Bien en esta ocasión traigo un API Rest construido en Net Con SQL Server.

El cual consiste en CRUD para una tabla llamada: Usuarios, que consultará todos los usuarios o un solo usuario, ingresará, actualizará y eliminará un usuario.

Vamos inicialmente a crear nuestra Base de Datos para probar el API REST, ésta BD está muy sencilla pues solo tiene una tabla con tres columnas pero por ahora es solo lo que requerimos para probar. La BD yo la llamé: ApiRest_DB, tú la puedes nombrar como quieras, pero deberás luego modificar la cadena de conexión, en el archivo de configuración: Webconfig. Crearemos una estructura llamada: Usuarios y tres procedimientos almacenados, para gestionar todo directamente en la BD , y agilizar las respuestas solicitadas al API, pues si bien el json es ligeramente más eficiente que el xml, todavía los servicios web pecan de ser algo lentos…

Una vez que hayas creado la BD, procederemos a crear ya el API en Net, yo trabajé con la versión Comunity 2019.

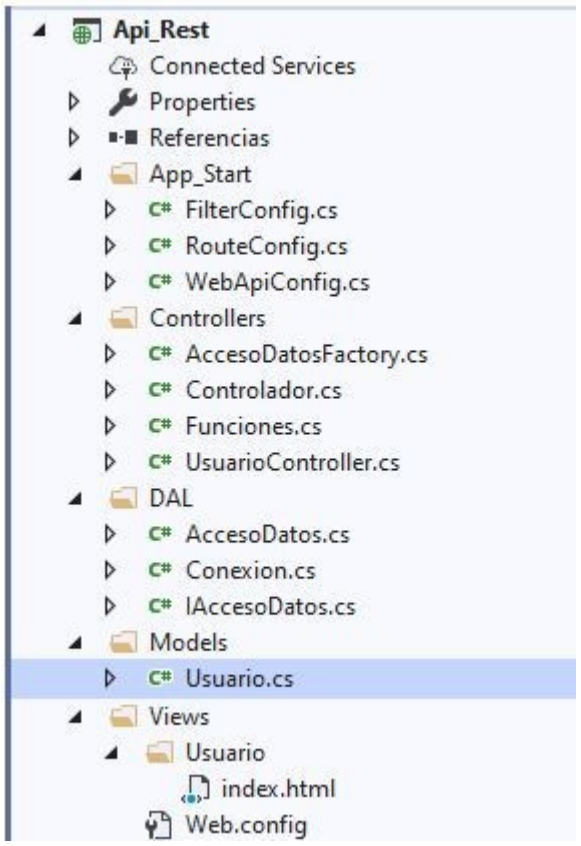
Le damos Crear un Proyecto y elegimos la opción: Aplicación web ASP.Net (.Net Framework)

Luego de darle un nombre a la aplicación, elegimos la opción API web

Realizado lo anterior, se nos crea una plantilla para construir APIS Rest, pero esta queda saturada de muchos archivos y carpetas que finalmente no son necesarias.

Bien ahora te detallo como quedó configurado el proyecto, para que se entienda la arquitectura y distribución de carpetas en el proyecto, yo utilicé MVC donde quedaron 4 directorios a saber:

Detalle Proyecto en Visual Studio



App_Start - Configuración del Proyecto: Es la capa que tiene la configuración inherente dentro del aplicativo, acá en este paquete quedan tres clases a saber:

- FilterConfig: Controla errores y filtra cualquier solicitud no válida.

- RouteConfig: Define reglas de navegación y la forma como han de llamarse los servicios.

- WebApiConfig: Es el encargado de definir la configuración y servicios del API web, así como sus rutas.

Controllers - Capa Lógica de Negocio: En esta alojamos toda la lógica propia del mundo que se pretende modelar, donde quedarán nuestros controladores. En realidad es una capa que sirve de enlace entre las vistas y los modelos (BO), respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de puente entre modelos y vistas, acá en este paquete quedan tres clases a saber:

- AccesoDatosFactory: Aquí emplearemos el patrón de Creación: Factory Method al cual le pasaremos como argumento la Interfaz IAccesoDatos, e invocaremos el método: CrearControlador alojado en la clase Funciones que nos devolverá, ya la clase AccesoDatos instanciada.

- Controlador: Como su nombre lo indica es el controlador manager general de todo el sistema. Todo tiene que pasar por esta clase.

- UsuarioController: Es el controlador del objeto usuario.

- Funciones: Esta es una clase transversal, desde donde se instancia el Factory Method.

DAL - Capa Acceso a Datos: Es la capa que contiene todos los mecanismos de acceso a nuestra BD, acá en este paquete quedan tres clases a saber:

- Conexion: Clase que conecta con el proveedor de Base de Datos-MySql.

- IAccesoDatos: Interfaz que solo expone los métodos que implementa el acceso a datos.

- AccesoDatos: Implementa la interfaz con todos sus métodos.

Models - Capa Objetos de Negocio: Acá queda alojado nuestro único objeto de negocio llamado: Usuarios, que es la clase que representa esta estructura en BD, y que solo tiene sus atributos con sus respectivos get y set.

Funcionamiento

El funcionamiento es simple: la capa de presentación pregunta a la BLL por algún objeto, ésta a su vez puede opcionalmente desarrollar alguna validación, y después llamar al DAL, para que esta conecte con la base de datos y le pregunte por un registro específico, cuando ese registro es encontrado, éste es retornado de la base de datos al DAL, quien empaqueta los datos de la base de datos en un objeto personalizado y lo retorna al BLL, para finalmente retornarlo a la capa de presentación, donde podrá ser visualizado por el cliente en formato json.

Tanto la capa lógica de negocio como la capa de acceso de datos consiguen una referencia a los objetos en el BO. Además, la capa de negocio consigue una referencia a la capa de acceso de datos para toda la interacción de datos.

Los objetos del negocio se colocan en una capa diferente para evitar referencias circulares entre la capa del negocio y la de datos.

Es importante destacar que en la DAL habrá una interface llamada IAccesoDatos, que será nuestra puerta de entrada, allí no hay implementación de ningún método, solo se exponen, quien use dicha interface es quien los debe implementar, para nuestro caso el DAL (AccesoDatos).

Nos apoyaremos en el patrón de creación: Factory Method que nos ayudará a la hora de instanciar la clase DAL, que debe implementar todos los métodos de la interface IAccesoDatos, lo que nos ahorrará trabajo ya que el conjunto de clases creadas pueden cambiar dinámicamente.

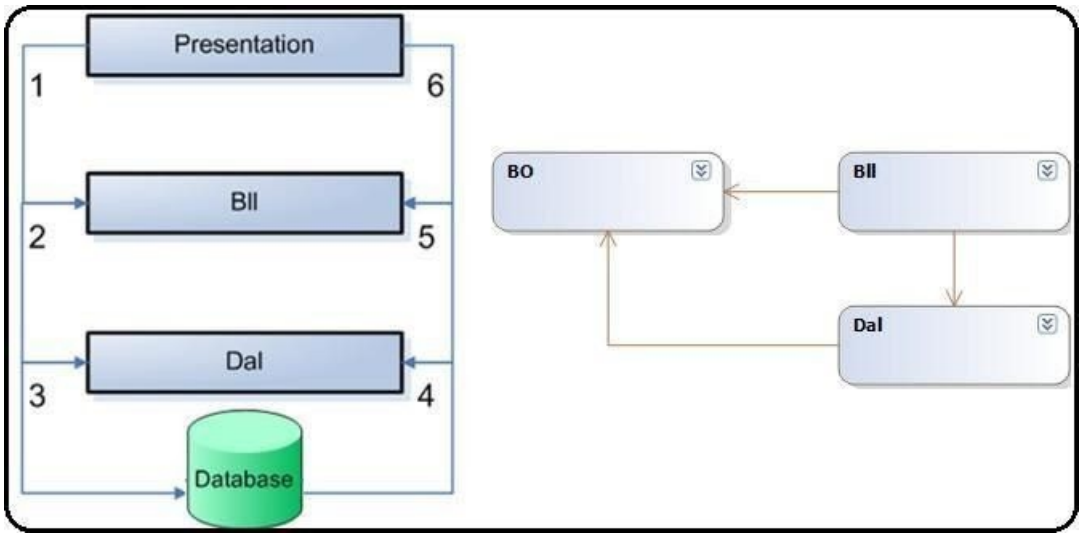
Nunca se llega directamente al DAL, siempre todo deberá pasar primero por el Controlador que será nuestro manager, encargado de orquestar, controlar y definir que método del DAL invocar, para que sea ésta última quien devuelva los resultados esperados, y dar así respuesta a las diferentes peticiones de la vista, requeridas por el usuario.

El DAL tiene por cada objeto del BO, métodos para obtener listados completos, o un solo ítem o registro, y por supuesto los demás métodos básicos del CRUD, para creación, actualización y eliminación de registros.

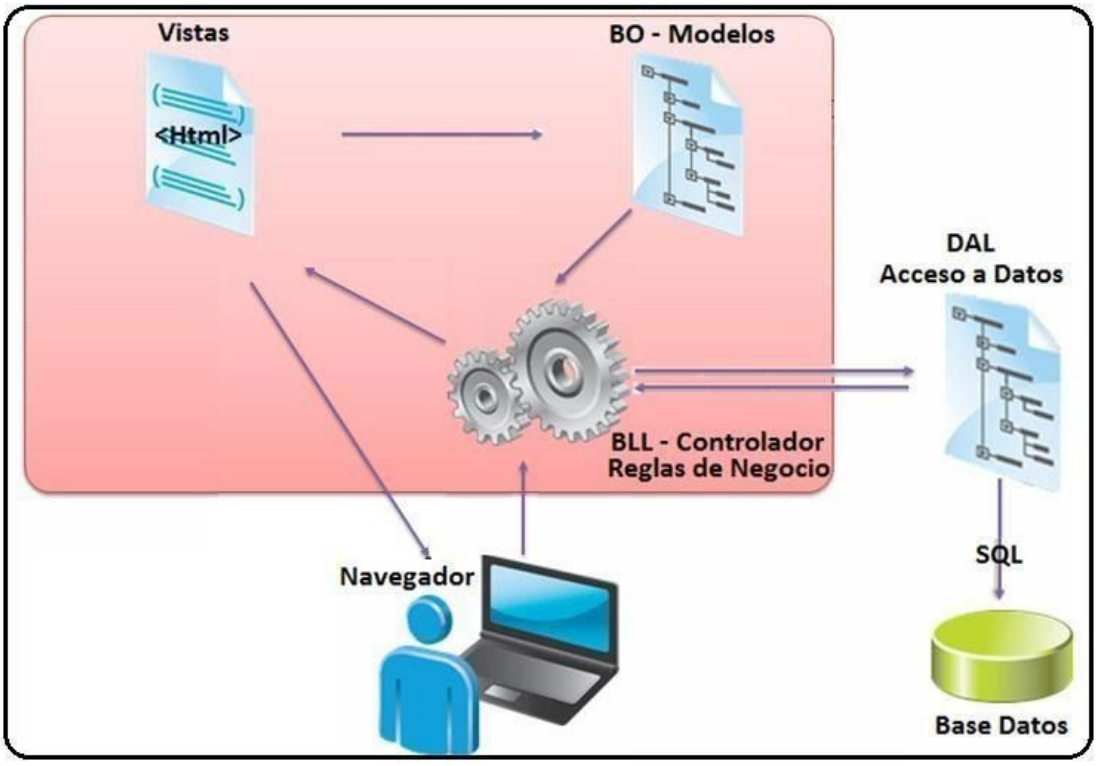
La finalidad de separar los sistemas en capas, no es otra que facilitar su posterior mantenimiento, donde cada capa expone servicios que otras aplicaciones o capas pueden consumir, lo que se traduce en una simplicidad conceptual, de alta cohesión y bajo acoplamiento, facilitando así su reutilización, y MVC nos ayuda con esa tarea, este patrón de arquitectura es definitivamente una belleza, por la forma como trabaja.

Donde predomina una “organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.” Garlan & Shaw.

ESQUEMA DE FUNCIONAMIENTO



Funcionamiento MVC



Estableceremos nuestra cadena de conexión a la BD que ya creamos, para ello ubicamos el archivo: Web.config en el explorador del proyecto y anexamos las siguientes líneas:

```
<connectionStrings>

<add name="Conexion" connectionString="server=XXXSQLEXPRESS; InitialCatalog=ApiRest_DB;
Integrated Security=SSPI;" providerName="System.Data.SqlClient"/>

</connectionStrings>
```

Donde server=XXX es el nombre de tu servidor, que para este caso se autentica con windows, si tienes protegida tu BD, deberás especificar el password, dentro de esta cadena de conexión.

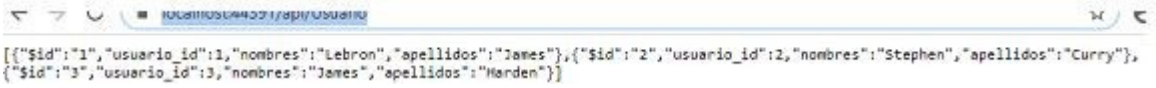
Para que la información sea devuelta como un servicio Rest en formato JSON, debemos asegurarnos de que el archivo: WebApiConfig.cs, que se encuentra ubicado dentro de la carpeta: App_Start, tenga las siguientes líneas:

```
// Configuración y servicios de API web

var json = config.Formatters.JsonFormatter;

json.SerializerSettings.PreserveReferencesHandling =

Newtonsoft.Json.PreserveReferencesHandling.Objects;

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

Una vez ejecutes el proyecto te debe visualizar el index así:

Ahora en la url colocarás el siguiente enlace:
https://localhost:TuPuerto/api/usuario

Te debe visualizar los usuarios existente en la BD así:

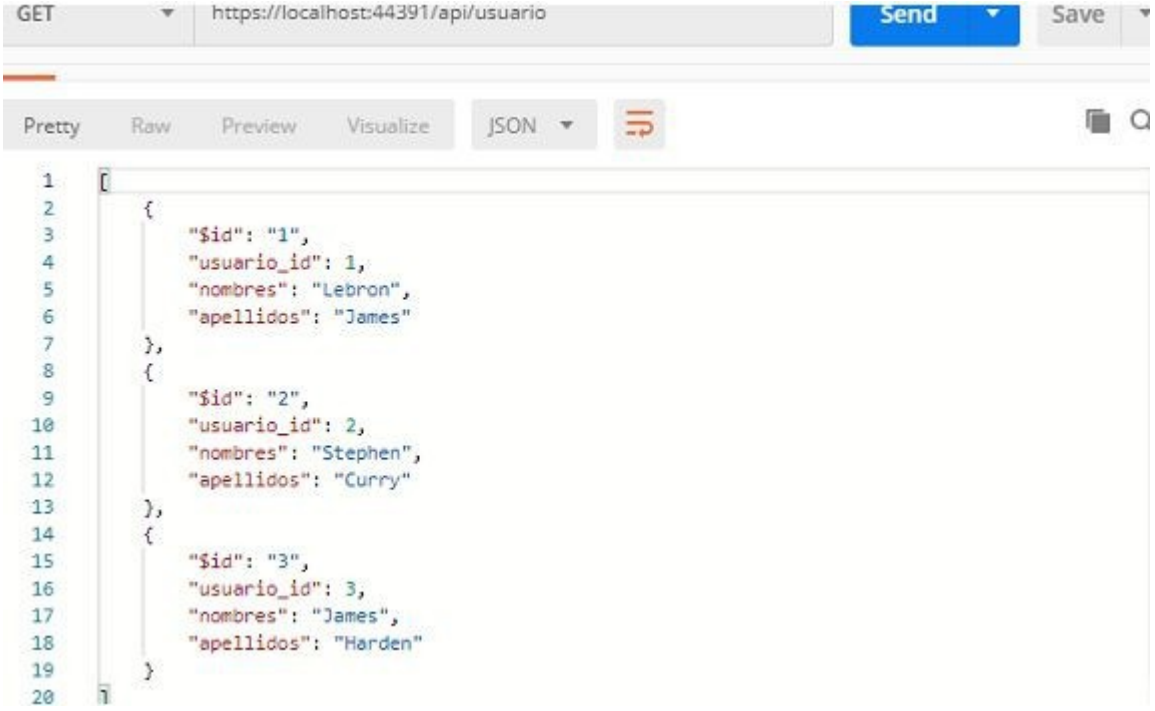


Opcionalmente puedes añadir al final de la url un id y te muestra el usuario al que pertenece ese id, para el caso estamos probando con el ID = 1

Pero probemos inicialmente todo con Postman

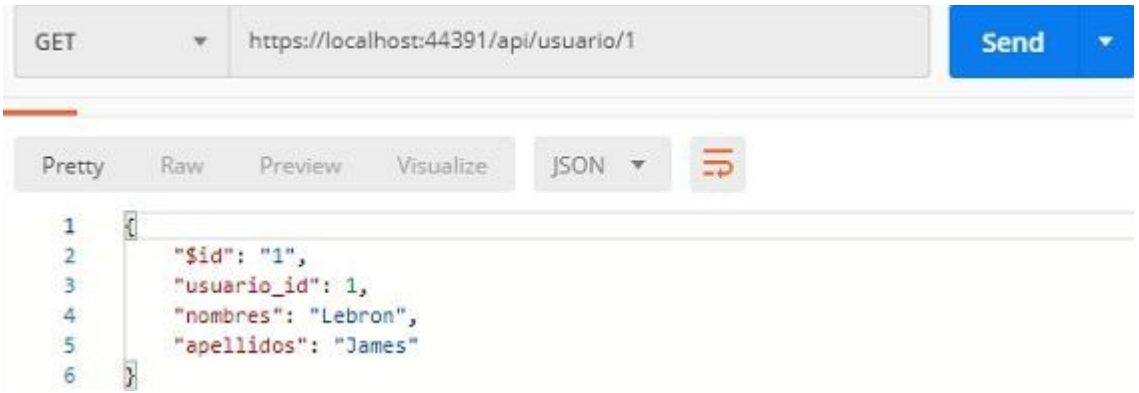
Obtener Todos los Usuarios

GET http://localhost:44391/api/usuario



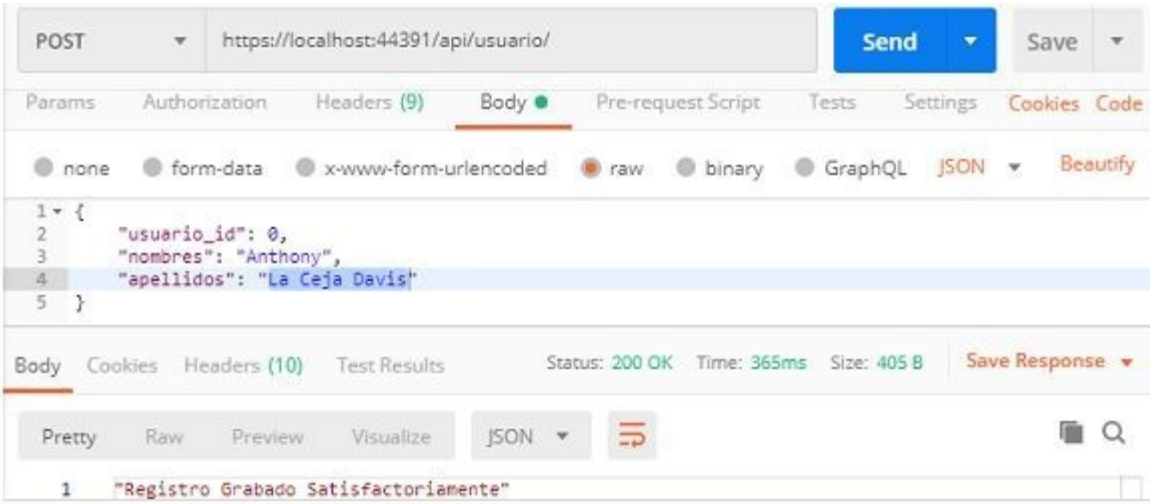
Obtener un Usuario

GET http://localhost:44391/api/usuario/1

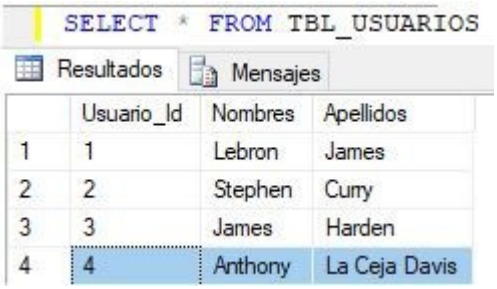


Agregar Un Usuario

POST http://localhost:44391/api/usuario



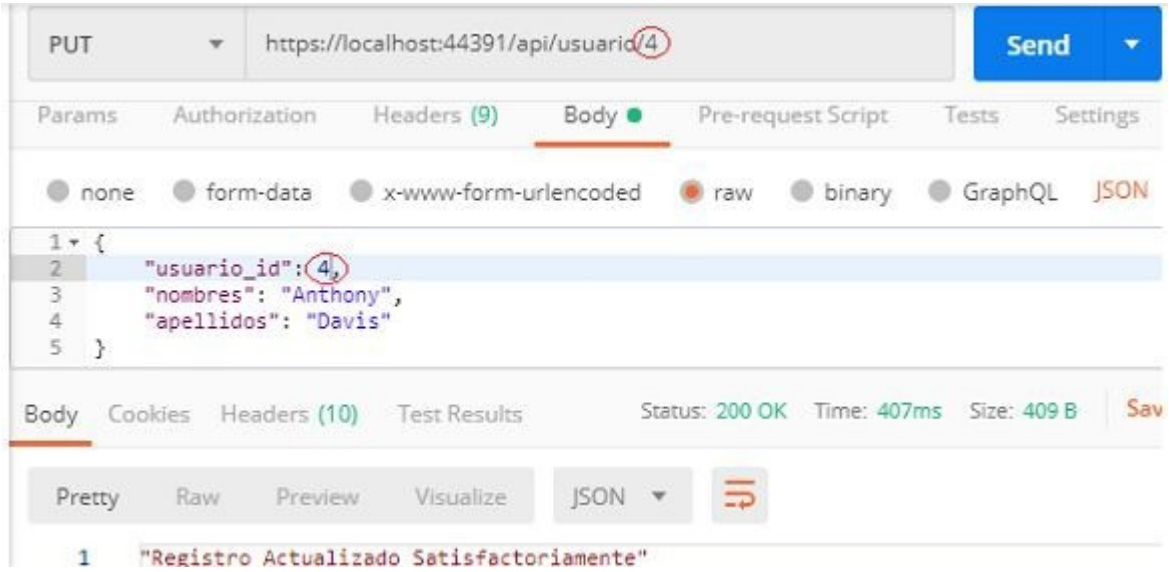
Verificando la consulta en BD



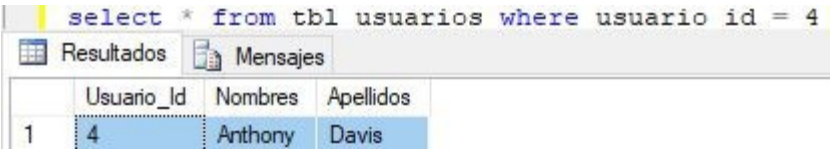
	Usuario_Id	Nombres	Apellidos
1	1	Lebron	James
2	2	Stephen	Curry
3	3	James	Harden
4	4	Anthony	La Ceja Davis

Actualizar Un Usuario

PUT http://localhost:44391/api/usuario/4



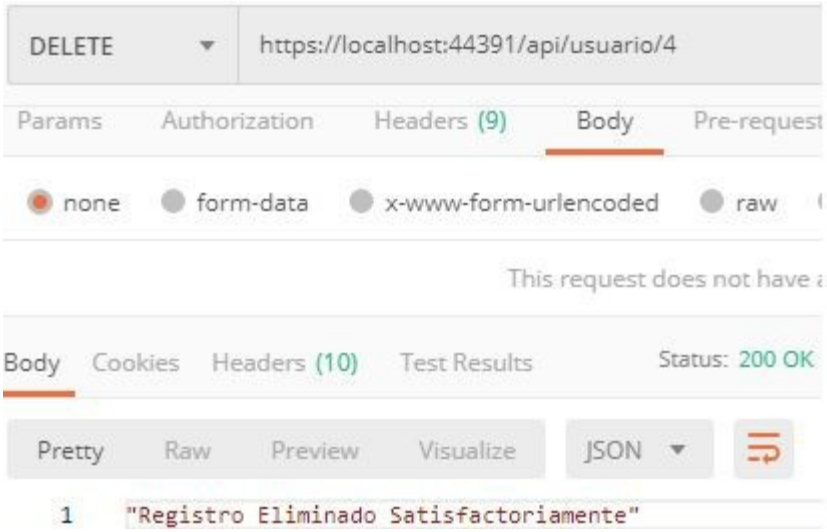
Verificando la consulta en BD



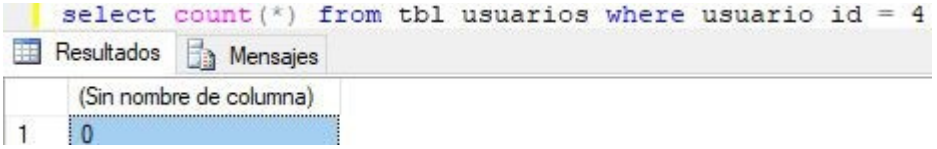
	Usuario_Id	Nombres	Apellidos
1	4	Anthony	Davis

Eliminar Un Usuario

DELETE http://localhost:44391/api/usuario/4

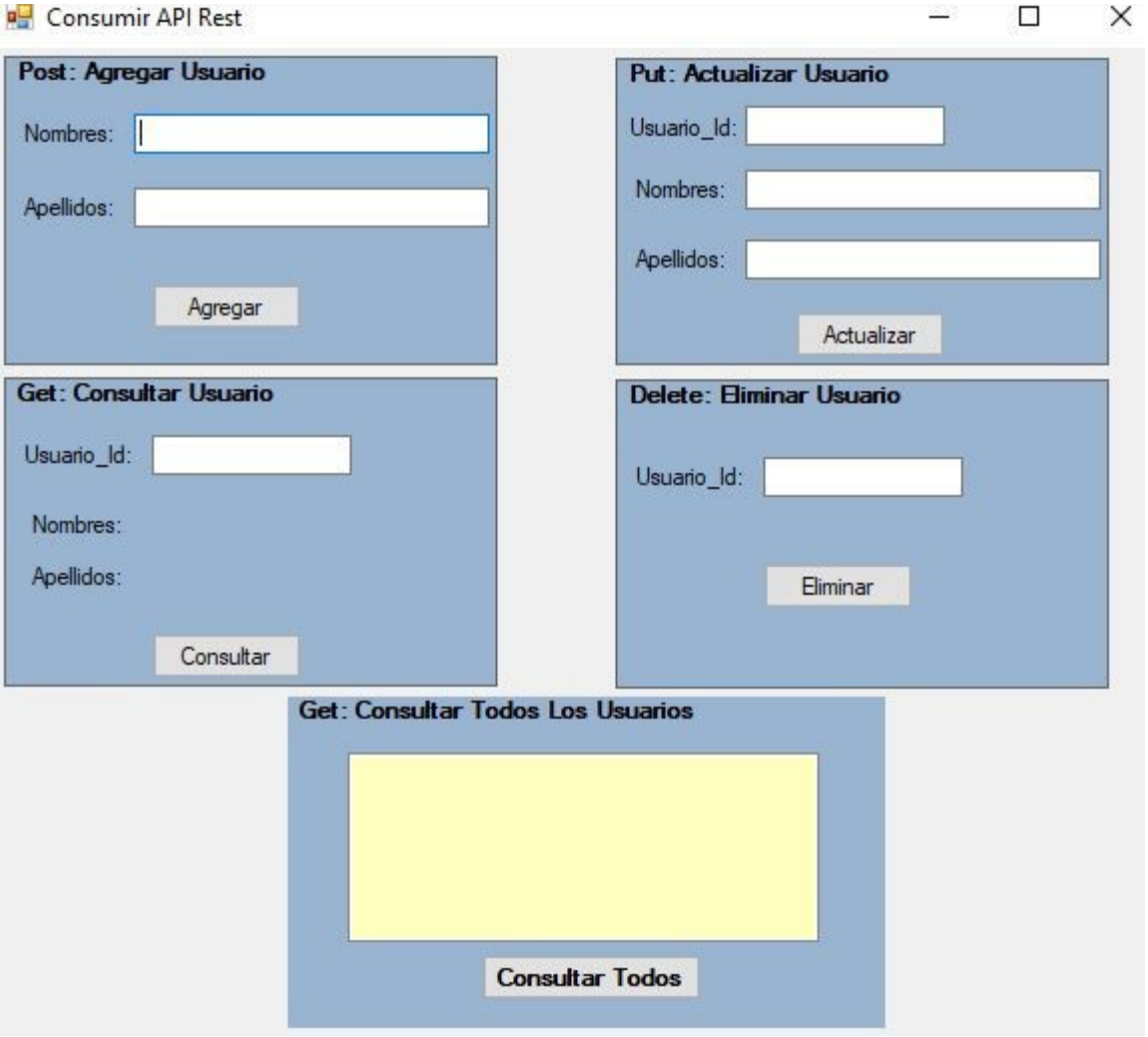


Verificando la consulta en BD



Es importante mencionar que el mensaje de respuesta está sujeto a lo que devuelva el procedimiento almacenado, para el caso de los procedimientos de inserción y borrado, siempre devolverá un 0, si la transacción fue exitosa, en caso contrario es que el registro que se intentó actualizar o borrar no existe, y devolverá un -2, en otro caso devuelve cualquier otro número asociado al error, y visualiza el mensaje: "Error, Se ha producido un error accesando la base de datos".

Ahora bien si no deseas probar con software externo como Postman u otro cualquiera, también lo puedes hacer con un cliente propio que construí, del cual adjunto fuente.



No creo que amerite mucha explicación, pero básicamente cada opción exige que se ingresen los datos (Usuario_ID o Nombres y Apellidos según sea el caso) y listo. Estas interfaces de los clientes no están para nada bonitas, es solo para probar y consumir el API Rest. Cada quien que las mejore y optimice, pues finalmente el objetivo de un cliente que consuma el servicio, busca en esencia es eso, consumir y explotar todas las funcionalidades que exponga el servicio, y presentar los datos a los usuarios de forma más entendible para ellos, no es mostrarlos como los devuelve el servicio en JSON, sino recorrer y extraer de ese JSON la información, y visualizarla en cajas de texto o tablas, según la cantidad de datos que venga.

Lo que sí es **importante** tener en cuenta es que se debe vincular la librería Newtonsoft.Json que ya es nativa en Net, pues los clientes fueron construidos con dicha librería, que bien puede utilizarse otra cualquiera, yo elegí esta porque es de amplio uso y muy fácil de implementar.

Puedes descargar de Github en el siguiente enlace:

https://github.com/JCorreal/API_REST_NET

Y cualquier duda o sugerencia pueden escribirme por acá en Github