

Prácticas

Tema 8 – Desarrollo de Servicios Web

Objetivos	<p>En esta práctica, se aborda la realización de tareas relacionadas con:</p> <ul style="list-style-type: none">• Escribir código para consumir un Servicio Web implementado y en explotación desde una aplicación Web cliente.• Agregar una Referencia de servicio a una aplicación Web cliente o de usuario para obtener acceso a un Servicio Web.• Desarrollar un Servicio Web SOAP empleando Visual C#.• Desarrollar un Servicio Web REST (RESTful).• Invocar la funcionalidad de un Servicio Web y consumir un Servicio Web en modo depuración desde una aplicación Web que actúa como aplicación cliente.• Utilizar herramientas de utilidad que permiten realizar solicitudes HTTP para probar las aplicaciones Web API, u otros tipos de Servicios Web, y poder ver los resultados obtenidos.
Requisitos previos	<p>Para realizar esta práctica, es necesario tener conocimientos sobre:</p> <ul style="list-style-type: none">• Creación de aplicaciones Web basadas en Web Forms y la tecnología de desarrollo ASP.NET.• Creación de aplicaciones Web basadas en la tecnología ASP.NET Core.• Programación lógica de servidor Web empleando archivos de código subyacente.• Lenguajes de marcas XML y JSON.• Construcción y estructura de documentos XML.• Construcción y estructura de documentos JSON.
Escenario	<p>Se presentan varios ejercicios dirigidos a comprender diversos aspectos prácticos sobre el desarrollo de aplicaciones Web distribuidas que están basadas en las Arquitecturas Orientadas a los Servicios (SOA), así como el desarrollo de Servicios Web basados en SOAP y de Servicios Web basados en REST y el consumo de Servicios Web desde una aplicación cliente.</p>

Ejercicio 1

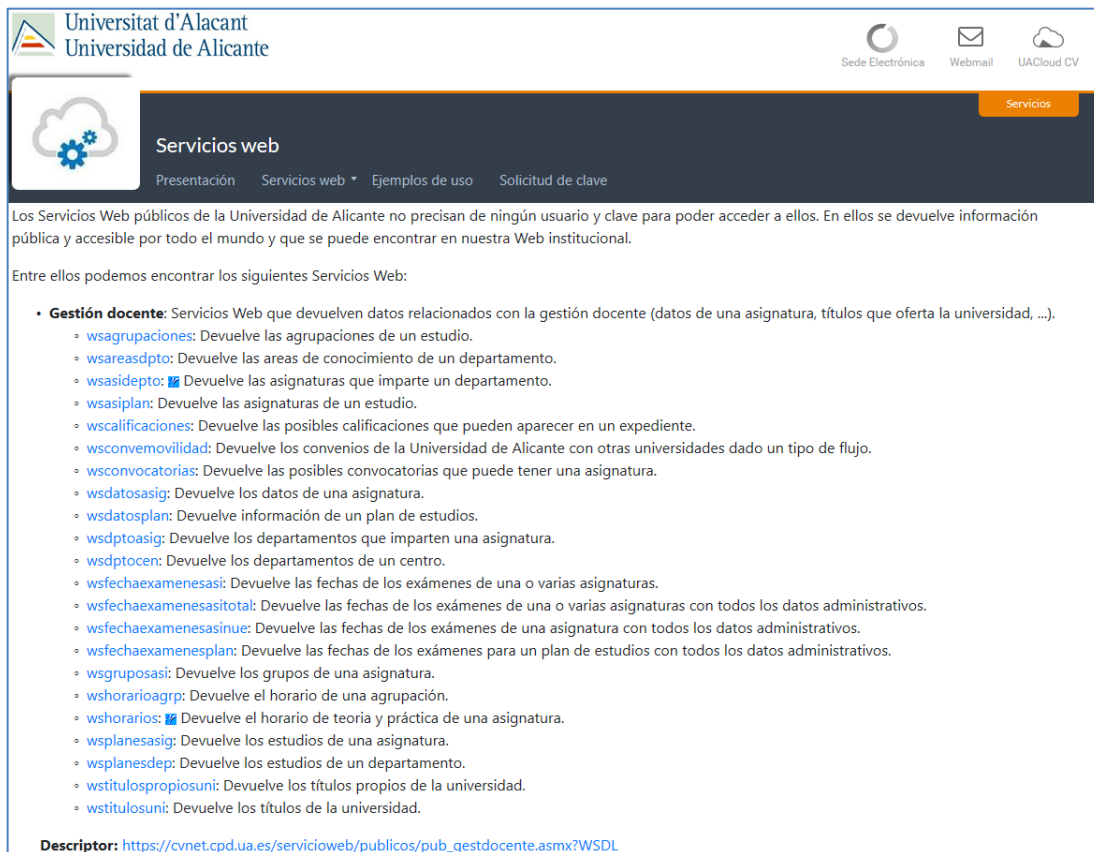
Consumir un Servicio Web ya implementado y publicado

En este ejercicio se aborda cómo obtener acceso a un Servicio Web (*Web Service*) ya implementado, desde una aplicación Web cliente o de usuario. En primer lugar, se localizará y seleccionará un Servicio Web de acuerdo con la funcionalidad deseada. Y, posteriormente, se abordará el desarrollo de una aplicación Web cliente basada en ASP.NET Web Forms para consumir el Servicio Web seleccionado.

Seleccionar y localizar el Servicio Web a consumir

Existen multitud de repositorios de Servicios Web donde se describen las especificaciones de uso y la funcionalidad de los Servicios Web que se ofrecen. En este ejercicio se va a desarrollar una aplicación Web que incorpore un Web Form para presentar diversa información sobre las titulaciones oficiales que ofrece la Universidad de Alicante. Para ello, se utilizará el Servicio Web **pub_gestdocente** que está disponible en el repositorio de Servicios Web públicos de la Universidad de Alicante y que incorpora funcionalidades para la gestión docente. En este mismo repositorio se encuentran disponibles otros Servicios Web públicos que están organizados por categoría. Para consultar la información sobre el Servicio Web **pub_gestdocente**, que corresponde con la categoría **Gestión docente**, acceder al repositorio de Servicios Web públicos que ofrece la Universidad de Alicante visitando la dirección Web:

<https://si.ua.es/es/servicios-web/serviciosweb/publicos/publicos.html>



Universitat d'Alacant
Universidad de Alicante

Sede Electrónica Webmail UACloud CV

Servicios web

Presentación Servicios web Ejemplos de uso Solicitud de clave

Los Servicios Web públicos de la Universidad de Alicante no precisan de ningún usuario y clave para poder acceder a ellos. En ellos se devuelve información pública y accesible por todo el mundo y que se puede encontrar en nuestra Web institucional.

Entre ellos podemos encontrar los siguientes Servicios Web:

- **Gestión docente:** Servicios Web que devuelven datos relacionados con la gestión docente (datos de una asignatura, títulos que oferta la universidad, ...).
 - [wsagrupaciones](#): Devuelve las agrupaciones de un estudio.
 - [wsareasdepto](#): Devuelve las áreas de conocimiento de un departamento.
 - [wsasidepto](#): Devuelve las asignaturas que imparte un departamento.
 - [wsasiplan](#): Devuelve las asignaturas de un estudio.
 - [wscalificaciones](#): Devuelve las posibles calificaciones que pueden aparecer en un expediente.
 - [wsconvemovilidad](#): Devuelve los convenios de la Universidad de Alicante con otras universidades dado un tipo de flujo.
 - [wsconvocatorias](#): Devuelve las posibles convocatorias que puede tener una asignatura.
 - [wsdatosasig](#): Devuelve los datos de una asignatura.
 - [wsdatosplan](#): Devuelve información de un plan de estudios.
 - [wsdptasig](#): Devuelve los departamentos que imparten una asignatura.
 - [wsdptocen](#): Devuelve los departamentos de un centro.
 - [wsfechaexamenasasi](#): Devuelve las fechas de los exámenes de una o varias asignaturas.
 - [wsfechaexamenasitotal](#): Devuelve las fechas de los exámenes de una o varias asignaturas con todos los datos administrativos.
 - [wsfechaexamenasinue](#): Devuelve las fechas de los exámenes de una asignatura con todos los datos administrativos.
 - [wsfechaexamenesplan](#): Devuelve las fechas de los exámenes para un plan de estudios con todos los datos administrativos.
 - [wsgruposasi](#): Devuelve los grupos de una asignatura.
 - [wshorarioagrp](#): Devuelve el horario de una agrupación.
 - [wshorarios](#): Devuelve el horario de teoría y práctica de una asignatura.
 - [wsplanesasig](#): Devuelve los estudios de una asignatura.
 - [wsplanesdep](#): Devuelve los estudios de un departamento.
 - [wstitulospropiosuni](#): Devuelve los títulos propios de la universidad.
 - [wstitulosuni](#): Devuelve los títulos de la universidad.

Descriptor: https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?WSDL

En el documento Web correspondiente a la dirección indicada anteriormente, se presenta una descripción general, así como información relevante para el acceso al Servicio Web **pub_gestdocente**:

- URI de localización del documento de descripción del Servicio Web (*Descriptor*):

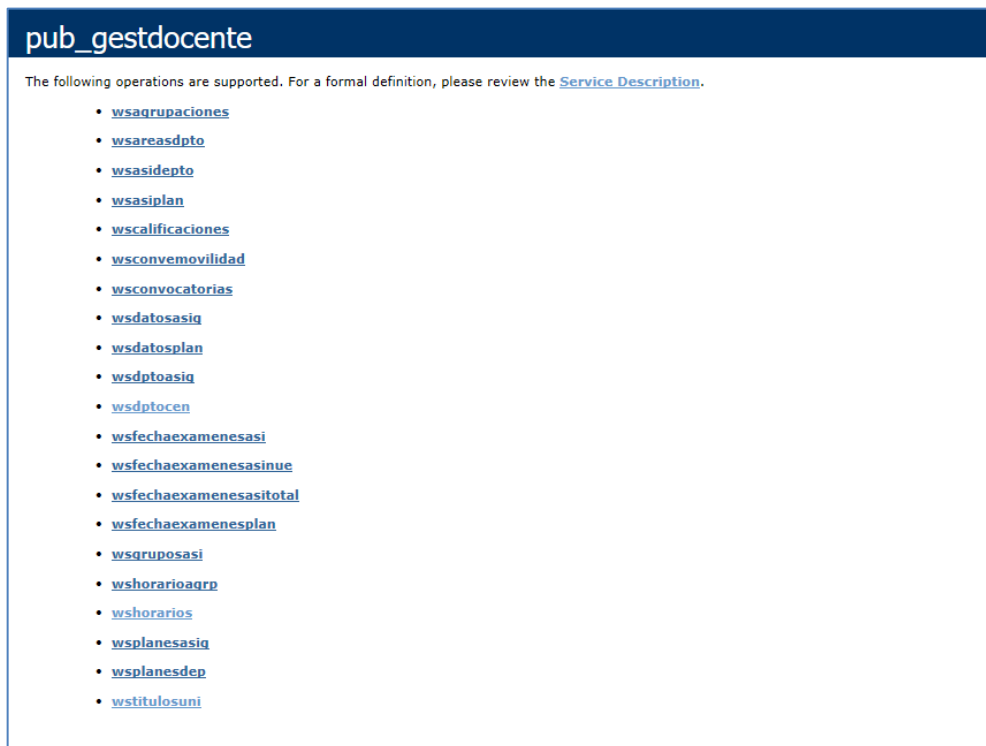
https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?WSDL

- URI de localización o dirección del Servicio Web (*EndPoint*):

https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx

- Breve comentario de los métodos o funcionalidades que expone este Servicio Web. En este ejercicio se utilizará el método **wstitulosuni()**, que permite obtener una lista de las titulaciones oficiales que se imparten en la Universidad de Alicante. A través del enlace correspondiente se puede acceder a la información de detalle sobre el uso del método **wstitulosuni()**.

Es posible obtener información sobre la funcionalidad **wstitulosuni()** a través del navegador Web. Para ello, hacer clic sobre el enlace **wstitulosuni** y se accederá a la página que contiene la información técnica de uso de esta funcionalidad. Además, se puede obtener más información técnica visitando la dirección Web del Servicio Web (*EndPoint*) y seleccionando el enlace de la operación **wstitulosuni()**. A continuación, se mostrará una página que incluye ejemplos de mensajes SOAP de petición y de respuesta, en los formatos de las versiones 1.1 y 1.2. También es posible acceder al documento WSDL de descripción del Servicio Web **pub_gestdocente**. Para ello, acceder a la dirección Web del Servicio Web (*EndPoint*) y seleccionar el enlace *Service Description* situado en la parte superior.



The screenshot shows a web page titled "pub_gestdocente". Below the title, it states "The following operations are supported. For a formal definition, please review the [Service Description](#)." followed by a list of 21 operations, each with a blue hyperlink:

- [wsagrupaciones](#)
- [wsareasdpto](#)
- [wsasidepto](#)
- [wsasiplan](#)
- [wscalificaciones](#)
- [wsconvemovilidad](#)
- [wsconvocatorias](#)
- [wsdatosasiq](#)
- [wsdatosplan](#)
- [wsdptoasiq](#)
- [wsdptocen](#)
- [wsfechaexamenesasi](#)
- [wsfechaexamenesasinue](#)
- [wsfechaexamenesasitotal](#)
- [wsfechaexamenesaplan](#)
- [wsgruposasi](#)
- [wshorarioagrp](#)
- [wshorarios](#)
- [wsplanesasiq](#)
- [wsplanesdep](#)
- [wstitulosuni](#)

Crear una aplicación Web cliente para consumir un Servicio Web

A continuación, se creará una aplicación Web basada en ASP.NET Web Forms (.NET Framework) que invocará el método **wstitulosuni()** del Servicio Web **pub_gestdocente** desde un Web Form. Esta aplicación Web permitirá acceder y presentar diversa información relativa a las titulaciones oficiales ofrecidas por la Universidad de Alicante.

Para crear una nueva aplicación Web de ASP.NET y un nuevo Web Form, hacer:

1. Crear una **Solución** de Visual Studio, denominada *ServiciosWeb* y, formando parte de esta, un **Proyecto** vacío de aplicación Web de ASP.NET para lenguaje C# denominado *ServiciosWebCS*. Para ello, puede seguirse el procedimiento realizado en las prácticas del primer tema.
2. Crear un nuevo Web Form denominado *Ejercicio1.aspx*. Para ello, hacer clic con el botón secundario sobre el nombre del Proyecto en el **Explorador de soluciones**, seleccionar **Agregar y Nuevo elemento...**, elegir **Formulario Web Forms** e introducir el nombre del Web Form.
3. Desarrollar una interfaz Web que presente una interfaz Web cuya apariencia sea similar a la que se muestra en la siguiente Ilustración.

CONSUMIR UN SERVICIO WEB YA EXISTENTE

Titulaciones Oficiales en la Universidad de Alicante

Curso académico (formato aaaa-aa)

4. Asignar el valor *btnObtenerInformacion* a la propiedad **Id** del control Button del Web Form.
5. Debajo del control TextBox que permite introducir el curso, agregar un control Label denominado *lblResultado* que se utilizará para mostrar el resultado obtenido de la invocación del método **wstitulosuni()**. Una vez agregado el control Label *lblResultado*, eliminar el valor de la propiedad Text para hacer que la etiqueta quede vacía.
6. Debajo del control Label agregado en el punto anterior, agregar un control de datos GridView, denominado *GridView1*. Este control GridView se empleará para presentar los resultados obtenidos del consumo del Servicio Web sobre la interfaz de la aplicación Web.

Agregar una Referencia de servicio

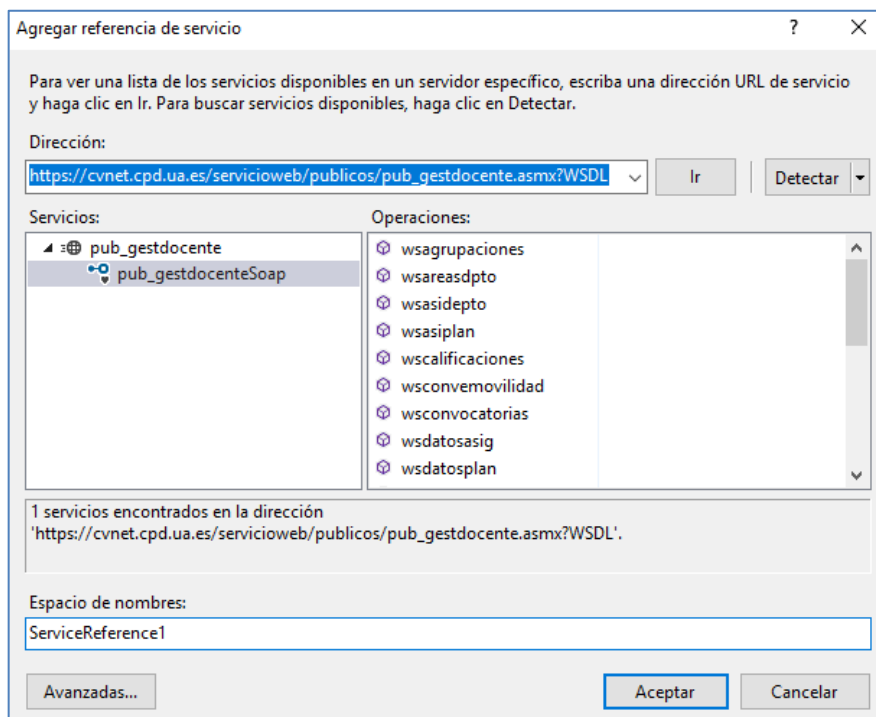
La localización de un Servicios Web es un proceso por el cual un cliente encuentra un Servicio Web y obtiene su descripción. El objetivo de este proceso es encontrar la descripción del servicio que se presenta como un documento XML que utiliza el lenguaje de descripción WSDL. La descripción del Servicio Web muestra los servicios disponibles y cómo interactuar con ellos. Es necesario obtener la descripción de un Servicio Web para poder interactuar con él. La aplicación Web que consumirá un Servicio Web, debe tener un mecanismo para comunicarse con el Servicio Web y poder así encontrarlo en tiempo de ejecución. Este mecanismo es una **Referencia de servicio** que se agrega al Proyecto y que sirve para interactuar con el Servicio Web y proporcionarle una representación local.

Para agregar una Referencia de servicio al Proyecto *ServiciosWebCS* que actuará como aplicación Web cliente que consume el Servicio Web **pub_gestdocente**, realizar las siguientes acciones:

1. En el **Explorador de soluciones**, seleccionar el Proyecto *ServiciosWebCS* y, a continuación, en el menú **Proyecto**, elegir la opción **Agregar referencia de servicio...** O también, en el **Explorador de soluciones** seleccionar el nombre del Proyecto, hacer clic en el botón derecho, desplegar la opción **Agregar** y seleccionar **Referencia de servicio...**
2. En el cuadro de texto **Dirección** de la ventana **Agregar referencia de servicio**, introducir la localización del *Descriptor* del Servicio Web al que desea tener acceso, que en este caso es:

https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx

3. A continuación, hacer clic en el botón **Ir** situado a la derecha del cuadro de texto utilizado anteriormente para recuperar la descripción del Servicio Web.



Agregar referencia de servicio

Para ver una lista de los servicios disponibles en un servidor específico, escriba una dirección URL de servicio y haga clic en Ir. Para buscar servicios disponibles, haga clic en Detectar.

Dirección:
 Ir Detectar

Servicios:

- pub_gestdocente
- pub_gestdocenteSoap

Operaciones:

- wsagrupaciones
- wsareasdpto
- wsasidepto
- wsasiplan
- wscalificaciones
- wsconvemovilidad
- wsconvocatorias
- wsdatosasig
- wsdatosplan

1 servicios encontrados en la dirección
'https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?WSDL'.

Espacio de nombres:

Avanzadas... Aceptar Cancelar

4. Una vez cargados los datos, en el cuadro **Servicios** desplegar la opción **pub_gestdocente** y seleccionar la opción **pub_gestdocenteSoap** para ver las operaciones o métodos disponibles.
5. En el cuadro de texto **Espacio de nombres** se propondrá, posiblemente, como nombre *ServiceReference1*. Este nombre constituirá la referencia local en la aplicación Web cliente para tener acceso al Servicio Web. El nombre propuesto *ServiceReference1* es adecuado.
6. Para finalizar, hacer clic en el botón **Aceptar**.

Al crear la Referencia de servicio, además de haberse descargado la descripción del servicio, se ha creado una representación local que facilita la comunicación entre la aplicación Web cliente y el Servicio Web que es consumido por esta. Mediante el **Explorador de soluciones** se puede comprobar que se habrá creado el componente *ServiceReference1* dentro de la carpeta *Connected Services* que contiene la información de la Referencia de servicio agregada al Proyecto *ServiciosWebCS*. Además, se habrá creado la sección `<system.serviceModel>` que incluye el elemento `<endpoint>` en el archivo *Web.config*. Esta sección define la configuración del acceso al Servicio Web.

En algunos casos, se pueden haber agregado varias secciones `<endpoint>` en el archivo *Web.config* que especifican los diversos formatos de acceso al Servicio Web. En estos casos, es necesario comentar o eliminar todas las secciones `<endpoint>` excepto aquella que se vaya a utilizar. Si se prueba el funcionamiento de la aplicación Web y existen varias secciones `<endpoint>` en el archivo de configuración *Web.config* se producirá el siguiente error de depuración en tiempo de ejecución:

"No se pudo cargar una sección de configuración de extremo para el contrato 'ServiceReference1.pub_gestdocentetSoap' porque se encontró más de una configuración de extremo para dicho contrato. Indique la sección de configuración de extremo preferida por nombre".

Obtener acceso al Servicio Web

Una vez agregada la Referencia de servicio del Servicio Web **pub_gestdocente** en el Proyecto de aplicación Web *ServiciosWebCS*, el siguiente paso es crear una instancia de la clase *proxy* en la aplicación Web para representar al Servicio Web. A partir de ese momento, se podrá obtener acceso a los métodos que expone el Servicio Web, de la misma forma que se haría para cualquier método de objeto. Una vez realizada la llamada al método del Servicio Web, la información SOAP recibida con los resultados obtenidos será presentada visualmente sobre interfaz del Web Form.

A continuación, se proporciona el código lógico para realizar la llamada al método **wstitulosuni()** del Servicio Web **pub_gestdocente** y la presentación visual de los resultados. Este código lógico se asocia al evento *Click* del control Button *btnObtenerInformacion* en el archivo de código subyacente *Ejercicio1.aspx.cs*. Para ello, realizar las siguientes acciones:

1. Abrir el archivo de código subyacente *Ejercicio1.aspx.cs*.
2. Especificar la cláusula *using* que referencia al espacio de nombres creado al agregar la Referencia de servicio:

```
using ServiciosWebCS.ServiceReference1;
```

3. Introducir el código asociado al evento *Click* del control Button *btnObtenerInformacion*:

```
protected void btnObtenerInformacion_Click(object sender, EventArgs e)
{
    // Consumo de Servicio Web wstitulosuni de la Universidad de Alicante
    // disponible en: https://si.ua.es/es/servicios-web/serviciosweb/publicos/publicos.html
    try
    {
        // Creación de la instancia y llamada al método
        pub_gestdocenteSoapClient ws = new pub_gestdocenteSoapClient();
        string StrCursoAcademico = txtCurso.Text;
        var resultado = ws.wstitulosuni("C", StrCursoAcademico);
        ws.Close();

        lblResultado.Text = "<b>Resultados obtenidos de la respuesta: </b>" + resultado;

        // Incluye los datos en tabla
        string StrTabla = "<div style='display:table; width:70%; margin:auto'>";
        StrTabla += "<div style='display:table-row;height:25px;'>";
        StrTabla += "<div style='display:table-cell;'><b>CÓDIGO</b></div>";
        StrTabla += "<div style='display:table-cell; width:50%;'><b>NOMBRE</b></div>";
        StrTabla += "<div style='display:table-cell;'><b>TIPO</b></div>";
        StrTabla += "<div style='display:table-cell;'><b>ÁREA</b></div>";
        StrTabla += "<div style='display:table-cell;'><b>ENLACE</b></div>";
        StrTabla += "</div>";
        foreach (var item in resultado)
        {
            StrTabla += "<div style='display:table-row; height:20px;'>";
            StrTabla += "<div style='display:table-cell;'>" + item.codigo + "</div>";
            StrTabla += "<div style='display:table-cell;'>" + item.nombre + "</div>";
            if (item.tipo == "0")
                StrTabla += "<div style='display:table-cell;'>Grado</div>";
            if (item.tipo == "1")
                StrTabla += "<div style='display:table-cell;'>Master</div>";
            StrTabla += "<div style='display:table-cell;'>" + item.area + "</div>";
            StrTabla += "<div style='display:table-cell;'>" +
                "<a href='" + item.url + "'>Más información</a></div>";
            StrTabla += "</div>";
        }
        StrTabla += "</div>";
        lblResultado.Text = StrTabla;

        // Incluye los datos de la respuesta en un GridView
        GridView1.AutoGenerateColumns = true;
        GridView1.DataSource = resultado;
        GridView1.DataBind();
    }
    catch (Exception ex)
    {
        string StrError = "<p>Se han producido errores durante el registro</p>";
        StrError = StrError + "<div>Código: " + ex.HResult + "</div>";
        StrError = StrError + "<div>Descripción: " + ex.Message + "</div>";
        lblResultado.Text = "Información de " + txtCurso.Text.ToUpper() +
            " no disponible <br />" + StrError;
        GridView1.Dispose();
        GridView1.DataBind();
    }
}
```


En el código anterior, puede observarse cómo se realiza la llamada al método o funcionalidad `wstitulosuni()` del Servicio Web `pub_gestdocente`, utilizando para ello la Referencia al servicio creada previamente. Este método devuelve una lista de objetos `ClaseTitulosUni` que incluyen información sobre las titulaciones oficiales ofrecidas por la Universidad de Alicante para cada curso. Una vez obtenida la respuesta se presenta visualmente la información recibida de dos formas diferentes mediante un control `Label` y un control `GridView`.

- En el control `Label lblResultado`, se muestra la información resultante en formato tabular que ha sido manipulada por el desarrollador en el código.
- En el control `GridView GridView1`, se presenta directamente la información recuperada sobre ese control. Ello es posible porque el resultado es una información compleja con una estructura bidimensional.

4. Iniciar la depuración para comprobar los resultados obtenidos.

CONSUMIR UN SERVICIO WEB YA EXISTENTE

Titulaciones Oficiales en la Universidad de Alicante

Curso académico (formato aaaa-aa)

CÓDIGO	NOMBRE	TIPO	ÁREA	ENLACE
C001	GRADO EN GEOGRAFÍA Y ORDENACIÓN DEL TERRITORIO	Grado	Ciencias Sociales y Jurídicas	Más información
C002	GRADO EN HISTORIA	Grado	Artes y Humanidades	Más información
C003	GRADO EN HUMANIDADES	Grado	Artes y Humanidades	Más información
C004	GRADO EN TURISMO	Grado	Ciencias Sociales y Jurídicas	Más información
C005	GRADO EN ESTUDIOS ÁRABES E ISLÁMICOS	Grado	Artes y Humanidades	Más información
C006	GRAU EN FILOLOGIA CATALANA	Grado	Artes y Humanidades	Más información
C007	GRADO EN ESTUDIOS FRANCESES	Grado	Artes y Humanidades	Más información
C008	GRADO EN ESPAÑOL: LENGUA Y LITERATURAS	Grado	Artes y Humanidades	Más información
C009	GRADO EN ESTUDIOS INGLESES	Grado	Artes y Humanidades	Más información
C010	GRADO EN TRADUCCIÓN E INTERPRETACIÓN	Grado	Artes y Humanidades	Más información
C011	GRADO EN TRADUCCIÓN E INTERPRETACIÓN. ALEMÁN	Grado	Artes y Humanidades	Más información
C012	GRADO EN TRADUCCIÓN E INTERPRETACIÓN. INGLÉS	Grado	Artes y Humanidades	Más información
C013	GRADO EN TRADUCCIÓN E INTERPRETACIÓN. FRANCÉS	Grado	Artes y Humanidades	Más información
C051	GRADO EN GEOLOGÍA	Grado	Ciencias	Más información
C052	GRADO EN MATEMÁTICAS	Grado	Ciencias	Más información
C053	GRADO EN QUÍMICA	Grado	Ciencias	Más información
C054	GRADO EN BIOLOGÍA	Grado	Ciencias	Más información
C055	GRADO EN CIENCIAS DEL MAR	Grado	Ciencias	Más información
C056	GRADO EN ÓPTICA Y OPTOMETRÍA	Grado	Ciencias de la Salud	Más información
C057	GRADO EN FÍSICA	Grado	Ciencias	Más información
C058	GRADO EN GASTRONOMÍA Y ARTES CULINARIAS	Grado	Ciencias Sociales y Jurídicas	Más información
C101	GRADO EN GESTIÓN Y ADMINISTRACIÓN PÚBLICA	Grado	Ciencias Sociales y Jurídicas	Más información
C102	GRADO EN DERECHO	Grado	Ciencias Sociales y Jurídicas	Más información
C103	GRADO EN CRIMINOLOGÍA	Grado	Ciencias Sociales y Jurídicas	Más información
C104	GRADO EN RELACIONES LABORALES Y RECURSOS HUMANOS	Grado	Ciencias Sociales y Jurídicas	Más información
C105	GRADO EN RELACIONES INTERNACIONALES	Grado	Ciencias Sociales y Jurídicas	Más información
C110	DOBLE GRADO EN DERECHO Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS	Grado	Ciencias Sociales y Jurídicas	Más información
C111	DOBLE GRADO EN DERECHO Y CRIMINOLOGÍA	Grado	Ciencias Sociales y Jurídicas	Más información
C151	GRADO EN SOCIOLOGÍA	Grado	Ciencias Sociales y Jurídicas	Más información
C152	GRADO EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS	Grado	Ciencias Sociales y Jurídicas	Más información
C153	GRADO EN ECONOMÍA	Grado	Ciencias Sociales y Jurídicas	Más información
C154	GRADO EN PUBLICIDAD Y RELACIONES PÚBLICAS	Grado	Ciencias Sociales y Jurídicas	Más información
C155	GRADO EN TRABAJO SOCIAL	Grado	Ciencias Sociales y Jurídicas	Más información
C156	GRADO EN MARKETING	Grado	Ciencias Sociales y Jurídicas	Más información
C160	DOBLE GRADO EN TURISMO Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS	Grado	Ciencias Sociales y Jurídicas	Más información

Ejercicio 2

Crear un Servicio Web SOAP

En este ejercicio se aborda el proceso de desarrollo de un Servicio Web que realice los cálculos para obtener el Total bruto, el Descuento, la Base imponible, el IVA y el Total en una línea de factura.

Crear un Servicio Web

A continuación, se creará una nueva **Solución** de Visual Studio, denominada *Factura*, y un nuevo **Proyecto de aplicación ASP.NET (.NET Framework)**, denominado *FacturaWS*, que contendrá un Servicio Web SOAP cuya funcionalidad permitirá obtener los resultados de los cálculos relativos a la venta de un producto en líneas de factura. Para ello, realizar las siguientes acciones:

1. Crear una **Solución** de Visual Studio denominada *Factura* y, formando parte de esta, un **Proyecto** vacío de **Aplicación web ASP.NET (.NET Framework)** para lenguaje C# denominado *FacturaWS*. Para ello, puede seguirse el proceso realizado en las prácticas del primer tema.
2. En el **Explorador de soluciones**, hacer clic con el botón secundario sobre el nombre del Proyecto, *FacturaWS*, desplegar la opción **Agregar...** y seleccionar la opción **Nuevo elemento...** En la ventana **Agregar nuevo elemento**, desplegar las opciones **Instalado** y **Visual C#** y, a continuación, seleccionar la opción **Web** en el cuadro de la izquierda. En el cuadro de la derecha, seleccionar la plantilla de archivo **Servicio Web (ASMX)**. En el cuadro de texto **Nombre** introducir el nombre del archivo del Servicio Web a desarrollar, que en este caso se denominará: *CalculosFacturaWS.asmx*. Finalmente, hacer clic en botón **Agregar**. En este momento, Visual Studio creará automáticamente los archivos y las referencias necesarias.
3. Mediante el **Explorador de Soluciones** puede comprobarse que se habrá creado el archivo *CalculosFacturaWS.asmx* en la carpeta raíz del Proyecto y el archivo de código subyacente *CalculosFacturaWS.asmx.cs*. El archivo del Servicio Web SOAP, *CalculosFacturaWS.asmx*, es el archivo que las aplicaciones Web cliente podrán invocar para llamar a los métodos o funcionalidades que incluya. Y el archivo de código subyacente *CalculosFacturaWS.asmx.cs* contendrá el código lógico que implementará los métodos o funcionalidades del Servicio Web y que quedarán definidos dentro de la clase *CalculosFacturaWS*.
4. El siguiente paso es escribir el código de los métodos del Servicio Web SOAP dentro de la clase *CalculosFacturaWS*, de manera que se pueda poder expresar la funcionalidad que expondrá este Servicio Web. Para ello, realizar las siguientes acciones:
 - a. Abrir el archivo de código subyacente *CalculosFacturaWS.asmx.cs*.
 - b. Buscar la declaración de la clase pública *CalculosFacturaWS* y adecuar el código del atributo *WebService* situado justo antes de la declaración de esta clase, para añadir una descripción sobre la funcionalidad del Servicio Web de la siguiente forma:

```
[WebService(Namespace = "http://tempuri.org/",  
Description = "Servicio Web para la obtención de cálculos en líneas de factura.")]
```

Este Servicio Web utiliza el espacio de nombres *http://tempuri.org/*. Cada Servicio Web necesita un espacio de nombres único para que las aplicaciones Web que lo consumen puedan diferenciar este servicio de otros Servicios Web. El espacio de nombres *http://tempuri.org/* de Microsoft está disponible para Servicios Web que

están en fase de desarrollo. Los Servicios Web que están publicados en un Servidor Web deben utilizar un espacio de nombres permanente y más significativo en el entorno de explotación, por lo que se recomienda cambiar el espacio de nombres predeterminado antes de implementar un Servicio Web en su entorno de explotación.

- c. Agregar el siguiente código lógico en la clase *CalculosFacturaWS*, justo antes del método predeterminado *HelloWorld*.

```
[Serializable()]
public class Calculos
{
    public Calculos() { }
    public string Bruto;
    public string Descuento;
    public string BaseImponible;
    public string Iva;
    public string Total;
}

[WebMethod]
public Calculos CalculosLineaFactura(double cantidad, double precio,
    double tipo_descuento, double tipo_iva)
{
    Calculos resultado = new Calculos();
    double w_bruto = cantidad * precio;
    double w_descuento = cantidad * precio * tipo_descuento / 100;
    double w_baseImponible = w_bruto - w_descuento;
    double w_iva = w_baseImponible * tipo_iva / 100;
    double w_total = w_baseImponible + w_iva;
    resultado.Bruto = String.Format("{0:c}", w_bruto);
    resultado.Descuento = String.Format("{0:c}", w_descuento);
    resultado.BaseImponible = String.Format("{0:c}", w_baseImponible);
    resultado.Iva = String.Format("{0:c}", w_iva);
    resultado.Total = String.Format("{0:c}", w_total);

    return resultado;
}
```

El código anterior expone la funcionalidad principal de este Servicio Web SOAP mediante el método *CalculosLineaFactura()* que utiliza el objeto *resultado* definido como instancia de la clase *Calculos*. En el código anterior puede observarse en el código cómo, una vez realizados los cálculos, el método devolverá el objeto *resultado* para su envío a la aplicación Web a través de un documento SOAP. De manera implícita y transparente para el desarrollador se realiza una **serialización** del objeto *resultado*. Se conoce por serialización el proceso de codificación de un objeto en un medio de almacenamiento, como puede ser un archivo, o un buffer de memoria, con el fin de transmitirlo a través de una red. Esta transmisión puede realizarse como una serie de bytes o en un formato humanamente más legible como XML, SOAP o JSON. La serialización es un mecanismo ampliamente usado para transportar objetos a través de una red. En este caso, de manera implícita, se codifica el objeto resultado en un documento SOAP para su envío a la aplicación Web que consume el Servicio Web.

5. Finalmente, guardar el archivo *CalculosFacturaWS.asmx.cs*.

Probar el funcionamiento del Servicio Web SOAP

Una vez creado un Servicio Web SOAP se va a comprobar su funcionamiento, tal como se haría con cualquier otro elemento de un Proyecto. Para probar el Servicio Web en modo depuración, hacer:

1. En el **Explorador de soluciones**, abrir y seleccionar el archivo *CalculosFacturaWS.asmx.cs* y a continuación, iniciar la depuración de la forma habitual.
2. Al iniciar la depuración del Servicio Web, aparece en el navegador una página que muestra una breve descripción de la funcionalidad del Servicio Web, un enlace que abre su definición formal o descripción WSDL y los enlaces hacia las operaciones o métodos que expone.



3. Para revisar la descripción WSDL del Servicio Web creado, hacer clic sobre el enlace de descripción de servicios. Esta descripción WSDL ha sido generada de manera automática y predeterminada por el entorno de desarrollo.
4. Desde la página inicial, es posible realizar una comprobación del funcionamiento de las funcionalidades agregadas, haciendo clic sobre el enlace de cada una de las operaciones o métodos definidos en el Servicio Web. En este caso, para comprobar su funcionamiento, se puede hacer clic sobre la opción *CalculosLineaFactura*.

Haga clic [aquí](#) para obtener una lista completa de operaciones.

CalculosLineaFactura

Prueba

Haga clic en el botón 'Invocar', para probar la operación utilizando el protocolo HTTP POST.

Parámetro	Valor
cantidad:	50
precio:	2
tipo_descuento:	5
tipo_iva:	21

SOAP 1.1

A continuación se muestra un ejemplo de solicitud y respuesta para SOAP 1.1. Es necesario reemplazar los marcadores de posición que aparecen con valores reales.

```
POST /CalculosFacturaWS.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/CalculosLineaFactura"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CalculosLineaFactura xmlns="http://tempuri.org/">
      <cantidad>double</cantidad>
      <precio>double</precio>
      <tipo_descuento>double</tipo_descuento>
      <tipo_iva>double</tipo_iva>
    </CalculosLineaFactura>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CalculosLineaFactura xmlns="http://tempuri.org/">
      <Bruto>100,00 €</Bruto>
      <Descuento>10,00 €</Descuento>
      <BaseImponible>90,00 €</BaseImponible>
      <Iva>18,90 €</Iva>
      <Total>108,90 €</Total>
    </Calculos>
  </soap:Body>
</soap:Envelope>
```

Como puede observarse, aparece una página predeterminada que solicita los valores de los argumentos o parámetros que utiliza el método *CalculosLineaFactura()*. Además, puede apreciarse que se presentan ejemplos de solicitud y respuesta, tanto SOAP como HTTP.

5. Al introducir valores en los cuadros de texto correspondientes y hacer clic en el botón **Invocar**, aparece en la ventana de visualización del navegador Web el documento XML de respuesta que el Servicio Web devuelve cuando se procesa el método con los valores introducidos.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Calculos xmlns="http://tempuri.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Bruto>100,00 €</Bruto>
  <Descuento>10,00 €</Descuento>
  <BaseImponible>90,00 €</BaseImponible>
  <Iva>18,90 €</Iva>
  <Total>108,90 €</Total>
</Calculos>
```

Finalizada la creación del Servicio Web, el siguiente paso sería implementarlo (desplegarlo) en un entorno de explotación de Servidor Web. y, en su caso, publicarlo en un servicio de descubrimiento para ponerlo a disposición de las potenciales aplicaciones que puedan utilizarlo. Dado que estas prácticas se realizan, exclusivamente, en un entorno de desarrollo no se abordan estas cuestiones.

Ejercicio 3

Consumir el Servicio Web SOAP creado anteriormente

En este ejercicio se aborda el proceso obtener acceso al Servicio Web creado en el ejercicio anterior.

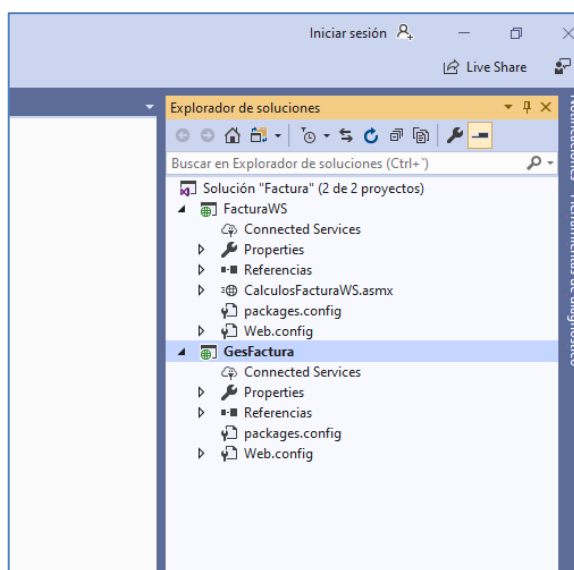
Creación de una aplicación Web cliente que consumirá un Servicio Web SOAP

En este apartado se creará un nuevo **Proyecto** vacío de aplicación Web de ASP.NET (.NET Framework), denominado *GesFactura*, dentro de la misma **Solución Factura**. En esta nueva aplicación Web se creará un Web Form que obtendrá acceso al Servicio Web creado en el ejercicio anterior. Para ello, hacer:

1. En el **Explorador de Soluciones**, seleccionar la **Solución Factura**, hacer clic sobre el botón secundario para desplegar la opción **Agregar** y seleccionar **Nuevo Proyecto...**
 - a. Seleccionar la plantilla de Proyecto **Aplicación web ASP.NET (.NET Framework)** que utiliza el lenguaje **C#** y hacer clic en **Siguiente**.
 - b. En la ventana **Configure su nuevo proyecto** introducir el nombre del nuevo Proyecto que será *GesFactura* y hacer clic en **Crear**.
 - c. Finalmente, seleccionar la opción **Vacío** y hacer clic en **Crear**.

Así, la Solución *Factura* contendrá dos Proyectos: el Proyecto *FacturaWS*, que incluye el Servicio Web y, el Proyecto *GesFactura* que representa una aplicación Web cliente que invocará las funcionalidades expuestas por el Servicio Web SOAP. Se han incluido ambos proyectos en la misma Solución porque en un entorno de desarrollo, solo es posible obtener acceso a Servicios Web que formen parte de la misma Solución de Visual Studio. En un entorno de explotación sería necesario implementar el Servicio Web en un servidor Web.

2. En el **Explorador de soluciones** puede apreciarse que el Proyecto *FacturaWS* aparece en **negrita**, porque constituye el **Proyecto de inicio** de la Solución *Factura*. Para definir el proyecto *GesFactura* como Proyecto de inicio, hacer clic sobre el botón secundario sobre su nombre y seleccionar la opción **Establecer como proyecto de inicio**. Para iniciar la depuración de un Web Form, es necesario que la aplicación Web que lo incluye sea el Proyecto de inicio de la Solución.



3. En el Proyecto *GesFactura* crear un nuevo Web Form, denominado *LineasFactura.aspx*. Y agregar los controles para obtener una apariencia de la vista Diseño similar a la siguiente.

La lista de controles agregados y sus propiedades más significativas es la siguiente:

Control	ID	Text
TextBox	txtCantidad	
TextBox	txtPrecio	
TextBox	txtDescuento	
TextBox	txtTipoIVA	
Button	btnEnviar	Enviar
Label	lblBruto	
Label	lblDescuento	
Label	lblBaseImponible	
Label	lblIva	
Label	lblTotal	



4. Iniciar la depuración del Web Form *LineasFactura.aspx* para comprobar el funcionamiento de la interfaz Web desarrollada. Debe tenerse en cuenta que para iniciar la depuración de un Web Form, es necesario que la aplicación Web que lo incluya sea el Proyecto de inicio de la Solución.

Agregar una Referencia de servicio

Una Referencia de servicio agregada a una aplicación Web establece un mecanismo de comunicación con un Servicio Web SOAP a través de una representación local. En el caso que se utilice solo el entorno de desarrollo y no se dispone de un Servidor Web **Internet Information Server (IIS)** instalado como servicio en el equipo local de desarrollo, porque se utiliza el Servidor Web integrado en el entorno de

desarrollo de Visual Studio. En estos casos, será posible agregar una Referencia de servicio que represente al Servicio Web *CalculosFacturaWS.asmx* siempre que el Servicio Web y la aplicación Web que lo consume formen parte de la misma Solución. Una Referencia de servicio en la aplicación Web *GesFactura* actúa de una forma similar a como lo haría si el Servicio Web SOAP estuviera implementado en un servidor Web de explotación. Para agregar la Referencia de servicio a Proyecto *GesFactura*, realizar las siguientes acciones:

1. Establecer el Proyecto *GesFactura* como proyecto de inicio de la Solución *Factura*.
2. En el **Explorador de soluciones**, seleccionar el Proyecto *GesFactura* y, a continuación, en el menú Proyecto, seleccionar la opción **Agregar referencia de servicio...**
3. En el cuadro de diálogo **Agregar referencia de servicio**, hacer:
 - a. Desplegar las opciones disponibles en el botón **Detectar** y seleccionar la opción **Servicios de la solución**, dado que se emplea un entorno de desarrollo.
 - b. En el cuadro **Servicios** desplegar la opción *CalculosFacturaWS.asmx* y, así mismo, desplegar la opción *CalculosFacturaWS* para recuperar la información sobre el Servicio Web SOAP para realizar los cálculos en líneas de factura que se ha desarrollado en el ejercicio anterior. A continuación, seleccionar la opción *CalculosFacturaWSSoap*. En el cuadro **Operaciones** aparecerán las funcionalidades o métodos del Servicio Web.
 - c. En el cuadro de texto Espacio de nombres, se propone un nombre para la Referencia del Servicio que podrá ser *ServiceReference1* y que puede ser válido. Este espacio de nombres constituirá la referencia local que se utilizará desde la aplicación Web para tener acceso al Servicio Web de destino.
7. Para finalizar, hacer clic en **Aceptar**.

Visual Studio incluye las referencias necesarias, accede a la descripción del servicio y genera una clase proxy que realiza la función de interfaz entre la aplicación Web cliente y el Servicio Web que consume.

Obtener acceso al Servicio Web SOAP

Una vez que se ha creado un Web Form que especifica la interfaz Web que obtendrá los datos necesarios para los cálculos a realizar y presentará los resultados y una vez que se ha creado la Referencia de servicio, a continuación, se incluirá el código lógico para obtener acceso o invocar el Servicio Web SOAP a través de su Referencia de servicio. Para ello, realizar las siguientes acciones:

1. En el archivo de código subyacente *LineasFactura.aspx.cs*, agregar el siguiente código para especificar las acciones de invocación al Servicio Web asociado al evento *Click* del control Button denominado *btnEnviar*.

```
protected void btnEnviar_Click(object sender, EventArgs e)
{
    CalculosFacturaWSSoapClient wsCalculos = new CalculosFacturaWSSoapClient();
    double w_cantidad = Convert.ToDouble(txtCantidad.Text);
    double w_precio = Convert.ToDouble(txtPrecio.Text);
    double w_descuento = Convert.ToDouble(txtDescuento.Text);
    double w_tipoIVA = Convert.ToDouble(txtTipoIVA.Text);

    // Invocación del Web Service que devuelve un objeto de tipo Calculos
    var resul = wsCalculos.CalculosLineaFactura(w_cantidad, w_precio,
                                                w_descuento, w_tipoIVA);
}
```

```
wsCalculos.Close();  
lblBruto.Text = resul.Bruto;  
lblDescuento.Text = resul.Descuento;  
lblBaseImponible.Text = resul.BaseImponible;  
lblIva.Text = resul.Iva;  
lblTotal.Text = resul.Total;  
}
```

El código anterior define una instancia de la Referencia de servicio para obtener acceso a la funcionalidad del Servicio Web SOAP. Puede apreciarse cómo el resultado de la invocación es un objeto del tipo *Calculos* que se asigna a la variable *resul*. De forma predeterminada y transparente al desarrollador se realiza un proceso de **deserialización** implícita de la instancia de la clase *Calculos*. Se conoce por deserialización el proceso de codificación de un medio de almacenamiento, como puede ser un archivo, o un buffer de memoria, en un objeto. Es, por tanto, el proceso inverso a la serialización. En el código anterior puede observarse en el código cómo, una vez enviados los datos introducidos como argumentos e invocado el método *CalculosLineaFatura()* del Servicio Web para realizar los cálculos, el método devolverá el objeto *resul* ya deserializado desde el documento SOAP recibido en la aplicación Web cliente.

2. En *LineasFactura.aspx.cs*, mediante la cláusula *using* correspondiente, agregar el espacio de nombres *ServiceReference1* para hacer referencia al Servicio Web *CalculosfacturaWS*.

```
using GesFactura.ServiceReference1;
```

3. Iniciar la depuración del Web Form *LineasFactura.aspx* para comprobar los resultados.

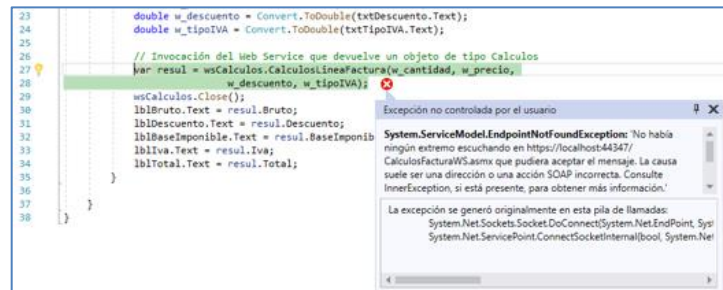
Uso de Servicio Web - Cálculos factura de un artículo

Cantidad	<input type="text" value="2"/>
Precio	<input type="text" value="100"/>
Descuento (%)	<input type="text" value="5"/>
Tipo de IVA (%)	<input type="text" value="21"/>

Resultados de los cálculos:

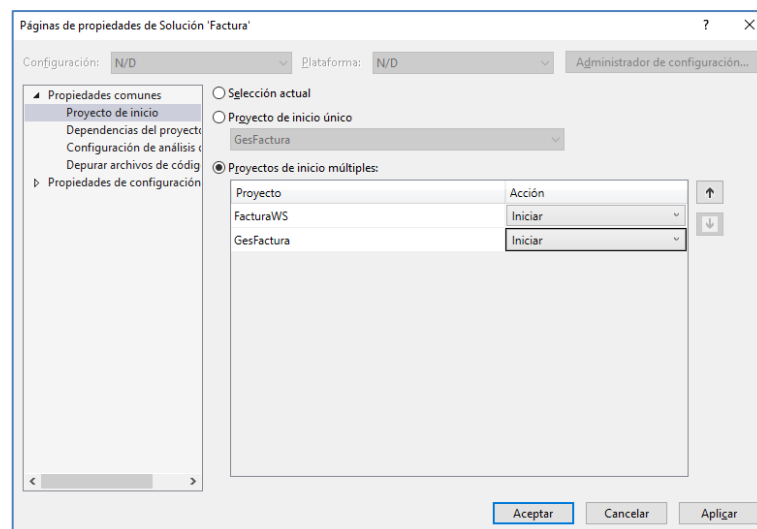
Bruto	Descuento	Base Imponible	IVA	Total
200,00 €	10,00 €	190,00 €	39,90 €	229,90 €

4. Se puede comprobar que al cerrar y al abrir nuevamente la solución *Factura*, el Servicio Web *CalculosFacturaWS* deja de funcionar y se produce el siguiente error en tiempo de ejecución.



Este error se produce porque el Servicio Web SOAP *CalculosFacturaWS* deja de ser accesible desde la aplicación Web *GesFactura* cada vez que se cierra la solución *Factura* en Visual Studio.

5. Para solucionar este problema en el entorno de desarrollo, realizar las siguientes acciones:
 - a. En el **Explorador de soluciones**, hacer clic sobre el botón secundario sobre el nombre de la solución *Factura* y seleccionar la opción **Establecer proyectos de inicio...**
 - b. En la ventana **Páginas de Propiedades de Solución 'Factura'**:
 - Seleccionar la opción **Proyectos de inicio** en el cuadro de la izquierda.
 - Seleccionar la opción **Proyectos de inicio múltiples**, en el cuadro de la derecha.
 - Seleccionar la opción **Iniciar** en el cuadro combinado **Acción** para asignarla tanto al proyecto *FacturaWS* como al proyecto *GesFactura*. Y hacer clic en **Aceptar**.



- c. En el **Explorador de soluciones**, hacer clic en el botón secundario sobre el nombre del Proyecto *FacturaWS* y seleccionar la opción **Propiedades**. A continuación, seleccionar la opción **Web** en el cuadro de la izquierda y la opción **No abrir una página. Esperar la solicitud de una aplicación externa** que está situada a la derecha.
 - d. En el **Explorador de soluciones**, hacer clic en el botón secundario sobre el nombre del proyecto *GesFactura* y seleccionar la opción **Propiedades**. A continuación, seleccionar la opción **Web** en el cuadro de la izquierda, seleccionar la opción **Página específica** que está situada a la derecha y seleccionar **LineasFactura.aspx** como página de inicio del proyecto, haciendo clic en el botón de selección correspondiente.
6. Iniciar la depuración para comprobar los resultados obtenidos.

Ejercicio 4

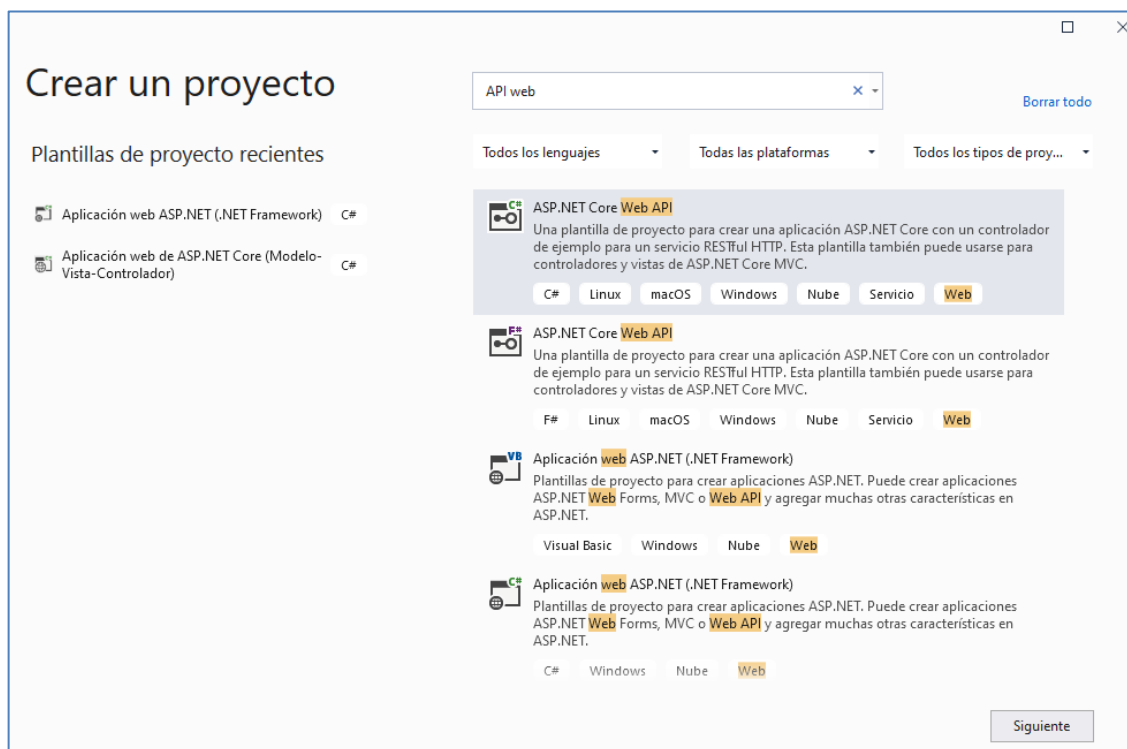
Desarrollo de una aplicación Web API (Servicios Web RESTful) basada en ASP.NET Core

La tecnología de desarrollo ASP.NET Core facilita la creación de Servicios Web RESTful (*Web Services RESTful*), a través de las **aplicaciones Web API de ASP.NET Core**. En este ejercicio se creará una nueva aplicación Web API, denominada *ApiContactos*, empleando la tecnología de desarrollo ASP.NET Core y el lenguaje de programación C#. Esta aplicación Web API facilitará la gestión de las solicitudes de información relativa a los contactos de una persona, empresa u organización. Para gestionar las solicitudes del Servicio Web, una aplicación Web API puede utilizar uno o varios controladores que derivan de la clase *ControllerBase*. El desarrollo la aplicación Web API incluirá la creación de una base de datos para almacenar los datos relativos a los contactos. Se emplea un enfoque *Code First* para el enlace a datos, de modo que la creación de la base de datos se realizará a partir del código.

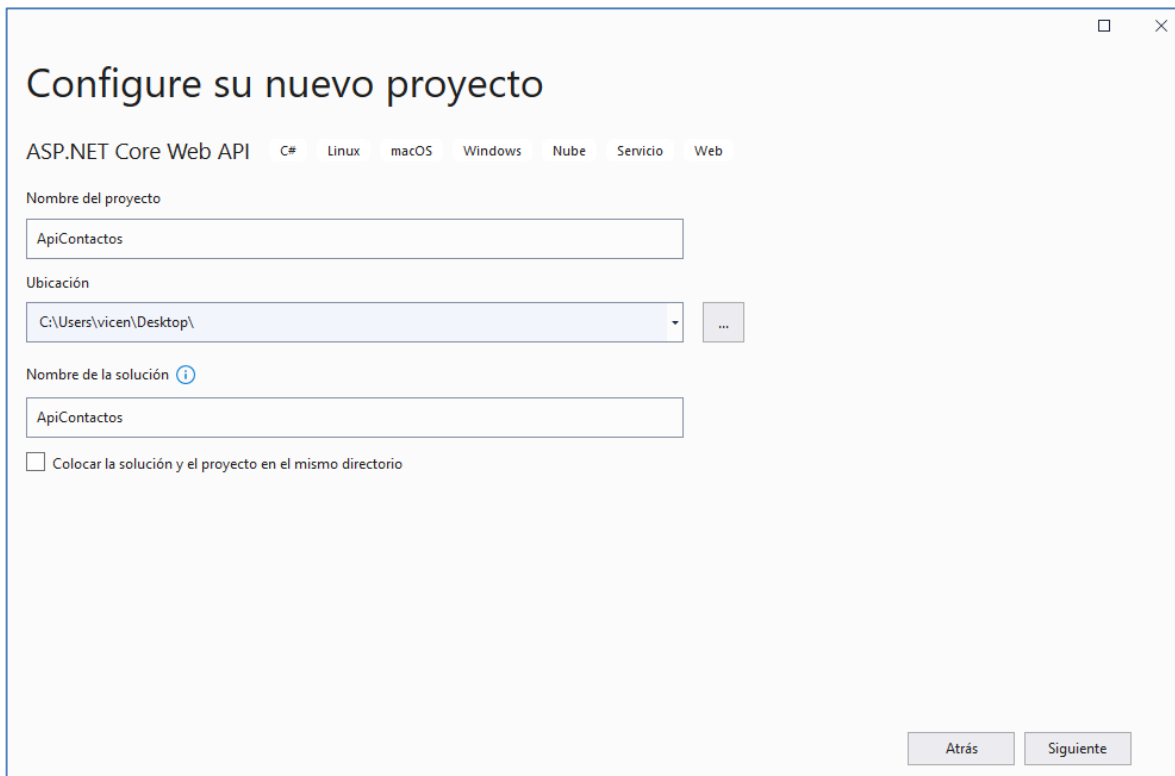
Crear una nueva aplicación Web API de ASP.NET Core

En primer lugar, se creará una aplicación Web API, también denominada aplicación API Web, utilizando la plantilla de Proyecto correspondiente. Para ello, realizar las siguientes acciones:

1. Acceder al menú **Archivo** de Visual Studio, expandir la opción **Nuevo** y seleccionar **Proyecto...**
2. Seleccionar la plantilla **ASP.NET Core Web API** que utiliza el **lenguaje de programación C#** en la ventana **Crear un Proyecto**, y hacer clic en **Siguiente**. Se facilita la búsqueda de la plantilla si se introduce el texto "API web" en el cuadro de texto de búsqueda.



3. A continuación, en la ventana **Configure su nuevo proyecto** realizar las siguientes acciones:
 - a. Introducir el **Nombre del proyecto** de aplicación Web API que será: **ApiContactos**.
 - b. Seleccionar una **Ubicación** para la Solución de Visual Studio mediante el botón ... situado a la derecha. Por ejemplo, puede utilizarse el **Escritorio** como ubicación.
 - c. En el cuadro de texto **Nombre de la solución** aparece, de manera predeterminada, el mismo nombre que se le ha dado al proyecto. En este caso, la Solución de Visual Studio y el Proyecto de aplicación Web API de ASP.NET Core tendrán el mismo nombre.
 - d. Comprobar que está desactivada la opción **Colocar la solución y el proyecto en el mismo directorio**.
 - e. Hacer clic en el botón **Siguiente**.



4. En la ventana **Información adicional**, realizar las siguientes acciones:
 - a. En el cuadro combinado **Plataforma de destino**, elegir la opción **.NET 5.0 (Actual)**. Esta opción especifica que se utilizará la versión .NET 5.0 para desarrollar la aplicación Web API denominada **ApiContactos**.
 - b. Comprobar que en el cuadro combinado **Tipo de autenticación**, está seleccionada la opción predeterminada **Ninguno**.
 - c. Comprobar que está activada la casilla de verificación **Configurar para HTTPS**.
 - d. Comprobar que está desactivada la opción **Habilitar Docker**, y que está deshabilitado el cuadro combinado **Sistema operativo de Docker**.
 - e. Comprobar que está activada la opción **Habilitar compatibilidad con OpenAPI**.
 - f. Finalmente, hacer clic en **Crear**.

Información adicional

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

Plataforma de destino ⓘ
.NET 5.0 (Actual)

Tipo de autenticación ⓘ
Ninguno

☒ Configurar para HTTPS ⓘ
☐ Habilitar Docker ⓘ

Sistema operativo de Docker ⓘ
Linux

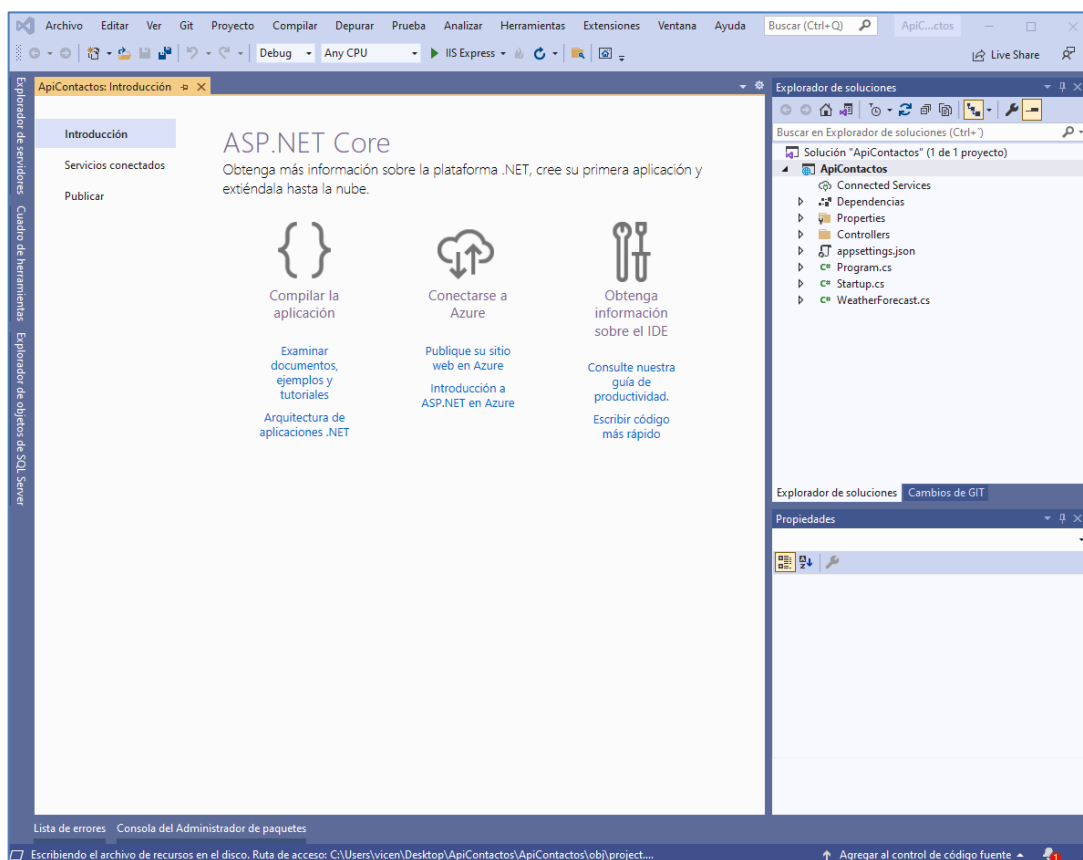
☒ Habilitar compatibilidad con OpenAPI ⓘ

Atrás Crear

Una vez finalizado el proceso de creación de la aplicación Web API basada en ASP.NET Core puede comprobarse, mediante el **Explorador de soluciones** de Visual Studio, que se habrá creado la Solución *ApiContactos* y, en su interior, el Proyecto de aplicación Web API basada en ASP.NET Core denominado también *ApiContactos*. Además, puede comprobarse que se habrá creado una estructura compleja de carpetas y archivos.

La estructura compleja de componentes o módulos que conforma una aplicación Web API de ASP.NET Core da soporte al *framework* de desarrollo de Servicios Web REST (RESTful), empleando para ello la tecnología de desarrollo de ASP.NET Core. A continuación, se describe brevemente la estructura de los archivos y las carpetas que componen una aplicación Web API basada en ASP.NET Core, tal como se muestra en el **Explorador de soluciones** de Visual Studio:

- La carpeta **/Controllers** contiene los controladores de la aplicación Web API. De manera predeterminada, un proyecto de aplicación Web API basada en ASP.NET Core incluye un controlador predeterminado para pruebas denominado *WeatherForecast*.
- La carpeta **/Dependencies** contiene las referencias a los paquetes y las librerías preinstaladas que son necesarias para la solución.
- Los componentes **/Properties**, **appsettings.json**, **Program.cs** y **Startup.cs** son carpetas y archivos de configuración de la aplicación Web API.
- La carpeta **/Models** puede contener las clases de entidades de datos y la clase del contexto de datos del modelo.
- La carpeta **/Migrations** contiene los archivos relacionados con la realización de migraciones.



Debe tenerse en cuenta que REST (*Representational State Transfer*) es una propuesta de diseño para Arquitecturas del software Orientadas a los Servicios (SOA), en las que los servicios se consideran recursos y la comunicación entre el cliente y el servidor no tiene estado. Esto quiere decir que un Servicio Web REST pierde todos los datos entre las solicitudes realizadas desde la aplicación cliente, de modo que los datos no se conservan entre solicitudes, lo que significa que cada solicitud es independiente y está desconectada del resto. La guía de diseño REST propone una arquitectura del software más simple que los Servicios Web basados en SOAP. Para ello, las aplicaciones RESTful emplean los métodos del protocolo HTTP para comunicarse con el Servicio Web.

El desarrollo de aplicaciones Web API de ASP.NET Core cumple con las condiciones y restricciones establecidas en la guía de diseño REST. Por este motivo, los servicios creados por las aplicaciones Web API de ASP.NET Core son Servicios Web RESTful. Las características más significativas de los Servicios Web RESTful, también denominados Servicios API REST o sencillamente API REST, son las siguientes:

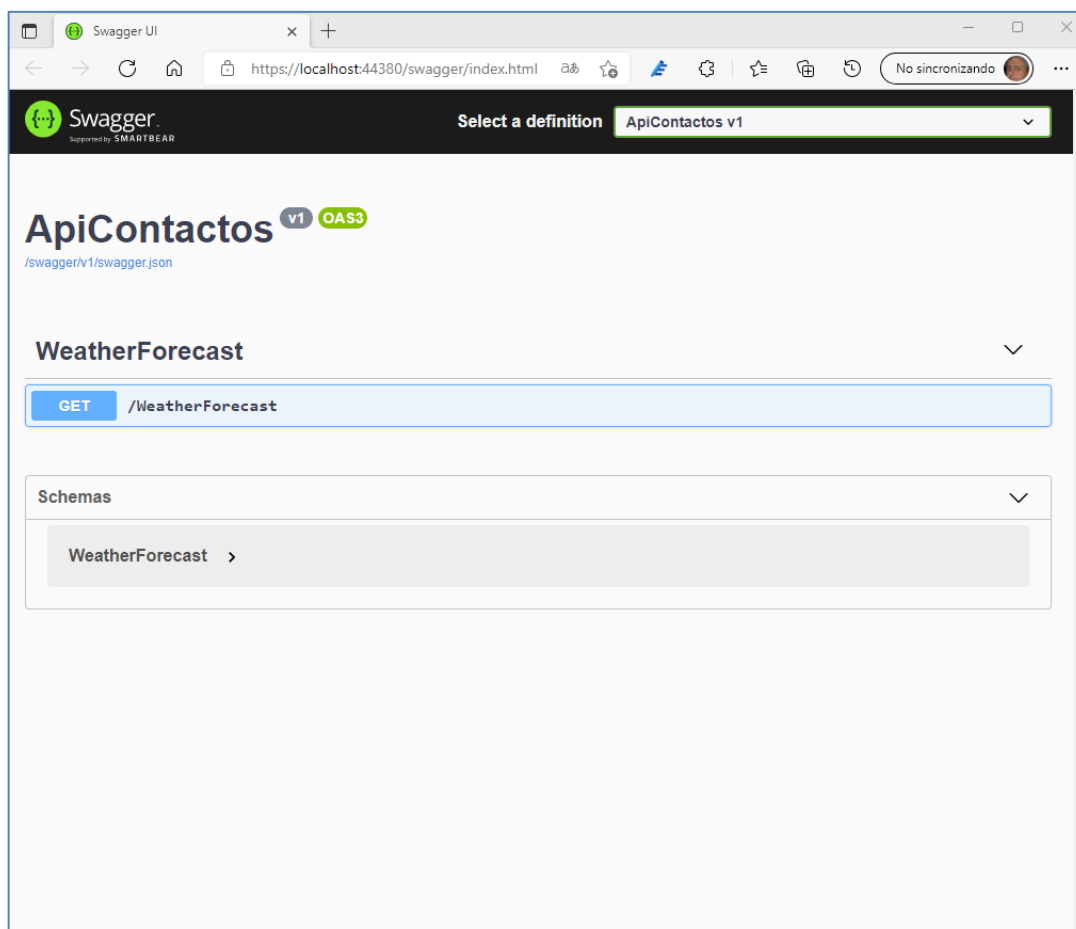
- Los Servicios Web RESTful están asociados a la información. Permiten realizar operaciones como: listar, crear, leer, actualizar y borrar información almacenada en bases de datos.
- Se emplean los métodos de solicitud del protocolo HTTP para acceder a las operaciones de listar, crear, leer, actualizar y borrar información del Servicio Web. Y, se utiliza una dirección URI para poder acceder a estas operaciones.
- Usualmente, devuelven la información en formato JSON.
- Retornan códigos de respuesta HTML, por ejemplo: 200, 201, 404, etc.

Utilizando el **Explorador de archivos** de Windows se puede comprobar que se habrá creado la carpeta de Solución *ApiContactos* en la ubicación deseada. Esta carpeta contendrá el archivo de Solución *ApiContactos.sln*, así como la carpeta del Proyecto *ApiContactos*, que a su vez contendrá los archivos y carpetas correspondientes al Proyecto de aplicación Web API basada en ASP.NET Core.

Probar el Proyecto de aplicación Web API basada en ASP.NET Core que se ha creado

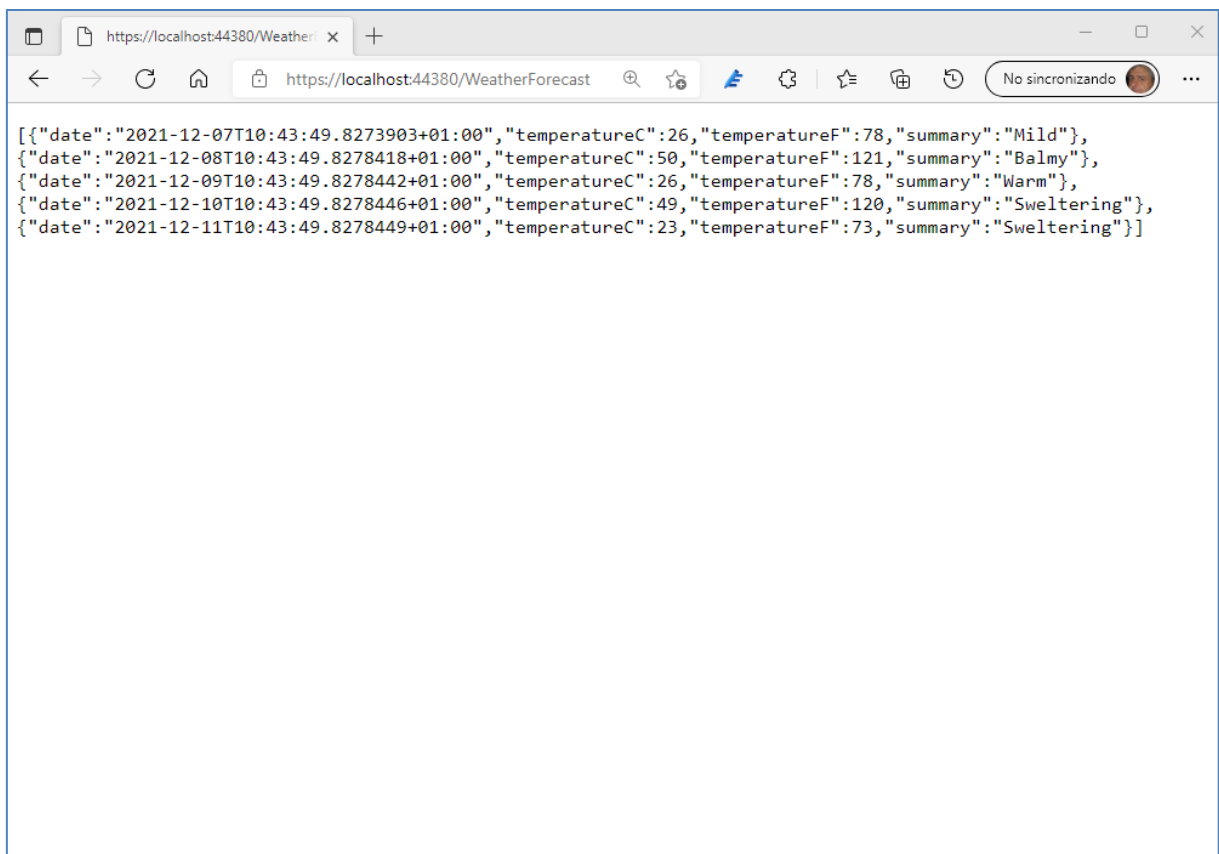
De manera predeterminada, la plantilla de proyecto de ASP.NET Core Web API, que ha sido utilizada para crear la aplicación Web API *ApiContactos*, genera una API compatible con *Swagger* denominada *WeatherForecast*. Esta API predeterminada permite realizar pruebas de ejecución de la aplicación Web API. La especificación *Swagger*, también denominada *Open API*, se utiliza para describir los Servicios API REST, de modo que permite a los usuarios de una API conocer las capacidades que ofrece, sin necesidad de tener acceso directo al código fuente. En las aplicaciones Web API de ASP.NET Core se puede usar *Swagger* para generar la documentación y las páginas Web de ayuda de la API.

El Proyecto de aplicación Web API creado anteriormente ya puede ejecutarse, para ello basta con **Iniciar la depuración**. Al iniciar la depuración por primera vez, puede ocurrir que se soliciten algunas confirmaciones de seguridad. Si es el caso, responder afirmativamente a las preguntas de seguridad sobre si se confía y si se instala el certificado SSL (*Secure Sockets Layer*) de IIS Express para *localhost*.



Como se observa en la ilustración anterior, al iniciarse la depuración se abre la página predeterminada de *Swagger* cuya dirección de enrutamiento es: `/swagger/index.html`. Esta página permite realizar las pruebas de ejecución del proyecto de aplicación Web API. Para ello, realizar las siguientes acciones:

1. Hacer clic en el botón **GET**.
2. A continuación, hacer clic en el botón **Try it out**.
3. Finalmente, hacer clic en el botón **Execute**. Podrá observarse que la página muestra las respuestas de la ejecución, que son las siguientes: el comando de CURL para probar la API *WeatherForecast*, la dirección URL (*Request URL*) para probar la API *WeatherForecast*, el código de la respuesta del servidor (*Server response*) y un cuadro combinado en el que se puede seleccionar el tipo de medio (*Media type*) para mostrar un valor de ejemplo.
4. Para probar la API *WeatherForecast* que se genera de manera predeterminada, copiar y pegar el valor de la dirección URL que aparece en la sección **Request URL** en el cuadro de solicitud de direcciones del navegador Web. El formato de la dirección Web a copiar y pegar será similar a la siguiente: `https://localhost:<puerto>/WeatherForecast`, aunque será necesario ajustar el valor del puerto en cada caso, para poder obtener los resultados. En el navegador se obtendrá un documento JSON similar al que se muestra en la siguiente ilustración, en el que se muestra el resultado de la solicitud de información a la API Web.



5. Finalmente, cerrar en navegador para finalizar las pruebas de ejecución del proyecto de aplicación Web API. Si fuera necesario, detener la depuración desde Visual Studio.

A continuación, se inician las tareas de desarrollo de la API Web denominada *ApiContactos*.

Actualización de launchUrl

Una vez comprobado el funcionamiento de la aplicación Web API, se va a eliminar *Swagger* porque no va a utilizarse la API predeterminada *WeatherForecast*. Para ello, realizar las siguientes acciones:

1. Abrir el archivo *launchSettings.json* almacenado en la carpeta */Properties* del Proyecto.
2. En el código de marcado *json* del archivo *launchSettings.json*, sustituir el valor "swagger" por "api/Contactos", tal como aparece resaltado a continuación.

```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:53441",
      "sslPort": 44380
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      // "launchUrl": "swagger",
      "launchUrl": "api/Contactos",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "ApiContactos": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": true,
      // "launchUrl": "swagger",
      "launchUrl": "api/Contactos",
      "applicationUrl": "https://localhost:5001;http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

3. Guardar los cambios realizados y cerrar el archivo *launchSettings.json*.

Agregar el modelo a la aplicación Web API

A continuación, se agregará el modelo a la aplicación Web API. Un modelo es un conjunto de clases que representan los datos que maneja la aplicación. En este caso, se creará una única clase de datos y la clase del contexto de datos. Estas dos clases, que constituirán el Modelo de la aplicación Web API, se utilizarán junto con *Entity Framework Core (EF Core)* para poder trabajar con una base de datos desde el modelo. *EF Core* es un *framework ORM (Object-Relational Mapping)* que se utiliza para administrar el acceso a los datos almacenados y simplificar el código de acceso a datos.

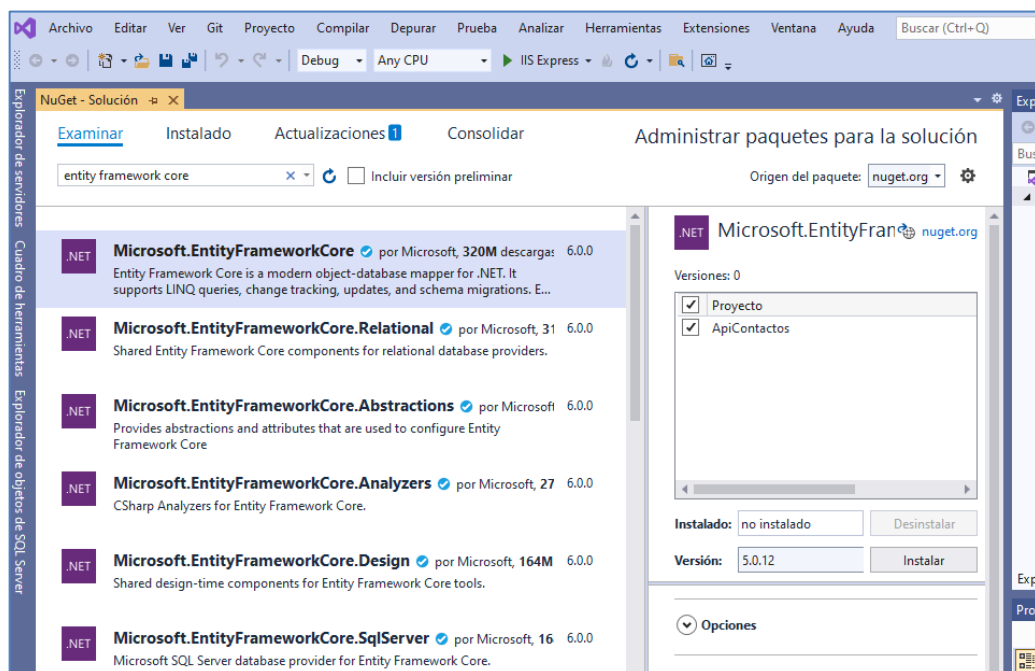
Para crear el modelo de la aplicación Web API basada en ASP.NET Core *ApiContactos* mediante un enfoque de acceso a datos *Code First*, realizar las siguientes acciones:

1. Como paso previo, es necesario instalar los paquetes de *NuGet* específicos que permiten utilizar las funcionalidades de *EF Core*. Las herramientas *NuGet* proporcionan la capacidad de producir y consumir paquetes que aportan funcionalidades de desarrollo que son útiles para la Solución. Estos paquetes están disponibles en el repositorio de paquetes *NuGet* (*NuGet Gallery*) que puede ser utilizado por los desarrolladores y consumidores de paquetes. En la aplicación Web API *ApiContactos* es necesario instalar los siguientes paquetes de *NuGet*:

- *Microsoft.EntityFrameworkCore*
- *Microsoft.EntityFrameworkCore.SqlServer*
- *Microsoft.EntityFrameworkCore.Tools*
- *Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore*
- *Microsoft.VisualStudio.Web.CodeGeneration.Design*

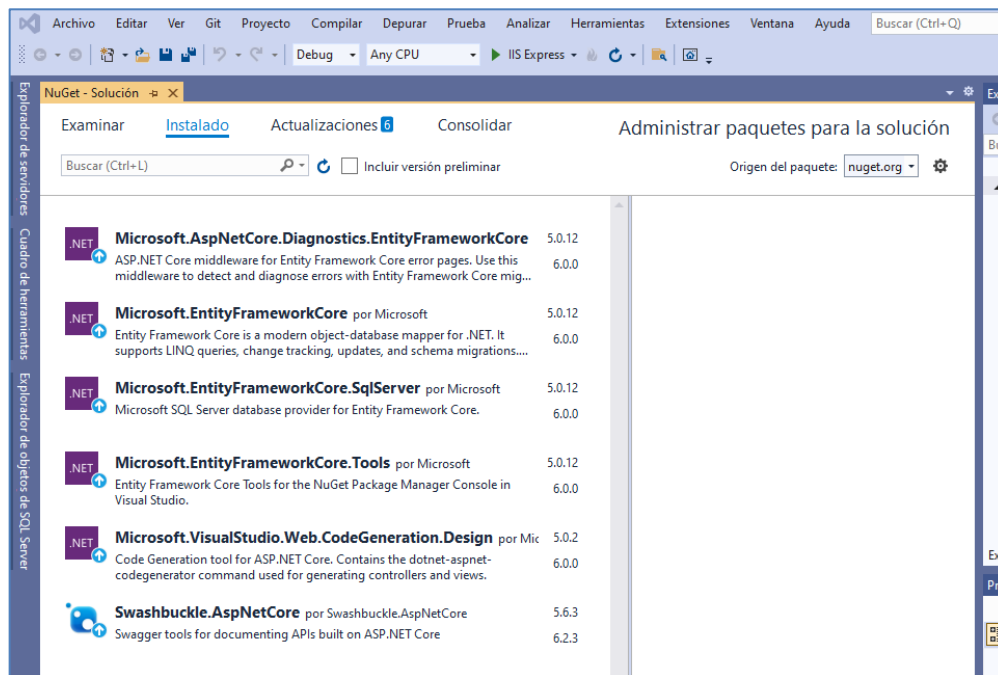
Para instalar estos paquetes, realizar las siguientes acciones:

- a. Seleccionar el menú **Herramientas** en la barra de menú de Visual Studio, desplegar la opción **Administrador de paquetes NuGet** y, a continuación, seleccionar la opción **Administrar paquetes NuGet para la solución...**
- b. En la ventana **NuGet – Solución**, seleccionar la opción **Instalado** y comprobar que no están instalados los paquetes enumerados anteriormente.
- c. Para instalar cada uno de los paquetes en la Solución, seleccionar la opción **Examinar** en la ventana **NuGet – Solución** y, a continuación, buscar el paquete empleando el cuadro de texto de búsqueda, seleccionar el paquete en los resultados de búsqueda y marcar la casilla *ApiContactos* y hacer clic en el botón **Instalar** del cuadro de la derecha.



Si se está utilizando la versión .NET 5.0, entonces se deberá instalar la versión 5.X más reciente de cada uno de los paquetes de *NuGet* necesarios para la Solución.

- d. Finalmente, en la ventana **NuGet – Solución**, seleccionar de nuevo la opción **Instalado** para comprobar que han sido instalados los paquetes necesarios en la Solución. El resultado mostrará una lista similar a la que se muestra a continuación.

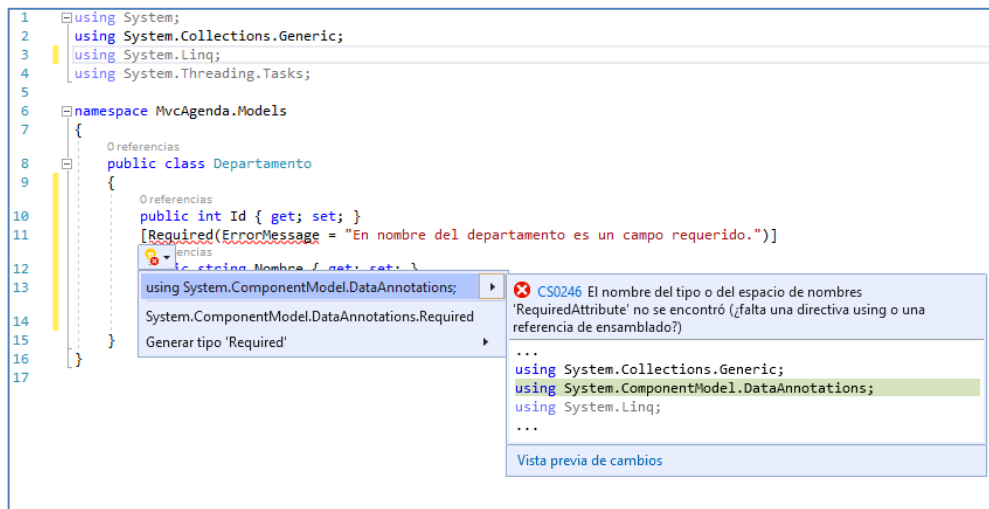


El Administrador de paquetes *NuGet* administra la instalación de los paquetes de la tecnología .NET cuyo uso sea necesario para la Solución de Visual Studio.

2. Se utilizará una clase de datos para representar la única entidad de datos del modelo. Para crear esta **clase de datos**, realizar las siguientes acciones:
- En primer lugar, se agregará la carpeta */Models* al Proyecto. Para ello, en el **Explorador de soluciones** hacer clic en el botón derecho del ratón sobre el nombre del Proyecto *ApiContactos*, desplegar la opción **Agregar**, seleccionar la opción **Nueva carpeta** y, finalmente, introducir el texto *Models* como nombre de la nueva carpeta.
 - En la carpeta */Models*, crear un archivo de clase denominado *Contacto.cs*. Para ello, en el **Explorador de soluciones**, hacer clic en el botón derecho sobre la carpeta */Models*, desplegar la opción **Agregar** y seleccionar la opción **Clase...** A continuación, introducir el nombre de la clase y añadir el siguiente código al archivo de clase creado:

```
public class Contacto
{
    public long Id { get; set; }
    [Required(ErrorMessage = "El nombre del empleado es un campo requerido.")]
    public string Nombre { get; set; }
    public string Telefono { get; set; }
    public string Email { get; set; }
    public bool Activado { get; set; }
}
```


Como puede apreciarse en el código de la clase anterior, se han añadido anotaciones de validación de datos o *DataAnnotations* para incluir características específicas a algunas propiedades. Al añadir el código se podrá comprobar que se subrayarán en rojo las anotaciones de validación, indicando un error. Este error se produce porque falta añadir una directiva *using* que referencee el espacio de nombres correspondiente al uso de *DataAnnotations*. Para solucionarlo, situar el puntero del ratón sobre una de las anotaciones de validación subrayada en rojo, desplegar las opciones del icono en forma de bombilla y seleccionar la primera opción propuesta para resolver el error:



Evidentemente, también podrían haberse solucionado los errores anteriores si se hubiera escrito directamente la línea de código siguiente mediante la instrucción *using* correspondiente al principio del código de la clase.

```
using System.ComponentModel.DataAnnotations;
```

De manera predeterminada y atendiendo a la convención de nombres establecida, *EF Core* interpreta que una propiedad de una clase de datos que se denomina *Id* es **clave principal**. Así, por ejemplo, en el código anterior la propiedad *Id* de la clase de datos *Contacto* se convertirá en la columna clave principal de la tabla correspondiente.

3. A continuación, se creará la **clase del contexto de datos** que coordina la funcionalidad y el comportamiento de *EF Core* y del modelo de datos en la aplicación Web API. Para crear la clase del contexto de datos, crear un archivo de clase denominado *ApiContactosContexto.cs* en la carpeta */Models* y añadir el siguiente código:

```
public class ApiContactosContexto : DbContext
{
    public ApiContactosContexto(DbContextOptions<ApiContactosContexto> options)
        : base(options)
    {
    }

    public DbSet<Contacto> Contactos { get; set; }
}
```

Tal como se ha realizado anteriormente, sobre los elementos del código que se subrayan en rojo, resolver los errores añadiendo las cláusulas *using* que correspondan en cada caso. Como puede apreciarse en el código anterior, **la clase del contexto de datos es una clase derivada de la clase *DbContext*** y especifica los conjuntos de entidades de datos que maneja la aplicación Web API. Además, de manera predeterminada y atendiendo a la convención de nombres establecida, *EF Core* interpreta que los nombres de las propiedades *DbSet* se usan como **nombres de tabla**. Así, el código de la clase del contexto define un único conjunto de entidades de datos denominado *Contactos* que estará formada por una colección de objetos de tipo *Contacto*. El nombre de la tabla correspondiente será *Contactos*.

4. En el menú **Archivo** de Visual Studio, seleccionar la opción **Guardar todo**.
5. Seleccionar la opción **Compilar ApiContactos** del menú **Compilar** de Visual Studio para compilar el código de las clases añadidas y poder comprobar que no existen errores. El resultado de la compilación se muestra en la ventana de **Salida** de Visual Studio.

Registrar el contexto de base de datos de la aplicación Web API

Todos los servicios de una aplicación basada en ASP.NET Core, como es, por ejemplo, el contexto de base de datos, deben registrarse en el contenedor de Inyección de Dependencias durante el inicio de la aplicación Web API. El contenedor de Inyección de Dependencias (DI) proporciona el servicio a los controladores y permite conectar fácilmente las clases y sus dependencias. Además de registrar el contexto de base de datos, se van a aprovechar los cambios en el código del archivo *Startup.cs* para eliminar las llamadas a *Swagger*, de modo que se va a eliminar completamente el uso predeterminado de *Swagger* para describir y documentar la aplicación Web API. Para registrar el contexto de la base de datos al iniciar la aplicación Web API y eliminar el uso predeterminado de *Swagger* en la aplicación Web API, realizar las siguientes acciones:

1. Abrir el archivo de clase *Startup.cs* que se encuentra ubicado en la carpeta raíz del proyecto.
2. Al final del método *ConfigureServices()*, añadir y modificar el código que aparece resaltado a continuación.

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

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers();

        services.AddDbContext<ApiContactosContexto>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("ApiContactos")));

        // Se comenta para eliminar el uso de Swagger
        // services.AddSwaggerGen(c =>
        // {
        //     c.SwaggerDoc("v1", new OpenApiInfo { Title = "ApiContactos", Version = "v1" });
        // });
    }
}
```

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();

        // Se comenta para eliminar el uso de Swagger
        // app.UseSwagger();
        // app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "ApiContactos v1"));
    }

    app.UseHttpsRedirection();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

Tal como se ha realizado anteriormente, sobre los elementos del código que se subrayan en rojo, resolver los errores añadiendo las cláusulas *using* que correspondan en cada caso. En el código anterior, puede apreciarse que se comentan ciertas líneas de código para eliminar las llamadas predeterminadas a *Swagger* para describir y documentar la aplicación Web API. Y, también puede observarse, cómo se realiza el registro del contexto de datos denominado *ApiContactosContexto* que almacena la información que maneja la aplicación Web API. En este registro, se especifica el nombre de la cadena de conexión que va a ser utilizada para conectar este contexto de datos con una base de datos. Puede observarse que el nombre de la cadena de conexión es *ApiContactos*.

Especificar la cadena de conexión de la base de datos

En el archivo *appsettings.json* se especifican las características de la cadena de conexión que enlaza la aplicación Web API con la base de datos correspondiente. Para especificar las características de la cadena de conexión denominada *ApiContactos*, abrir el archivo *appsettings.json* que se encuentra ubicado en la carpeta raíz del proyecto y añadir el código que aparece resaltado a continuación.

```
{
  "ConnectionStrings": {
    "ApiContactos":
    "Server=(localdb)\\mssqllocaldb;Database=ApiContactosBD;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

En el código anterior puede observarse cómo se han especificado las características de la cadena de conexión *ApiContactos*, de manera que se utilizará la instancia del servidor de base de datos de SQL Server denominada *(localdb)\mssqllocaldb* y se empleará el archivo de base de datos de SQL Server *ApiContactosBD.mdf* para almacenar la información. La ubicación predeterminada del archivo de base de datos de SQL Server será la carpeta raíz del usuario actual: */usuarios/<usuario_actual>*.

Migración inicial para crear la base de datos

En este momento del proceso de desarrollo de una aplicación Web API basada en ASP.NET Core se va a crear la base de datos, para lo cual se va a realizar una **Operación de Migración**. Las migraciones son una característica de *EF Core* que incluyen el uso de un conjunto de herramientas que permiten crear y actualizar el esquema de una base de datos para que corresponda exactamente con la definición de las clases del modelo. En todo momento, debe existir una correspondencia entre la estructura del modelo y el esquema de la base de datos. Si esta correspondencia no existe, entonces se produce un error en tiempo de ejecución al intentar acceder a la base de datos. Para crear la base de datos a partir de la definición de las clases del modelo mediante una migración inicial, realizar las siguientes acciones:

1. Como paso previo, conviene compilar el proyecto, mediante la opción **Compilar ApiContactos** del menú **Compilar**. Es importante comprobar que no se producen errores de compilación.
2. En el menú **Herramientas**, desplegar la opción **Administrador de paquetes NuGet** y, a continuación, seleccionar la opción **Consola del Administrador de paquetes**.
3. En la **Consola del Administrador de paquetes (PCM)** ejecutar el siguiente comando:

```
PM> Add-Migration Migracion_Inicial
```

El comando **Add-Migration** genera un archivo de migración en la carpeta */Migrations* del proyecto con el nombre de la migración especificado en el primer argumento del comando. Al tratarse de la migración inicial, la clase generada en la carpeta */Migrations* contendrá el código necesario para crear la base de datos, considerando las clases de datos del modelo. Dado que se ha definido un único modelo en la aplicación, cuya clase del contexto de datos se denomina *ApiContactosContexto*, entonces no es necesario utilizar la opción *-context* del comando para especificar el nombre de la clase del contexto de datos a migrar.

4. En la **Consola del Administrador de paquetes** ejecutar el siguiente comando:

```
PM> Update-Database
```

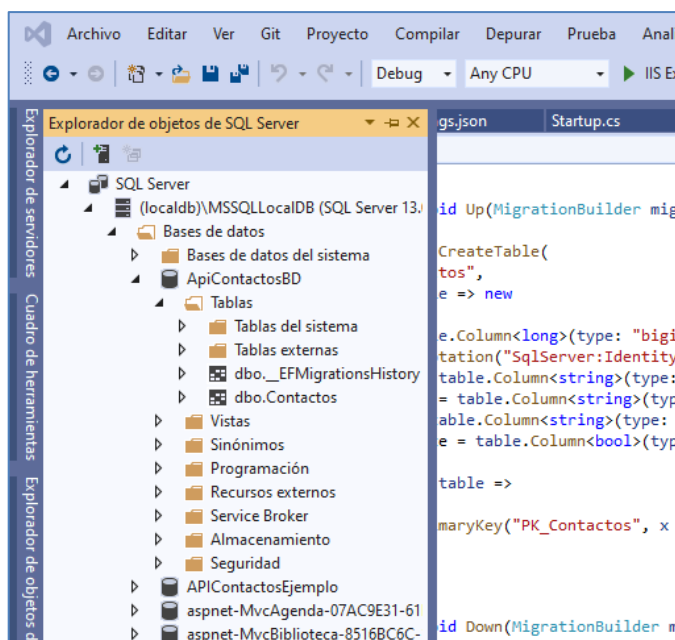
El comando **Update-Database** actualiza el esquema de la base de datos a la migración más reciente. En este caso, ejecuta el método *Up()* del archivo de migración *Migracion_Inicial.cs* que se habrá creado en la carpeta */Migrations*, como consecuencia de la ejecución de la instrucción *Add-Migration* anterior. Al ejecutar el comando **Update-Database** se creará la base de datos que almacena la información que maneja la aplicación Web API.

Si en un futuro, fuera necesario modificar la estructura de los datos que maneja la aplicación Web API, entonces será necesario modificar en el código de las clases del modelo y, a continuación, realizar una nueva operación de migración para producir reflejo de las modificaciones en la base de datos y de ese modo, actualizar el esquema de la base de datos a partir del código del modelo (*Code First*).

Comprobar la creación de la base de datos y poder trabajar con ella

Para comprobar que la base de datos se ha creado correctamente, realizar las siguientes acciones:

1. En el menú **Ver**, abrir **Explorador de objetos de SQL Server** (SSOX). Podrá comprobarse que sobre la ventana correspondiente puede elegirse la opción **Ocultar automáticamente** para hacer que quede disponible en la barra vertical izquierda de Visual Studio.
2. Desplegar la opción **SQL Server**. A continuación, desplegar la opción **(localdb)\MSSQLLocalDB** y desplegar la opción **Bases de datos**.
3. Para acceder a los objetos de la base de datos, desplegar la opción correspondiente al nombre del archivo de la base de datos de SQL Server que está definido en el archivo *appsetting.json* y que, en este caso, es *ApiContactosBD*.

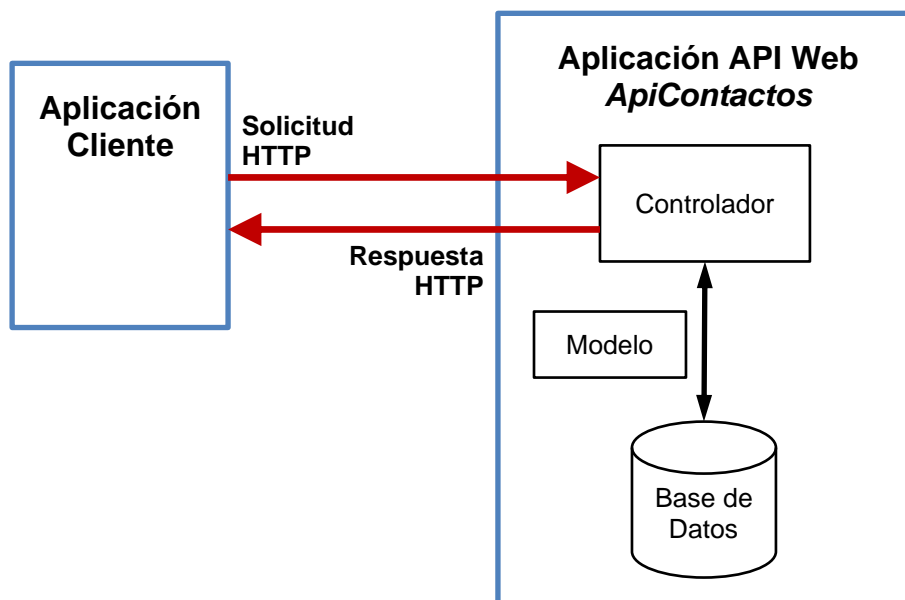


4. Al desplegar la opción **Tablas**, puede observarse que se muestra la tabla **dbo.Contactos**.
5. Para visualizar la información almacenada, hacer clic con el botón derecho sobre el nombre de la tabla y seleccionar la opción **Ver datos**. La tabla no contiene ninguna información.
6. También se puede visualizar la información de diseño de la tabla. Para ello, hacer clic con el botón derecho sobre el nombre de la tabla y seleccionar la opción **Diseñador de vistas**. Es importante tener presente que no se debe modificar el diseño de las tablas directamente sobre la base de datos. Para modificar el esquema de la base de datos, deberá modificarse el código de las clases del modelo y, a continuación, realizar una nueva operación de migración.
7. De modo predeterminado, el archivo de base de datos de SQL Server que almacena los datos se crea en la capeta raíz del usuario actual, que en Windows es: *c:/Usuarios/{usuario actual}*. Mediante el **Explorador de archivos de Windows**, acceder a esta ubicación para comprobar que existen los dos archivos correspondientes, cuyo nombre coincide con el establecido en la definición de la cadena de conexión del archivo *appsettings.json* y cuyas extensiones de nombre son: *.mdf* y *.ldf*. Estos dos archivos son: *ApiContactosBD.mdf* y *APIContatosBD.ldf*.

Crear un controlador para manejar la información almacenada en la tabla Contactos

En este apartado del ejercicio se desarrollará el procesamiento que permitirá gestionar las solicitudes de información del Servicio Web REST o aplicación Web API sobre cada una de las entidades de datos que forman el modelo y que representan los datos almacenados en la base de datos. En las aplicaciones Web API de ASP.NET Core se utilizan controladores para gestionar las solicitudes de información al Servicio Web REST sobre cada una de las entidades de datos del modelo.

La aplicación Web API *ApiContactos* que se está desarrollando, facilitará el acceso a los recursos u operaciones de información disponibles en el Servicio Web relativas a los contactos de una persona, empresa u organización, a través de las solicitudes URI correspondientes que se realizan desde otra aplicación, denominada aplicación cliente. Las respuestas del Servicio Web devuelven la información solicitada, en formato JSON, hacia la aplicación cliente. Tanto las solicitudes como las respuestas de información se realizan a través del protocolo HTTP.



A continuación, se va a desarrollar un controlador en la aplicación Web API *ApiContactos* para poder gestionar las solicitudes URI que se reciban desde una aplicación cliente. En este caso, solo se va a desarrollar un controlador, porque solo se ha definido una clase de datos en el modelo, la clase *Contacto*. Este controlador, que se denominará *Contactos*, incluirá las acciones que expresen cada una de las operaciones de información que se ofrecerán a las aplicaciones cliente para su consumo.

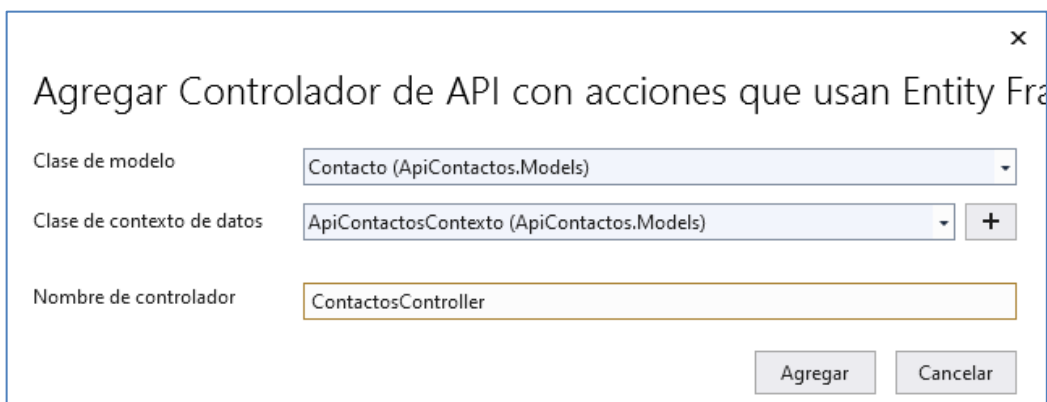
Los recursos u operaciones de información que ofrecerá el controlador *Contactos* de la aplicación Web API permitirán listar, crear, leer, actualizar y borrar la información relativa a los contactos de una persona u organización. Para acceder a cada una de las acciones del controlador se utiliza **una solicitud o petición URI (URL)** y se emplean los **métodos o verbos de solicitud del protocolo HTTP** (GET, POST, PUT y DELETE) para definir la acción que se desea realizar sobre el recurso. El acceso a los recursos que ofrece o expone el controlador *Contactos* permitirá expresar un procesamiento de tipo CRUD (*Create Read Update and Delete*) sobre la entidad de datos *Contacto*.

La aplicación Web API *ApiContactos* se encargará de atender, gestionar y responder las solicitudes de información relativas a los contactos de una persona, empresa u organización. El controlador *Contactos* de la aplicación Web API ofrecerá los siguientes recursos u operaciones de información:

Recursos REST	Descripción	Cuerpo de la solicitud	Cuerpo de la respuesta
GET /api/Contactos	Obtener la lista de todos los contactos	Ninguno	Lista de contactos
GET /api/Contactos/{id}	Obtener un contacto identificado en la URI	Ninguno	Contacto
POST /api/Contactos	Añadir un nuevo contacto	Contacto	Contacto
PUT /api/Contactos/{id}	Actualizar un contacto existente	Contacto	Ninguno
DELETE /api/Contactos/{id}	Eliminar un contacto existente	Ninguno	Ninguno

Para crear un controlador de la clase de datos *Contacto*, realizar las siguientes acciones:

1. En primer lugar, compilar el proyecto para poner las clases del modelo a disposición de los controladores que se van a crear y comprobar que el resultado de la compilación no devuelve ningún error. Para ello, seleccionar la opción **Compilar ApiContactos** del menú **Compilar**.
2. Para crear un controlador para la clase de datos *Contacto*, en el **Explorador de soluciones** hacer clic en el botón derecho sobre la carpeta */Controllers*, desplegar la opción **Agregar** y, a continuación, seleccionar la opción **Nuevo elemento con Scaffold...**
3. En la ventana **Agregar nuevo elemento con scaffolding**, seleccionar la opción **Controlador de API con acciones que usan Entity Framework** y hacer clic en **Agregar**.
4. En la ventana **Agregar Controlador de API con acciones que usan Entity Framework**, hacer:
 - a. Seleccionar *Contacto* como **Clase de modelo**.
 - b. Seleccionar *ApiContactosContexto* como **Clase de contexto de datos**.
 - c. Introducir *ContactosController* como **Nombre de controlador**.
 - d. Hacer clic en **Agregar**.



- Finalizado el trabajo del asistente, se habrá creado el controlador *ContactosController.cs* en la carpeta */Controllers*. Este controlador incluye las acciones que implementan las operaciones de manipulación de datos relativas a la entidad de datos *Contacto* del modelo de la aplicación Web API. Revisar el código del controlador para comprobar que se han especificado las acciones correspondientes a las operaciones de información del Servicio Web RESTful, tal como se han descrito en la tabla anterior. Se recomienda analizar con detenimiento el código para comprender su uso, funcionalidad y utilidad. Además, puede apreciarse en el código que la clase *ContactosController* deriva de la clase *ControllerBase* y que se especifica el atributo de filtro [*ApiController*]. Este atributo indica que el controlador responde a las solicitudes URI correspondientes de la API Web. Además, también puede apreciarse en el código que se utiliza una inyección de dependencia para insertar en contexto de datos, *ApiContactosContexto*, en el controlador. En efecto, se utiliza la variable privada *_context* para representar el contexto de datos en el código de cada una de las acciones CRUD del controlador.
- A continuación, modificar el código del método de acción *PostContacto()* para usar el operador *nameof()*, tal como aparece resaltado a continuación:

```
// POST: api/Contactos
[HttpPost]
public async Task<ActionResult<Contacto>> PostContacto(Contacto contacto)
{
    _context.Contactos.Add(contacto);
    await _context.SaveChangesAsync();

    // return CreatedAtAction("GetContacto", new { id = contacto.Id }, contacto);
    return CreatedAtAction(nameof(GetContacto), new { id = contacto.Id }, contacto);
}
```

El operador *nameof()* del lenguaje C# se suele utilizar para evitar especificar de forma rígida, mediante el valor literal del nombre, la acción final en el *return()* de un método de acción. Y además, puede observarse que se utiliza la llamada al método *CreatedAtAction()* para devolver un código de estado HTTP 201 como respuesta, cuando el método de acción *PostContacto()* se ha ejecutado correctamente, una vez que se ha añadido un nuevo contacto a la base de datos.

- Guardar el controlador *Contactos*.

Realizar las pruebas de funcionamiento del controlador *Contactos* de la aplicación Web API

En este punto del desarrollo, se puede ya probar el funcionamiento del controlador *Contactos* de la aplicación Web API *ApiContactos*.

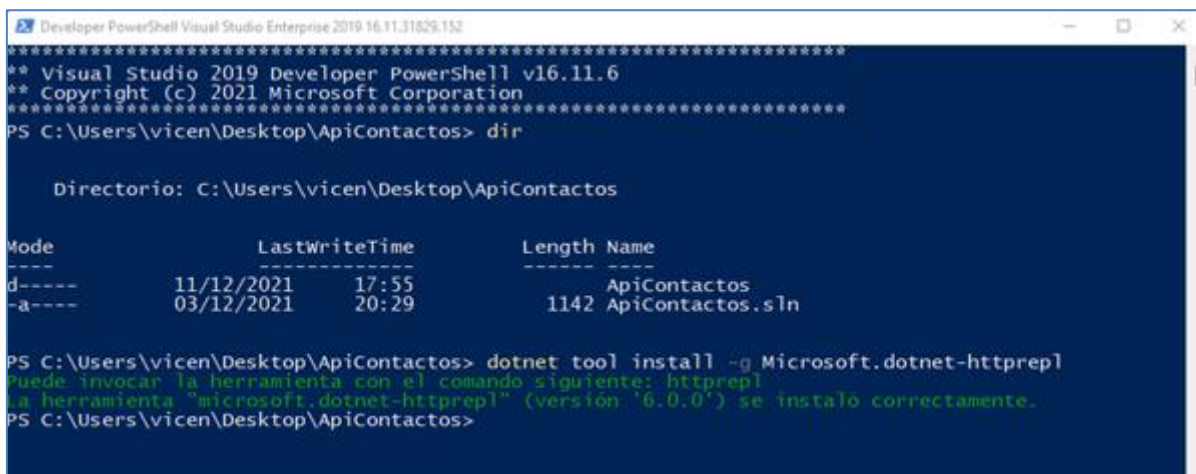
Las pruebas de ejecución de la Web API RESTful pueden realizarse desde una aplicación cliente. Sin embargo, cuando se está trabajando en un entorno de desarrollo, en lugar de desarrollar una aplicación Web cliente que permita realizar las solicitudes HTTP que consuman las acciones u operaciones de información de la aplicación Web API que se está desarrollando, se suelen utilizar herramientas de utilidad que permiten realizar las solicitudes HTTP para probar las aplicaciones Web API, u otros tipos de Servicios Web RESTful. De este modo, se pueden obtener y ver los resultados obtenidos por las respuestas a las solicitudes realizadas. Estas herramientas de utilidad facilitan la realización de pruebas de las acciones de las aplicaciones Web API. Algunas de estas herramientas de utilidad para probar las aplicaciones Web API basadas en ASP.NET Core son: HttpRepl, Postman, etc.

En este ejercicio se va a utilizar la herramienta HttpRepl. Se trata de una herramienta de línea de comandos ligera, multiplataforma y compatible con la tecnología .NET Core. HttpRepl se usa para realizar solicitudes HTTP con el fin de probar las aplicaciones Web API basadas en ASP.NET Core, así como otros tipos de Web API y Servicios Web REST, y poder ver los resultados. Permite probar las aplicaciones Web API hospedadas en cualquier entorno, incluidos *localhost* y *Azure App Service*.

Para probar el funcionamiento del controlador *Contactos* de la aplicación Web API *ApiContactos*, realizar las siguientes acciones:

1. Como paso previo, es necesario instalar la herramienta **http-repl**. Para ello, hacer:
 - a. Abrir un terminal de línea de comandos de PowerShell de Windows. Para ello, desplegar la opción **Línea de comandos** del menú **Herramientas** de Visual Studio y, a continuación, seleccionar la opción **PowerShell para desarrolladores**. Se abrirá una ventana de **PowerShell** de Windows en la ubicación de la aplicación Web API. También podría haberse obtenido el mismo resultado, abriendo una ventana de **PowerShell** desde **Windows** y desplazándose a la misma ubicación.
 - b. Para instalar la herramienta http-repl, ejecutar el siguiente comando:

```
dotnet tool install -g Microsoft.dotnet-httprepl
```



```
Developer PowerShell Visual Studio Enterprise 2019 16.11.31829.152
*****
** Visual Studio 2019 Developer PowerShell v16.11.6
** Copyright (c) 2021 Microsoft Corporation
*****
PS C:\Users\vicen\Desktop\ApiContactos> dir

Directorio: C:\Users\vicen\Desktop\ApiContactos

Mode                LastWriteTime         Length Name
----                -
d-----         11/12/2021   17:55             ApiContactos
-a-----         03/12/2021   20:29          1142 ApiContactos.sln

PS C:\Users\vicen\Desktop\ApiContactos> dotnet tool install -g Microsoft.dotnet-httprepl
Puede invocar la herramienta con el comando siguiente: httprepl
La herramienta "microsoft.dotnet-httprepl" (versión '6.0.0') se instaló correctamente.
PS C:\Users\vicen\Desktop\ApiContactos>
```

Una vez instalada la herramienta httprepl que va a facilitar la realización de pruebas y la visualización de resultados de las solicitudes HTTP que se realicen a la aplicación Web API *ApiContactos*, se puede continuar con la realización de las pruebas de ejecución.

2. Iniciar la depuración de la aplicación Web API. Podrá observarse que se abre el navegador, sin embargo, no presenta ninguna información porque la tabla *Contactos* aún está vacía.
3. En la ventana de **PowerShell**, en primer lugar, ejecutar el siguiente comando para iniciar una sesión httprepl y establecer la comunicación con el controlador *Contactos* de la Web API *ApiContactos*. Puede ser necesario tener modificar el número de puerto del comando, si es que la ejecución de la aplicación Web API, que está depurándose en el navegador, está utilizando un número de puerto diferente.

```
httprepl https://localhost:44380/api/Contactos
```

Se puede comprobar que se habrá establecido la comunicación con la aplicación Web API a través de la herramienta `htprepl`, porque habrá cambiado el símbolo de *prompt* que se muestra en la ventana de **PowerShell** y será similar al siguiente:

```
https://localhost:44380/api/Contactos/>
```

- Una vez iniciada la sesión `htprepl` y establecida la comunicación con la Web API, ejecutar el siguiente comando para añadir un nuevo contacto.

```
post -h Content-Type=application/json -c '{"Nombre":"Juan García",  
"Telefono": "666112233", "Email":"juan@emp.com", "Activado":true}'
```

Una vez ejecutado el comando se obtendrá un resultado similar al siguiente en la ventana del terminal, lo que significará que el resultado obtenido es el correcto.

```
https://localhost:44380/api/Contactos/> post -h Content-Type=application/json -c '{"Nombre":  
"Juan García", "Telefono": "666112233", "Email":"juan@emp.com", "isComplete":true}'  
HTTP/1.1 201 Created  
Content-Length: 96  
Content-Type: application/json; charset=utf-8  
Date: Sat, 11 Dec 2021 19:47:42 GMT  
Location: https://localhost:44380/api/Contactos/1  
Server: Microsoft-IIS/10.0  
X-Powered-By: ASP.NET  
  
{  
  "id": 1,  
  "nombre": "Juan García",  
  "telefono": "666112233",  
  "email": "juan@emp.com",  
  "isComplete": true  
}  
  
https://localhost:44380/api/Contactos/>
```

- A continuación, añadir un segundo contacto, ejecutando el siguiente comando.

```
post -h Content-Type=application/json -c '{"Nombre":"Vicente Fernández",  
"Telefono": "666887722", "Email":"vicente@emp.com", "Activado":true}'
```

- Ejecutar el siguiente comando para obtener la lista de contactos.

```
get
```

Y el siguiente comando para obtener la información del contacto cuyo *id* es igual a 1.

```
get 1
```

- Para finalizar la depuración de la aplicación Web API, realizar las siguientes acciones:
 - En la ventana de **PowerShell**, ejecutar el siguiente comando para finalizar la sesión `htprepl` y, por tanto, la comunicación con la Web API *ApiContactos*.

```
exit
```

- b. En la ventana de **PowerShell**, ejecutar el siguiente comando para finalizar la sesión de línea de comandos de **PowerShell** de **Windows**.

```
exit
```

- c. Cerrar la instancia de ejecución del navegador que está depurando la aplicación Web API. Si fuera necesario, finalizar la depuración desde Visual Studio.
8. A continuación, se va a probar la operación de información PUT para modificar la información almacenada de un contacto existente. En primer lugar, se va a crear un nuevo contacto sobre el cual, posteriormente, se modificará el valor de alguna de sus propiedades. Para ello, hacer:
- Iniciar la depuración de la aplicación Web API.
 - Abrir una ventana de **PowerShell para desarrolladores** desde Visual Studio.
 - Iniciar la sesión httprepl y establecer la comunicación con la aplicación Web API.

```
httprepl https://localhost:44380/api/Contactos
```

- d. A continuación, ejecutar el siguiente comando para añadir un nuevo contacto:

```
post -h Content-Type=application/json -c '{"Nombre":"Julián García", "Telefono":  
"666334433", "Email":"julian@emp.com", "Activado":true}'
```

- e. Puede obtenerse la lista de los contactos almacenados en la base de datos, realizando para ello una operación GET, tal como se ha realizado anteriormente.

```
get
```

- f. Una vez que se haya comprobado que se ha realizado correctamente la operación de añadir un nuevo contacto, finalizar la conexión con la aplicación Web API.

```
exit
```

9. A continuación, se va a modificar el nombre del último contacto añadido, cuyo valor de la propiedad *id* es 3. Para ello, se ejecutará una operación PUT de la siguiente forma:
- Iniciar sesión y establecer la comunicación con el contacto cuyo *id*=3. Lógicamente, se debe disponer del elemento a modificar en memoria para poder modificar sus datos.

```
httprepl https://localhost:44380/api/Contactos/3
```

- b. Asegurarse que el contacto cuyo *id*=3 está en memoria, ejecutando el comando.

```
get
```

- c. Ejecutar el siguiente comando correspondiente a la operación de información PUT, que llama a la acción *PutContacto()*, para modificar el nombre del contacto cuyo *id*=3.

```
put -h Content-Type=application/json -c '{"id":3, "Nombre":"Julián",  
"Telefono":"666334433", "Email":"julian@emp.com", "Activado":true}'
```

- d. Una vez que se haya añadido el nuevo contacto, finalizar la sesión httprepl.

```
exit
```

10. Obtener la lista de los contactos almacenados en la base de datos realizado una operación GET, tal como se ha realizado anteriormente. Será necesario iniciar la sesión httprepl y establecer la comunicación con la aplicación Web API para obtener la lista de contactos.
11. Finalmente, se va a eliminar el contacto cuyo valor de la propiedad *id* es 3. Para ello, hacer:
- Iniciar sesión httprepl y establecer la comunicación con el elemento cuyo id=3. Se debe disponer del elemento a eliminar en memoria para poder eliminarlo posteriormente.

```
httprepl https://localhost:44380/api/Contactos/3
```

Si no se finalizó la sesión httprepl anterior, reemplazar el comando httprepl anterior por el comando connect para establecer la comunicación con el contacto cuyo id=3.

- b. Asegurarse que el contacto cuyo id=3 está en memoria, ejecutando el comando.

```
get
```

- c. Ejecutar el siguiente comando correspondiente a la operación de información DELETE para eliminar el contacto cuyo id=3.

```
delete
```

- d. Una vez ejecutada la operación de información DELETE, finalizar la conexión con la aplicación Web API.

```
exit
```

12. Comprobar que se ha realizado la eliminación del contacto cuyo id=3, realizado una operación GET para obtener la lista de los contactos almacenados en la base de datos.
13. Finalizar la sesión httprepl, finalizar la sesión de línea de comandos de **PowerShell** de **Windows** y finalizar la depuración de la aplicación Web API.

En este ejercicio, se ha utilizado la herramienta httprepl para probar el funcionamiento de la aplicación Web API *ApiContactos*. Utilizar esta herramienta para realizar las pruebas de funcionamiento, u otras herramientas similares como Postman, constituye una práctica habitual en el ámbito del desarrollo Web en el entorno del servidor. En un ámbito de desarrollo más amplio, es habitual utilizar páginas de HTML que utilizan formularios y controladores de evento creados con *Javascript*, para generar las solicitudes HTTP que llaman a las acciones de una aplicación Web API. De esta manera, se pueden ejecutar las operaciones de información API REST necesarias para realizar los procesamientos de una aplicación Web cliente. Esta cuestión queda fuera del alcance del estudio del desarrollo Web en el entorno del servidor, por lo que no se aborda en este ámbito. Puede obtenerse más información consultando el tutorial “Llamada a una Web API de ASP.NET Core con Javascript”, disponible en:

<https://docs.microsoft.com/es-es/aspnet/core/tutorials/web-api-javascript?view=aspnetcore-5.0>