



Tecnicatura Universitaria  
en Programación

## PROGRAMACIÓN II

Unidad Temática N°4:  
Introducción a Servicios

Material Teórico  
1° Año – 2° Cuatrimestre



INTRODUCCIÓN A SERVICIOS	2
Introducción	2
CONCEPTOS INICIALES	3
API (Application Programming Interface)	3
Servicio Web	4
Características de los Servicios Web	4
Arquitecturas orientadas a servicios	5
SOAP	5
REST	6
Tipos de APIs	8
CONSTRUIR UNA WEB API CON C#	9
Desarrollar una API Rest	9
.Net Core Web API	10
Web API paso a paso	11
¿Qué es POSTMAN?	18
Probar la Web API desde POSTMAN	19
BIBLIOGRAFÍA	22

## INTRODUCCIÓN A SERVICIOS

### Introducción

El diseño del software tiende a ser cada vez más modular. Las aplicaciones se componen de una serie de componentes (servicios) reutilizables, que pueden encontrarse distribuidos a lo largo de una serie de máquinas conectadas en red.

Los Servicios Web nos permitirán distribuir nuestra aplicación a través de Internet, pudiendo una aplicación utilizar los servicios ofrecidos por cualquier servidor conectado a Internet. Es decir, estos servicios juegan un papel fundamental para exponer las APIs (Application Programming Interface) en la WEB y permitir que las aplicaciones cliente obtengan y realicen operaciones con los datos allí expuestos.

El objetivo de esta unidad es analizar los conceptos y tecnologías necesarias para poder construir una API Web que permita exponer una API de negocio sobre utilizando C# como lenguaje de soporte.

## CONCEPTOS INICIALES

### API (Application Programming Interface)

imagínese que se necesita refactorizar el presupuestador de carpintería metálica pero ahora con posibilidad de que cualquier cliente vía WEB pueda dejar su solicitud de aberturas. Evidentemente se debe pensar en una aplicación cliente-servidor que permita generar un presupuesto por Internet. Suponga que además el dueño solicita la posibilidad de registrar un cobro (en concepto de adelanto) para que el cliente pueda confirmar la ejecución de su presupuesto. ¿Desarrollaría una tecnología para hacer cobros desde cero? La respuesta es simplemente no, directamente se desarrolla un componente que se conecte a una pasarela de pago como Paypal u otras ofrecidas en el mercado y se aprovecha el desarrollo hecho por otras empresas en la aplicación. Ese es el espíritu real de las APIs.

Desde el punto de vista conceptual una API es una **interfaz**. Las interfaces constituyen una capa de abstracción para que dos programas se comuniquen entre si (compartiendo datos) sin necesidad de que ninguno conozca los detalles de implementación del otro. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A veces, las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Debido a que simplifican la forma en que los desarrolladores integran los elementos de las aplicaciones nuevas en una arquitectura actual, las API permiten la colaboración entre el equipo comercial y el de TI. Las necesidades comerciales suelen cambiar rápidamente en respuesta a los mercados digitales en constante cambio, donde la competencia puede modificar un sector entero con una aplicación nueva. Para seguir siendo competitivos, es importante admitir la implementación y el desarrollo rápidos de servicios innovadores.

Graficamente:

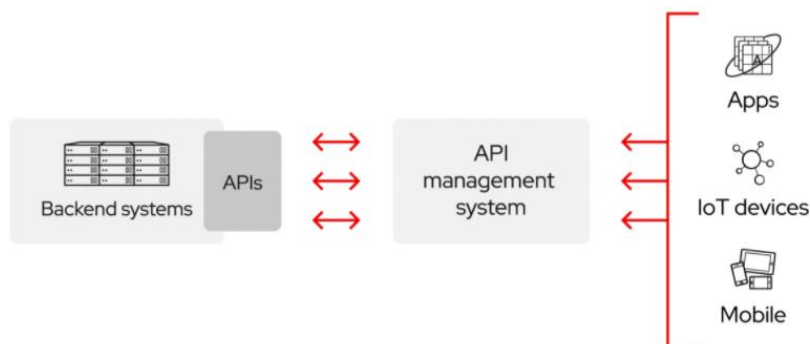


Imagen 1: Recuperado de RedHat

### Servicio Web

Normalmente un Servicio Web es una colección de procedimientos (métodos) a los que podemos llamar desde cualquier lugar de Internet o desde una intranet, siendo este mecanismo de invocación totalmente independiente de la plataforma que utilicemos y del lenguaje de programación en el que se haya implementado internamente el servicio.

Cuando se conecta a un servidor web desde nuestro navegador, el servidor devuelve la página web solicitada, que es un documento que se mostrará en el navegador para que lo visualice el usuario, pero es difícilmente entendible por una máquina. Se puede ver esto como web para humanos. En contraposición, los Servicios Web ofrecen información con un formato estándar que puede ser entendido fácilmente por una aplicación. En este caso se estaría ante una web para máquinas.

### Características de los Servicios Web

Los servicios Web son un mecanismo estándar que permite comunicar aplicaciones residentes en equipos conectados a una red. Las características deseables de un servicio Web son:

- Debe poder ser **accesible a través de la Web**. Para ello debe utilizar protocolos de transporte estándares como HTTP, y codificar los mensajes en un lenguaje estándar que pueda conocer cualquier cliente que quiera utilizar el servicio.

- Debe contener una **descripción de sí mismo**. De esta forma, una aplicación podrá saber cuál es la función de un determinado Servicio Web, y cuál es su interfaz, de manera que pueda ser utilizado de forma automática por cualquier aplicación, sin la intervención del usuario.
- Debe poder **ser localizado**. Se debe tener algún mecanismo que permita encontrar un servicio Web que realice una determinada función. De esta forma se tiene la posibilidad de que una aplicación localice el servicio que necesite de forma automática, sin tener que conocerlo previamente el usuario.

### Arquitecturas orientadas a servicios

Una arquitectura de software define la forma en la que está diseñado un sistema, cómo se organizan sus componentes, cómo se comunican entre ellos y qué funciones cumplen. En el ámbito de los servicios Web la arquitectura de referencia es SOA (Service-Oriented Architecture).

SOA es una arquitectura que se basa en la integración de aplicaciones mediante servicios, los servicios representan la medida más granular de la arquitectura, sobre la que se construyen otros artefactos. Existen dos enfoques muy comunes que siguen la misma arquitectura orientada a servicios: REST y SOAP.

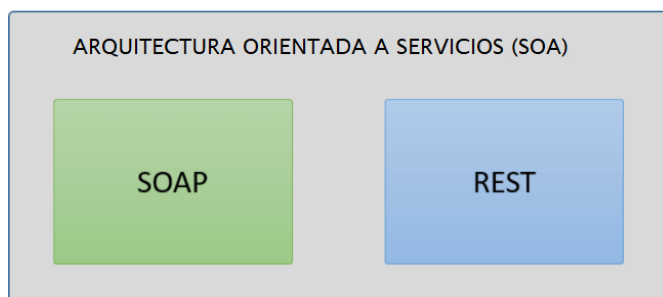


Imagen 2: Elaboración propia

### SOAP

El protocolo SOAP (Simple Object Access Protocol) es: “un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML”. Los servicios SOAP funcionan por lo general por el protocolo HTTP que es lo más común cuando invocamos un servicio Web, sin

embargo, SOAP no está limitado a este protocolo, si no que puede ser enviado por FTP, POP3, TCP, Colas de mensajería (JMS, MQ, etc).

Esquemáticamente:

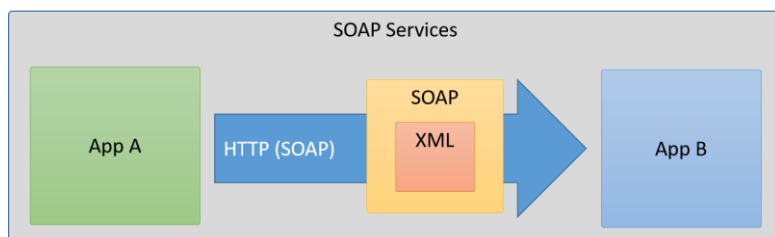


Imagen 3: Elaboración propia

La característica principal de SOAP es que es en sí mismo un protocolo para crear servicios Web. SOAP usa XML (Extensible Markup Language) como lenguaje de intercambio de datos con una estructura compleja que es capaz de albergar todo tipo de datos sobre la solicitud o respuesta generada.

Como es un protocolo, impone reglas integradas que aumentan la complejidad y la sobrecarga, lo cual puede retrasar el tiempo que tardan las páginas en cargarse. Sin embargo, estos estándares también ofrecen normas integradas que pueden ser ideales para el sector empresarial. Con SOAP, es más fácil que las aplicaciones que funcionan en entornos distintos o están escritas en diferentes lenguajes compartan información.

## REST

Otra especificación es la Transferencia de Estado Representacional (**REST**).

REST es un conjunto de principios arquitectónicos que se ajusta a las necesidades de los servicios web y las aplicaciones móviles ligeros. Dado que se trata de un conjunto de pautas, la implementación de las recomendaciones depende de los desarrolladores, no es un protocolo como SOAP.

Cuando se envía una solicitud de datos a una API de REST, se suele hacer a través del protocolo HTTP. Una vez que reciben la solicitud, las API diseñadas para REST (conocidas como API o servicios web de RESTful) pueden devolver mensajes en distintos formatos: HTML, XML, texto sin formato y JSON. El formato preferido para los mensajes es la notación de objetos JavaScript (**JSON**), ya que, a pesar de su nombre, puede leerlo cualquier lenguaje de programación, es ligero y lo

comprenden tanto las personas como las máquinas. De esta forma, las API de RESTful son más flexibles y se pueden configurar con mayor facilidad.

Se considera que una aplicación es RESTful si cumple con seis pautas arquitectónicas. Una aplicación de RESTful debe tener lo siguiente:

- Una arquitectura cliente-servidor compuesta por clientes, servidores y recursos.
- Una comunicación cliente-servidor sin estado, lo cual significa que el contenido de los clientes no se almacena en el servidor entre las solicitudes, sino que la información sobre el estado de la sesión queda en el cliente.
- Datos que pueden almacenarse en caché para eliminar la necesidad de algunas interacciones cliente-servidor.
- Una interfaz uniforme entre elementos para que la información se transfiera de forma estandarizada, en lugar de ser específica para las necesidades de cierta aplicación. Roy Fielding, el creador de REST, lo describe como "la característica principal que distingue el estilo arquitectónico de REST de los demás estilos basados en la red".
- Una restricción del sistema en capas, en el que las interacciones cliente-servidor pueden estar mediadas por capas jerárquicas.
- Código según se solicite, lo que permite que los servidores amplíen las funciones de un cliente al transferir el código ejecutable (esto también reduce la visibilidad, así que es una pauta opcional).

La diferencia entre REST y SOAP es básica: SOAP es un protocolo, mientras que REST es un estilo de arquitectura. Esto significa que no hay ningún estándar oficial para las API web de RESTful. Si bien las limitaciones REST pueden parecer demasiadas, son mucho más sencillas que un protocolo definido previamente. Por eso, las API de RESTful son cada vez más frecuentes que las de SOAP.

En los últimos años, la especificación de OpenAPI se ha convertido en un estándar común para definir las API de REST. OpenAPI establece una forma independiente del lenguaje para que los desarrolladores diseñen interfaces API de REST, que permite a los usuarios entenderlas con el mínimo esfuerzo.

### Tipos de APIs

Las API pueden ser locales o remotas. Las locales son aquellas se ejecutan dentro del mismo entorno. Por ejemplo, cuando utilizamos una librería .dll para utilizar

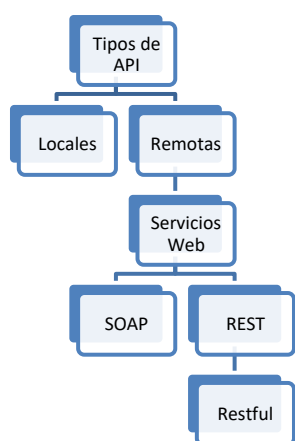


un servicio de Windows e integrarlo con nuestra aplicación estamos haciendo uso de una API local.

Las APIs remotas están diseñadas para interactuar en una red de comunicaciones. Por "remoto" se refiere a que los recursos que administran las API están, de alguna manera, fuera de la computadora que solicita alguno de dichos recursos. Debido a que la red de comunicaciones más usada es Internet, la mayoría de las API están diseñadas de acuerdo con los estándares web. No todas las API remotas son API web, pero se puede suponer que las API web son remotas.

Las API web normalmente usan HTTP para solicitar mensajes y proporcionar una definición de la estructura de los mensajes de respuesta. Por lo general, estos mensajes de respuesta toman la forma de un archivo XML o JSON, que son los formatos preferidos porque presentan los datos en una manera fácil de manejar para otras aplicaciones.

Esquemáticamente:



**Imagen 4:** Elaboración propia

Según el gráfico anterior podemos concluir entonces, que las API Web son APIs expuestas mediante servicios WEB que pueden seguir ciertos principios arquitectónicos como las API REST o bien respetar un protocolo específico como es

el caso de los servicios SOAP. A continuación, nos centraremos en construir una Web API con C# siguiendo los principios y recomendaciones del enfoque REST.

## CONSTRUIR UNA WEB API CON C#

### Desarrollar una API Rest

Cuando se desarrolla una API Rest es necesario comprender ciertos conceptos:

- **Recurso:** Un recurso hace referencia a un concepto importante de nuestro negocio (Facturas, Cursos, Compras etc.). Es lo que habitualmente se denomina un objeto de negocio. Este estilo permite un primer nivel de organización permitiendo acceder a cada uno de los recursos de forma independiente, favoreciendo la reutilización, aumentando la flexibilidad y abordando operaciones de inserción, borrado, búsqueda etc. Cada recurso tiene un identificador único llamado **URI (Identificador de Recurso Uniforme)**. Cuando se solicita un recurso a una Web API siempre utilizamos un tipo especial de URI que es la URL (Localizador de Recurso Uniforme). Esta URL contiene la ubicación completa del recurso y generalmente se la llama **endpoint**. Ejemplos de endpoints son:
  - un\_dominio/api/products
  - un\_dominio/api/products/1
- **Códigos de estado:** cuando solicitamos un recurso el servidor puede contestar con diferentes códigos de estado: 2xx, 3xx, 4xx o 5xx. Con estos códigos podremos saber qué pasó con la petición de nuestro recurso y qué hacer en consecuencia.
- **Verbos:** o métodos HTTP. Nos permite definir cómo interactuar con la API. Dependiendo de cada tipo de operación se utilizará un método diferente de envío.
  - *GET*: Se usará para solicitar consultar a los recursos
  - *POST*: Se usará para insertar nuevos recursos
  - *PUT*: Se usará para actualizar recursos
  - *DELETE*: Se usará para borrar recursos



Imagen 5: Elaboración propia

- **Formatos de respuestas:** las APIs pueden devolver información en diferentes formatos. El formato más común es JSON (Representación de Objetos Javascript). Este no es el único formato válido, es posible devolver archivos XML o incluso texto plano. Lo importante es que sin necesidad de cambiar la API Rest es posible devolver información en diferentes formatos.

### .Net Core Web API

Hasta el momento hemos venido trabajando con .NET Framework en su versión 4.x, en donde podemos decir que es un excelente framework, robusto, muy probado y eficiente. Pero ya que estamos por empezar un nuevo tipo de programación (Web), tenemos la posibilidad de empezar a trabajar con frameworks más modernos y eficientes, por lo tanto, el estándar que vamos a utilizar en el dictado de la materia será el de .Net Core, la evolución de .Net Framework

.NET Core es una implementación del estándar .NET que, al igual que otras implementaciones como .NET Framework o Mono, incluye todo lo necesario para crear y ejecutar aplicaciones: como los compiladores, las bibliotecas de clases básicas o la máquina virtual o runtime que proporciona el entorno donde se ejecutan las aplicaciones.

En otras palabras, con .NET Core puedes crear una aplicación con lenguajes como C# o VB.NET, utilizar en ella clases básicas como las habituales string, int o List<T>, compilarla a bytecode CIL y ejecutarla en tu ordenador. Algunas de las características particulares de este nuevo NET Core son:

- Es nuevo, y escrito prácticamente desde cero
- Es open source
- Es multiplataforma
- Es modular (piezas NuGet)

- Las operaciones principales de .NET Core se realizan desde línea de comandos
- Alto rendimiento (performance optimizada para muchas peticiones)
- Se puede distribuirse de varias formas
- .NET Core no soporta todos los modelos de aplicación ni todos los frameworks
- Se pueden desarrollar aplicaciones .NET Core con cualquier editor o IDE

### Web API paso a paso

Para crear una Web API sobre .Net Core vamos a seguir los siguientes pasos:

- **(I) Crear un proyecto WEB:**

- (a) En el menú *Archivo*, seleccione *Nuevo > Proyecto*.
- (b) Seleccione la plantilla *ASP.NET Core Web API* y haga clic en *Siguiente*

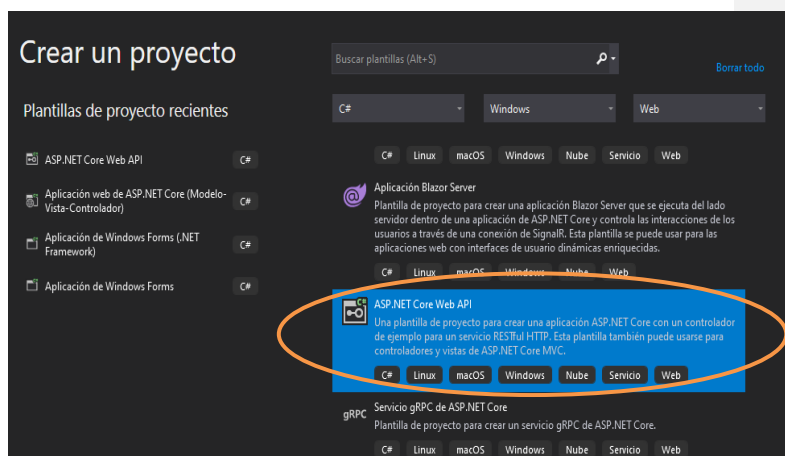
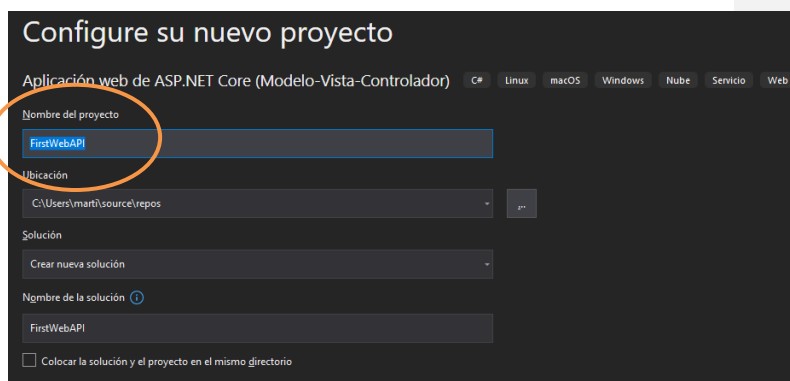


Imagen 6: Elaboración propia

Nombrar el proyecto *FirstWebAPI* y hacer click en Siguiente.



Configure su nuevo proyecto

Aplicación web de ASP.NET Core (Modelo-Vista-Controlador) C# Linux macOS Windows Nube Servicio Web

Nombre del proyecto  
FirstWebAPI

Ubicación  
C:\Users\marti\source\repos

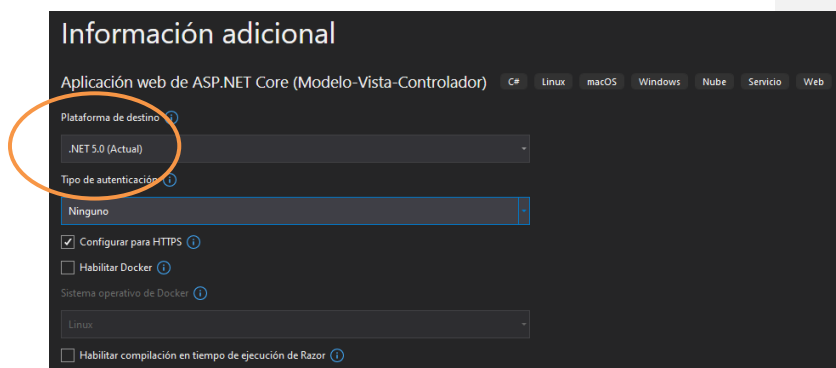
Solución  
Crear nueva solución

Nombre de la solución  
FirstWebAPI

☐ Colocar la solución y el proyecto en el mismo directorio

Imagen 7: Elaboración propia

- (c) En el cuadro de diálogo Crear una nueva aplicación web ASP.NET Core, confirme que .NET Core y ASP.NET Core 5.0 están seleccionados. Seleccione la plantilla de API y haga clic en Crear.



Información adicional

Aplicación web de ASP.NET Core (Modelo-Vista-Controlador) 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 compilación en tiempo de ejecución de Razor

Imagen 8: Elaboración propia

- (II) Probar el proyecto

- a) Presione **IIS Express** para ejecutar sin el depurador.
- b) Visual Studio muestra el siguiente cuadro de diálogo cuando un proyecto aún no está configurado para usar SSL:

Comentado [1]:

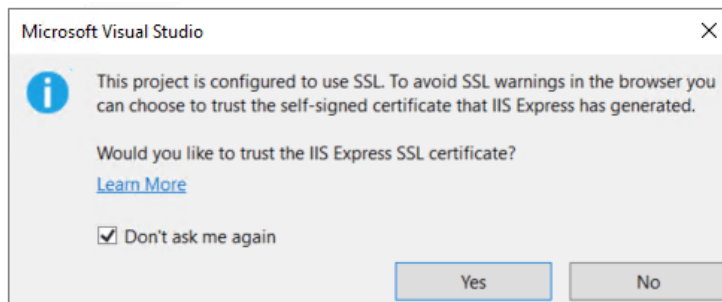


Imagen 9: Recuperado de microsoft.com

Seleccione Sí si confía en el certificado SSL de IIS Express.

Aparece el siguiente cuadro de diálogo:

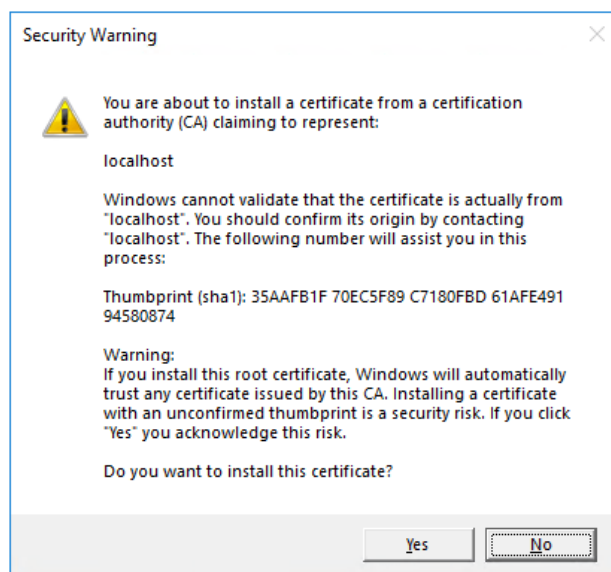


Imagen 10: Recuperado de microsoft.com

Seleccione Sí si acepta confiar en el certificado de desarrollo.

A continuación, se abrirá el Explorador que tengamos configurado por defecto con la siguiente salida:

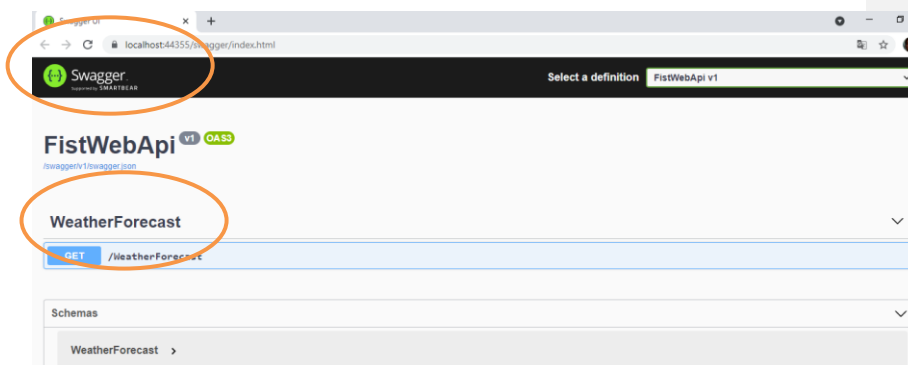


Imagen 11: Elaboración propia

Swagger (OpenAPI) es una especificación independiente del lenguaje para describir las API REST. Permite que tanto las computadoras como los humanos comprendan las capacidades de una API REST sin acceso directo al código fuente. Sus principales objetivos son:

- Minimizar la cantidad de trabajo necesario para conectar servicios desacoplados.
- Reducir la cantidad de tiempo necesario para documentar con precisión un servicio.

Cuando probamos el proyecto sobre IIS (Internet Information Server), servidor por defecto, se ejecuta de manera integrada la herramienta Swagger, que permite probar los servicios de la API, a la vez que ofrece una documentación estandarizada de los mismos. Si se observa la URL escrita en el navegador tiene la forma: `https://localhost:44355/swagger/index.html`. Desde esta página es posible acceder vía **GET** al **endpoint**: `/WeatherForecast` y consumir (hacer una petición HTTP) el método Get del controlador **WeatherForecast** (creado automáticamente por el IDE).

Para manejar las peticiones (request) las Web API utilizan **Controllers**. Los controladores son clases que derivan de **ControllerBase**. Toda web API consiste en una o más clases Controllers. La plantilla de proyecto Web API proporciona un controlador de inicio: **WeatherForecastController**.

- (III) Agregar un modelo

Un modelo es simplemente una clase que representa datos de una aplicación. Para crear un modelo primero vamos a crear una carpeta llamada *models* y luego agregar una clase llamada *Values* con las properties *Nombre* y *Valor* con se muestra en la siguiente imagen.

```
namespace FistWebApi.models
{
    Oreferencias
    public class Values
    {
        Oreferencias
        public String Nombre { get; set; }
        Oreferencias
        public object Valor { get; set; }
    }
}
```

Imagen 12: Elaboración propia

- (IV) Agregar un controlador propio

Primero vamos a eliminar desde el Explorador de soluciones el controlador creado por defecto (*WeatherForecastController*) en la carpeta *Controllers*.

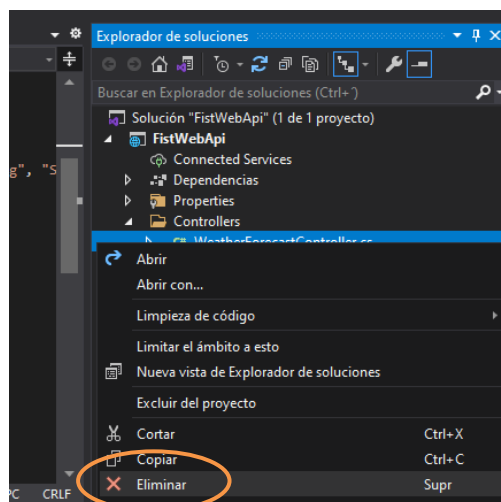


Imagen 13: Elaboración propia

Luego se crea un nuevo controlador haciendo click derecho sobre la carpeta *Controllers* y seleccionando la opción *Agregar>>Controlador...*



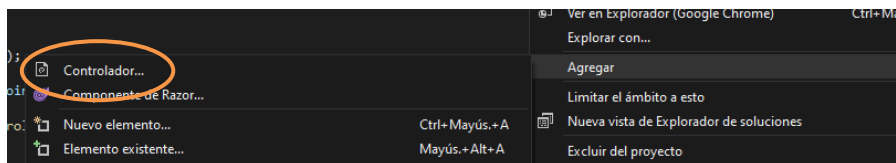


Imagen 14: Elaboración propia

En la pantalla siguiente elegir categoría **API – Controlador de API: en blanco** y seleccionar la opción **Agregar**.

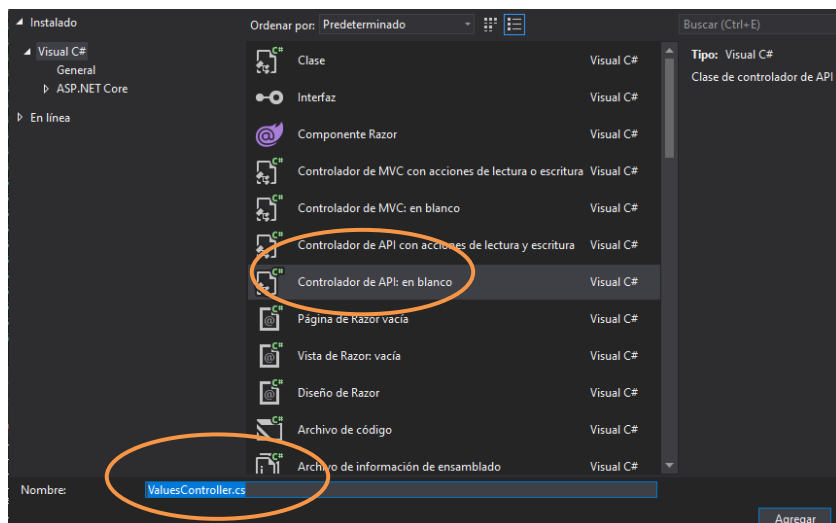


Imagen 15: Elaboración propia

Ingresar un nombre para la clase sufriendo siempre con la palabra Controller, por ejemplo: *FirstController*. Mediante este controlador vamos a crear *endpoints* uno para obtener una lista de objetos Values y otro para enviar un nuevo objeto Value desde un cliente. El código completo resulta:

```
[Route("api/[controller]")]
[ApiController]

public class FirstController : ControllerBase
{
    static readonly List<Values> lst = new List<Values>();

    [HttpGet]
    public IActionResult Get()
    {
        return Ok(lst);
    }
}
```

```
[HttpPost]
public IActionResult Save(Values dto)
{
    if (dto == null)
        return BadRequest();
    lst.Add(dto);
    return Ok("Se añadió existosamente!");
}

}
```

Notar que:

- La clase hereda de ControllerBase
- En la parte superior se indican dos *decoradores*:
  - ✓ [Route("api/[controller]")]: permite indicar la ruta con la que se accederá al controller. En este caso para consumir cualquiera de los métodos del controlador será necesario escribir la URL:  
https://localhost:44355/api/First
  - ✓ [ApiController]: este atributo permite indicar que la clase será un punto de entrada a nuestra API.
- El controlador tendrá como atributo de clase una lista de objetos Values. Al ser un atributo estático será accesible desde cualquier instancia de FirstController.
- El *primer endpoint* permite obtener la lista de objetos Values. Tiene un decorador [HttpGet] para indicar que el método por el cual se accederá a este punto es mediante **GET**. Con la sentencia:

```
return Ok(lst);
```

Se indica que se va retornar la lista de objetos como un JSON (formato por defecto de respuesta) con un código de estado 200 Ok.

- El *segundo endpoint* se utiliza para guardar un objeto Value recibido como parte de la petición. En este caso el decorador es [HttpPost] para indicar que el método por el cual se accederá es **POST**. Esto implica que la URL es la misma, pero al tener verbos HTTP diferentes el controlador puede responder con métodos distintos. Por último, dentro de este método se valida que el objeto recibido no es nulo. En caso de serlo se genera una respuesta con estado 400. Caso contrario se devuelve una cadena indicando que se agregó exitosamente.

- El tipo de retorno *ActionResult* es apropiado cuando son posibles varios tipos de retorno en una acción. Los tipos *ActionResult* representan varios códigos de estado HTTP. Cualquier clase no abstracta que se derive de *ActionResult* califica como un tipo de retorno válido.

### ¿Qué es POSTMAN?

**POSTMAN** es una herramienta que permite principalmente crear peticiones sobre APIs de una forma muy sencilla y poder, de esta manera, probar las APIs.

Para instalarte Postman es necesario que se descargue el software desde el área de descargas de <https://www.postman.com/downloads/>. Están disponibles aplicaciones para *Windows*, *Linux* y *Mac*.

El uso de Postman es gratuito, si bien nos ofrece un par de planes adicionales que serían el *Postman Pro* que nos ofrece más ancho de banda para las pruebas y *Postman Enterprise* que nos permite, entre otras cosas, poder integrar la herramienta en los sistemas de SSO de nuestra empresa.

Algunas características de POSTMAN:

- **Crear Peticiones**, te permite crear y enviar peticiones http a **servicios REST** mediante una interfaz gráfica. Estas peticiones pueden ser guardadas y reproducidas a posteriori.
- **Definir Colecciones**, mediante **Postman** podemos agrupar las APIs en colecciones. En estas colecciones podemos definir el modelo de autenticación de las APIs para que se añada en cada petición. De igual manera podemos ejecutar un conjunto de test, así como definir variables para la colección.
- **Gestionar la Documentación**, genera documentación basada en las API y colecciones que hemos creado en la herramienta. Además, **esta documentación podemos hacerla pública**.
- **Entorno Colaborativo**, permite compartir las API para un equipo entre varias personas. Para ello se apoya en una herramienta de colaborativa en Cloud.
- **Genera código de invocación**, dado un API es capaz de generar el código de invocación para diferentes lenguajes de programación: *C, cURL, C#, Go, Java, JavaScript, NodeJS, Objective-C, PHP, Python, Ruby, Shell, Swift,...*

- **Establecer variables**, con **Postman** podemos crear variables locales y globales que posteriormente utilicemos dentro de nuestras invocaciones o pruebas.
- **Soporta Ciclo Vida API management**, desde **Postman** podemos gestionar el ciclo de vida del API Management, desde la conceptualización del API, la definición del API, el desarrollo del API y la monitorización y mantenimiento del API.
- **Crear mockups**, mediante **Postman** podemos crear un servidor de mockups o sandbox para que se puedan testear nuestras API antes de que estas estén desarrolladas.

### Probar la Web API desde POSTMAN

Para probar la API Web desarrollada en el punto anterior vamos a seguir los siguientes pasos:

- 1) Acceder a la lista de objetos Values mediante GET
  - Seleccionar la opción **APIs** del menú lateral derecho
  - Ingresar la URL <https://localhost:44355/api/First> en el campo que está junto a la opción **SEND**
  - Seleccionar método **GET** de la lista desplegable
  - Seleccionar opción **SEND**
  - Gráficamente:

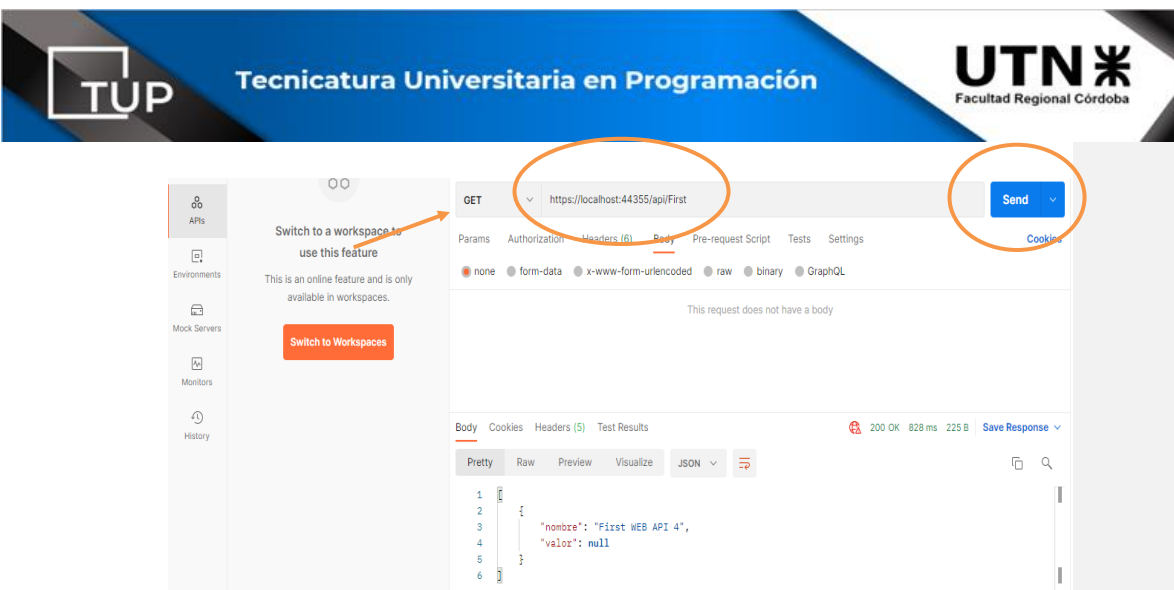


Imagen 16: Elaboración propia

## 2) Enviar un objeto Values en formato JSON mediante POST

Para enviar un objeto Values necesitamos:

- Con la misma URL cambiar el método HTTP a **POST**
- Seleccionar la pestaña **Body**
- En la parte superior seleccionar la opción **raw**. Esto habilita en un cuadro de texto el ingreso del objeto a enviar en el formato que se desee. Ingresar:

```
{
  "Codigo": 5,
  "Nombre": "Valor 5"
}
```

Las llaves indican comienzo y fin del objeto. Cada property se envía como pareja clave: valor, separada por coma.

- Seleccionar de la lista de formatos: **JSON**
- Seleccionar opción **SEND**
- Gráficamente:

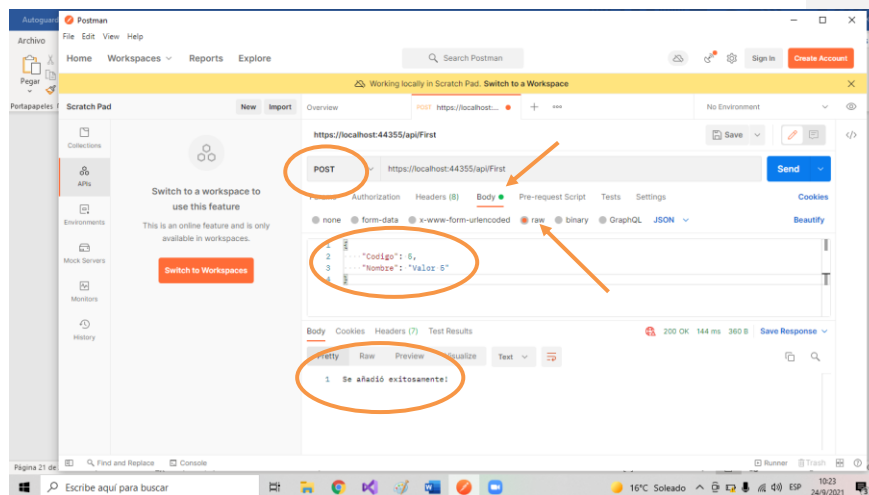


Imagen 17: Elaboración propia

Notar que en **body** de la respuesta se muestra la cadena: “Se añadió exitosamente!”

- 3) Consumir nuevamente el endpoint de la lista para validar que se agregó el objeto enviado en el punto 2).

Siguiendo los mismos pasos descriptos en 1) consultar la lista de objetos Values nuevamente y validar que la lista tiene un nuevo elemento.

## BIBLIOGRAFÍA

¿Qué es una API? Recuperado de:  
<https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>

¿Qué es REST? Recuperado de: <https://www.arquitecturajava.com/que-es-rest/>.

Diferencias entre .NET Core y .NET Framework. Recuperado de:  
<https://www.campusmvp.es/recursos/post/10-diferencias-entre-net-core-y-net-framework.aspx>.

Microsoft. Crear Web APIs con ASP.NET Core. Recuperado de:  
<https://docs.microsoft.com/>

API Rest en C#. Recuperado de: <https://www.netmentor.es/entrada/api-rest-csharp>

Postman. Getting Started.

Recuperado de: <https://learning.postman.com/docs/getting-started/introduction/>



Atribución-No Comercial-Sin Derivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterado su contenido, ni se comercialice. Referenciarlo de la siguiente manera:

Universidad Tecnológica Nacional Facultad Regional Córdoba (S/D). Material para la Tecnicatura Universitaria en Programación, modalidad virtual, Córdoba, Argentina.