo a actualizar,

Y el nuevo valor que tendrá.

[   {"op" : "replace", "path" : "/str\_nombre", "value" : "Marina"},

    {"op" : "replace", "path" : "/str\_status", "value" : "Activo"}

]

Ejemplo actualizar únicamente el nombre y el status

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**CORS:**

Habilitar cors para que la api pueda ser consumida desde otro proyecto:

**Agregamos config.EnableCors();**

En App\_Start-WebApiConfig

Texto

Descripción generada automáticamente

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

**//Configuracion y servicios de API Web habilitar CORS**

**config.EnableCors();**

// Modifica para que la repuesta de la api sea en formato Json remueve la salida en Xml:

var json = config.Formatters.JsonFormatter;

json.SerializerSettings.PreserveReferencesHandling =

Newtonsoft.Json.PreserveReferencesHandling.Objects;

config.Formatters.Remove(config.Formatters.XmlFormatter);

// Rutas de Web API

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

**En el controller :**

Agregamos arriba de la clase:

[EnableCors(origins: "http://localhost:4200", headers:"\*", methods:"\*")]

Indicamos las url que tendrán acceso, podemos agregar mas separándolas por coma,

namespace MiWebApi.Controllers

{

**////aquí indicamos los clientes que tendrán acceso a la api**

**[EnableCors(origins: "http://localhost:4200", headers:"\*", methods:"\*")]**

public class UsuariosController : ApiController

{

private UsuariosDBEntities dbContext = new UsuariosDBEntities();

//Mostrar todos los registros (api/usuarios)

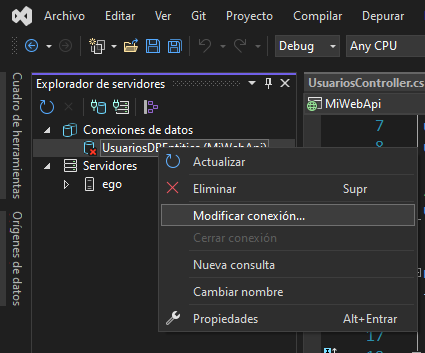
[HttpGet]

public IEnumerable<usuario> Get()

{

**ACTUALIZAR EL SERVIDOR DE LA BASE DE DATOS EN LA WEB API:**

En explorador de servidores-clic derecho-modificar:

****

Ingresamos los datos del nuevo servidor que tiene la base de datos:

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Y seleccionamos la base de datos:

Contraseña: Sqlexp

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Probamos la conexión:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto, Correo electrónico

Descripción generada automáticamente

Aceptamos:

**CONSUMIR EL METODO DE LA WEB API QUE REALIZA LA BUSQUEDA DEL USUARIO POR MEDIO DE SU NOMNRE:**

* le debemos indicar el nomnbre dela variable que tiene en la api

string api = "api/Usuarios?nomb=";

Creamos el método en el controlador:

//GET USUARIO POR NOMBRE

//realizamos una busqueda del registro por medio del nombre

public async Task<ActionResult> UserForNameList(string nomb)

{

//le debemos indicar el nomnbre dela variable que tiene en la api

string api = "api/Usuarios?nomb=";

//creamos una lista donde guardaremos la informacion de los usuarios

List<Usuarios> ListUsuarios = new List<Usuarios>();

//realizamos la conexion utilizanod HttpClient()

using (var client = new HttpClient())

{

//URL base de la api

client.BaseAddress = new Uri(UrlBase);

client.DefaultRequestHeaders.Clear();

//indicamos el encabezado de la api que seran json

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

//hacemos una peticion await a la parte de la api que lista a los usuarios usando Httpclient

HttpResponseMessage Respuesta = await client.GetAsync(api + nomb);

//validamos si obtenemos una Respuesta con exito

if (Respuesta.IsSuccessStatusCode)

{

//si Respuesta = true entra y asigna los datos a la varaible UsuariosRespuesta

var UsuariosRespuesta = Respuesta.Content.ReadAsStringAsync().Result;

//Deserializar el api y asigna los datos de UsuariosRespuesta a la lista InfoUsuarios

ListUsuarios = JsonConvert.DeserializeObject<List<Usuarios>>(UsuariosRespuesta);

}

//Retorna la lista de los usuarios

return View(ListUsuarios);

}

}

Creamos la vista:

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

**ORDER BY POR NOMBRE:**

//GET ORDERBY POR NOMBRE

public async Task<ActionResult> NameOrderBy()

{

//le debemos indicar la path de la api

string api = "api/OrderByNombre";

//creamos una lista donde guardaremos la informacion de los usuarios

List<Usuarios> ListUsuarios = new List<Usuarios>();

//realizamos la conexion utilizanod HttpClient()

using (var client = new HttpClient())

{

//URL base de la api

client.BaseAddress = new Uri(UrlBase);

client.DefaultRequestHeaders.Clear();

//indicamos el encabezado de la api que seran json

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

//hacemos una peticion await a la api con el metdo OrderByNombre usando Httpclient

HttpResponseMessage Respuesta = await client.GetAsync(api);

//validamos si obtenemos una Respuesta con exito

if (Respuesta.IsSuccessStatusCode)

{

//si Respuesta = true entra y asigna los datos a la varaible UsuariosRespuesta

var UsuariosRespuesta = Respuesta.Content.ReadAsStringAsync().Result;

//Deserializar el api y asigna los datos de UsuariosRespuesta a la lista InfoUsuarios

ListUsuarios = JsonConvert.DeserializeObject<List<Usuarios>>(UsuariosRespuesta);

}

//Retorna la lista de los usuarios

return View(ListUsuarios);

}

}

Creamos la vista con plantilla de lista:

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

**Búsqueda por dirección:**

**Usamos el método post para recibir los parámetros en el controller:**

//GET BUSCAR TEXTO CONTENIDO EN DIRECCION

[HttpPost]

public async Task<ActionResult> ContieneDireccion(string dir)

{

//le debemos indicar la variable de la api

string api = "api/Usuarios?dir=";

//creamos una lista donde guardaremos la informacion de los usuarios

List<Usuarios> ListUsuariosDir = new List<Usuarios>();

//realizamos la conexion utilizanod HttpClient()

using (var client = new HttpClient())

{

//URL base de la api

client.BaseAddress = new Uri(UrlBase);

client.DefaultRequestHeaders.Clear();

//indicamos el encabezado de la api que seran json

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

//hacemos una peticion await al metodo de la api, usando Httpclient

HttpResponseMessage Respuesta = await client.GetAsync(api + dir);

//validamos si obtenemos una Respuesta con exito

if (Respuesta.IsSuccessStatusCode)

{

//si Respuesta = true entra y asigna los datos a la varaible UsuariosRespuesta

var UsuariosRespuesta = Respuesta.Content.ReadAsStringAsync().Result;

//Deserializar el api y asigna los datos de UsuariosRespuesta a la lista

ListUsuariosDir = JsonConvert.DeserializeObject<List<Usuarios>>(UsuariosRespuesta);

}

//Retorna la lista

return View(ListUsuariosDir);

}

}

**Llamamos al método desde la vista index y le pasamos la dirección a buscar**

**Usando un form con el metro post:**

<div>

<h4>

Buscar por Dirección:

</h4>

<p>llamamos al metodo ContieneDireccion(dir) y le pasamos la dirección a buscar usamos un form con el metodo post </p>

<!--CREAMO UN FORMAULARIO PARA ENVIAR LOS DATOS POR MEIO DE POST-->

<form action="/Usuarios/ContieneDireccion" method="post">

<label>Dirección a Buscar: </label>

<input type="text" id="dir" name="dir" placeholder="Imgrese una Dirección" />

<input type="submit" value="Enviar" />

</form>

</div>

Vista:

@model IEnumerable<ConsumeApi.Models.Usuarios>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.int\_id)

</th>

<th>

@Html.DisplayNameFor(model => model.str\_status)

</th>

<th>

@Html.DisplayNameFor(model => model.dat\_fecha)

</th>

<th>

@Html.DisplayNameFor(model => model.str\_nombre)

</th>

<th>

@Html.DisplayNameFor(model => model.str\_direccion)

</th>

<th>

@Html.DisplayNameFor(model => model.str\_departamento)

</th>

<th>

@Html.DisplayNameFor(model => model.int\_telefono)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.int\_id)

</td>

<td>

@Html.DisplayFor(modelItem => item.str\_status)

</td>

<td>

@Html.DisplayFor(modelItem => item.dat\_fecha)

</td>

<td>

@Html.DisplayFor(modelItem => item.str\_nombre)

</td>

<td>

@Html.DisplayFor(modelItem => item.str\_direccion)

</td>

<td>

@Html.DisplayFor(modelItem => item.str\_departamento)

</td>

<td>

@Html.DisplayFor(modelItem => item.int\_telefono)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { /\* id=item.PrimaryKey \*/ }) |

@Html.ActionLink("Details", "Details", new { /\* id=item.PrimaryKey \*/ }) |

@Html.ActionLink("Delete", "Delete", new { /\* id=item.PrimaryKey \*/ })

</td>

</tr>

}

</table>