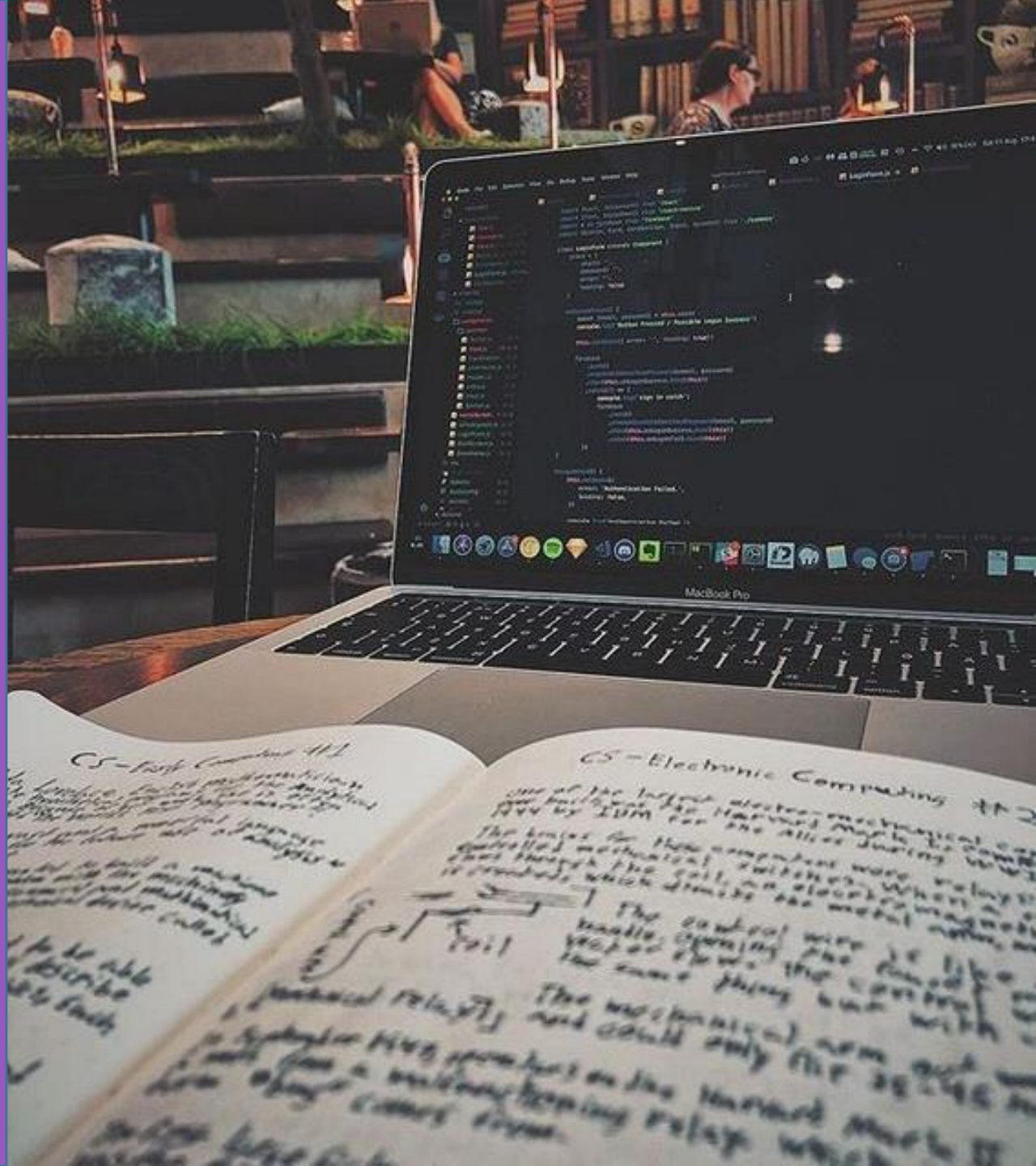


Clase Nro 9

- Qué es Json
- Un poco de Historia
- Usos más comunes para este formato
- Formato de ejemplo
- Tipos aceptados por archivos Json
- Comparación de una archivo JSON con XML
- Datos a tener en cuenta
- Json en .Net
- Serialización de datos y objetos
- Servicios Webs
- Consumir Servicios web desde .Net



Json

JSON (acrónimo de **JavaScript Object Notation**, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos.



Json

Si bien JSON nace como notación Literal de Objetos de JavaScript, sin embargo en la práctica es un formato de texto completamente independiente de lenguaje de programación.

Es, entonces un formato ligero de intercambio de datos, que resulta sencillo de leer y escribir para los programadores y simple de interpretar y generar para las máquinas.

Json – Un poco de historia

Douglas Crockford, Gurú de JavaScript es su creador.

¿Porqué JSON?

A principios de la década de los 90 surgió el problema de que las máquinas pudieran entenderse entre sí. Entonces utilizaban diferentes sistemas operativos y sus programas estaban escritos en diferentes lenguajes de programación. Una de las soluciones fue crear el estándar XML.

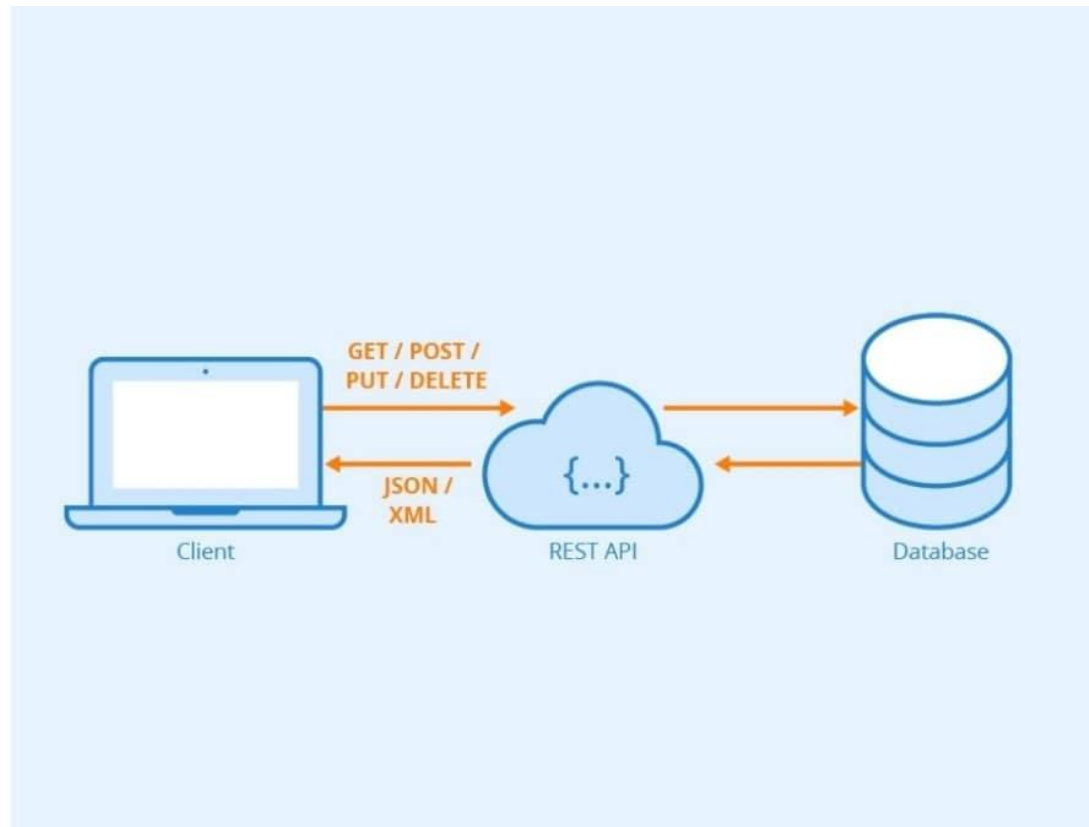
Sin embargo, XML presentaba problemas sobre todo cuando se trataba de trabajar con gran volumen de datos, puesto que el procesamiento se volvía lento

Desde entonces JSON se caracteriza por reducir el tamaño de los archivos y el volumen de datos que es necesario transmitir. Por ello fue adquiriendo popularidad hasta convertirse en un estándar. Esto no significa que XML haya dejado de utilizarse, en la actualidad ambos se emplean para el intercambio de datos.

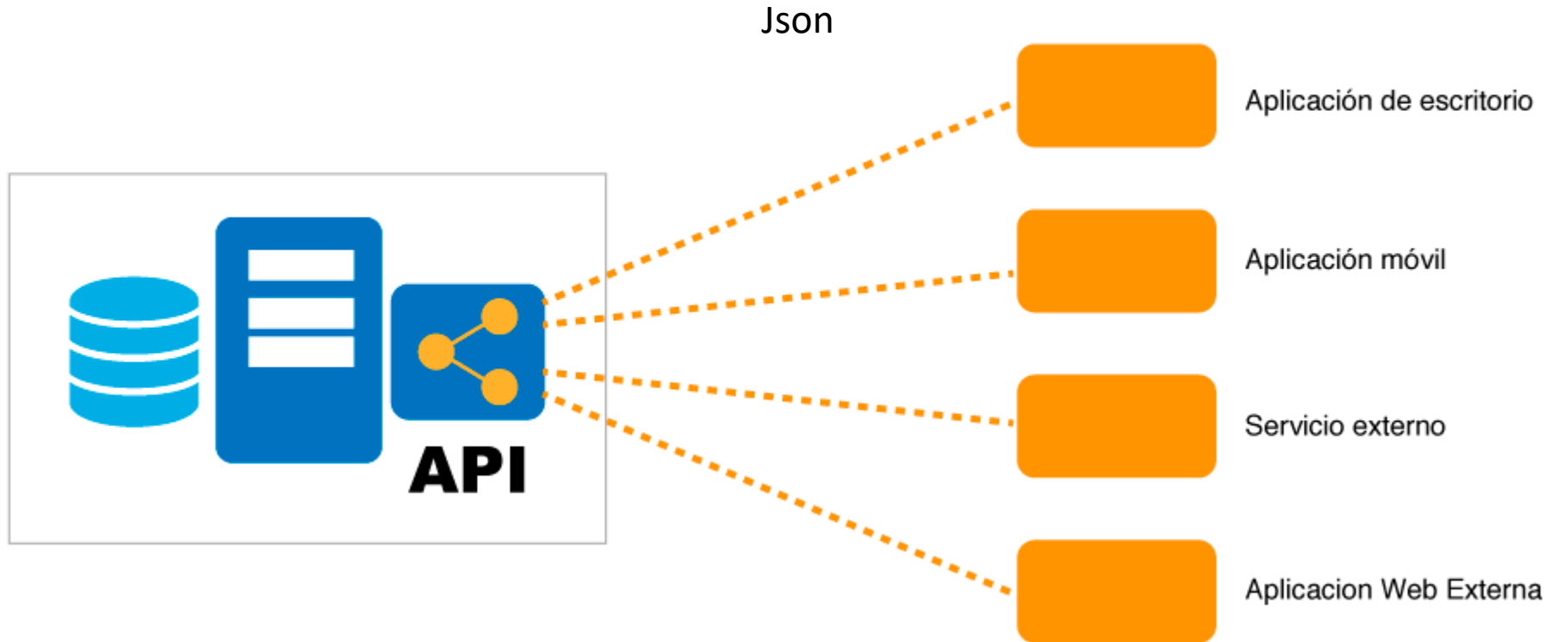


Json - Usos

Todas estas características hacen de JSON un formato de intercambio de datos ideal para usar con [API REST](#) o [AJAX](#).



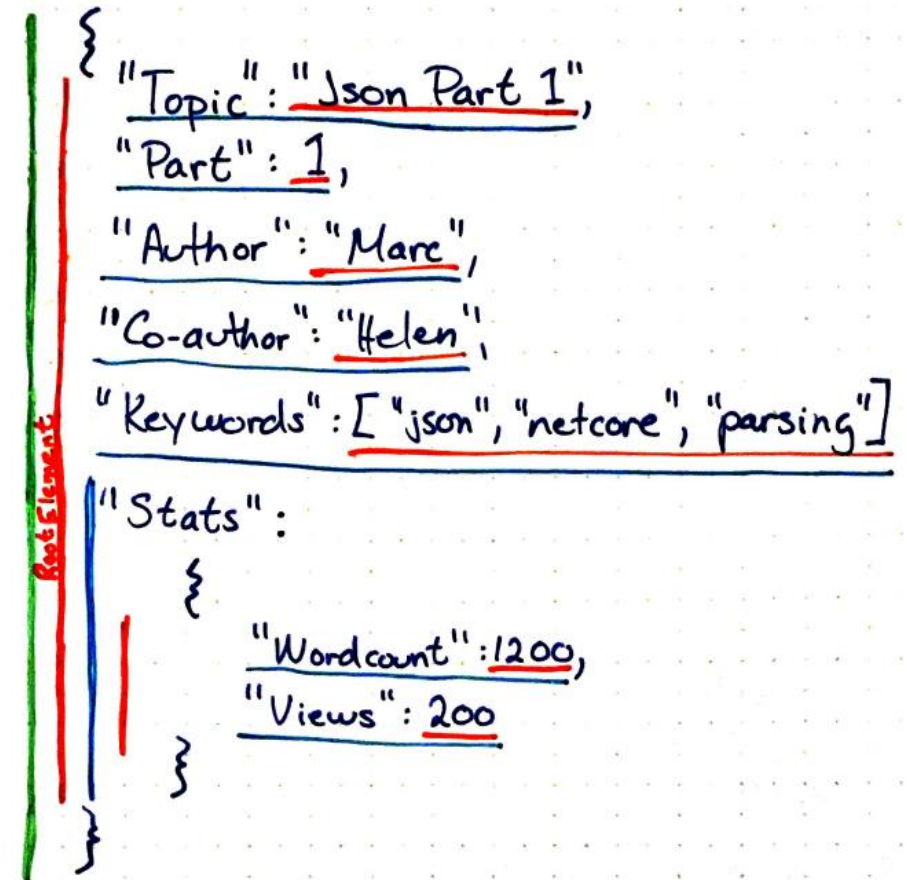
Json – Usos: Servicios web



Json

JSON

```
{
  "departamento": 8,
  "nombredepto": "Ventas",
  "director": "Juan Rodríguez",
  "empleados": [
    {
      "nombre": "Pedro",
      "apellido": "Fernández"
    }, {
      "nombre": "José",
      "apellido": "Macome"
    }
  ]
}
```



A handwritten JSON object on a grid background. The JSON is:

```
{
  "Topic": "Json Part 1",
  "Part": 1,
  "Author": "Marc",
  "Co-author": "Helen",
  "Keywords": ["json", "netcore", "parsing"],
  "Stats": {
    "Wordcount": 1200,
    "Views": 200
  }
}
```

 Annotations include: a green vertical line on the left labeled 'RootElement' in red; a blue vertical line on the left of the 'Stats' object; and red underlines under 'Json Part 1', '1', 'Marc', 'Helen', 'json', 'netcore', 'parsing', '1200', and '200'.

— JsonDocument

— JsonProperty

— JsonElement

Json - Los tipos de datos aceptados

1. **null**: Representan el valor nulo.
2. **Números**: Se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos.
3. **Cadenas**: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape. Ejemplo: "Hola"
4. **Booleanos**: Representan valores booleanos y pueden tener dos valores: true y false
5. **Array**: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo
["juan","pedro","jacinto"]
6. **Objetos**: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena entre comillas dobles. El valor puede ser de cualquier tipo

Json – Comparativa con xml

JSON

```
{
  "actor":
  [
    {
      "id": "01",
      "name": "Tom",
      "lastname": "Hanks"
    },
    {
      "id": "02",
      "name": "Nick",
      "lastname": "Thameson"
    }
  ]
}
```

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <actor>
    <id>01</id>
    <name>Tom</name>
    <lastname>Hanks</lastname>
  </actor>
  <actor>
    <id>02</id>
    <name>Nick</name>
    <lastname>Thameson</lastname>
  </actor>
</root>
```

Json – Algunos conceptos

- JSON es solo un formato de datos.
- Requiere usar comillas dobles para las cadenas y los nombres de propiedades. Las comillas simples no son válidas.
- Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione.
- Puede tomar la forma de cualquier tipo de datos que sea válido para ser incluido en un JSON, no solo arreglos u objetos. Así, por ejemplo, una cadena o un número único podrían ser objetos JSON válidos.
- A diferencia del código JavaScript, en el que las propiedades del objeto pueden no estar entre comillas, en JSON solo las cadenas entre comillas pueden ser utilizadas como propiedades.

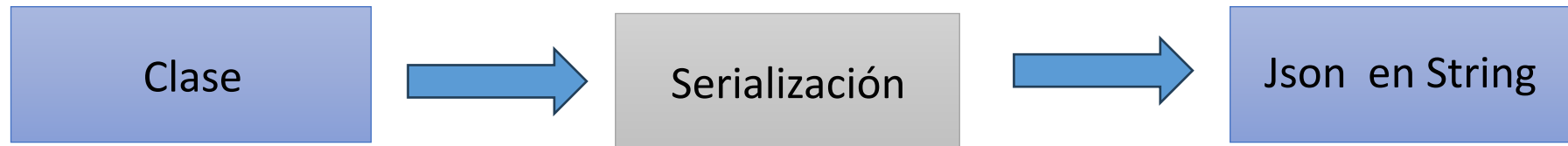
Json – Serialización y deserialización

Json – Serialización y deserialización

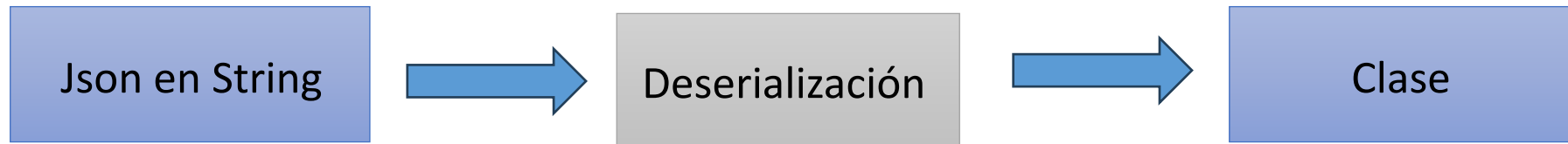
Serializar/deserialización

La serialización es el proceso de convertir un objeto en una secuencia de bytes o una cadena de texto que representa al objeto. Esto permite que el objeto sea fácilmente almacenado, transmitido a través de una red, o guardado en una base de datos.

Serializar



Deserialización



Json – Serialización y deserialización

Aplicación

En el contexto de aplicaciones y sistemas, la serialización es fundamental para:

- **Persistencia de Datos:** Almacenar el estado del objeto en archivos o bases de datos.
- **Comunicación de Red:** Enviar datos entre diferentes sistemas o componentes a través de redes.
- **Clonación de Objetos:** Crear una copia exacta del objeto.
- **Colas de Mensajes:** Enviar objetos a través de sistemas de mensajería.

Json

Yo diría que si.

¿Es JSON la forma
mas conveniente de
transmitir datos?



Json – En Net Core

Json – Serialización

Tipos de serializaciones de .Net

.NET incluye las siguientes tecnologías de serialización:

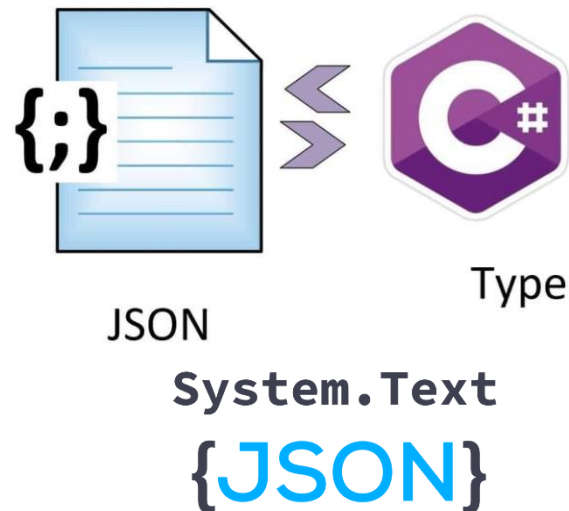
- La [serialización binaria](#) preserva la fidelidad de tipo, lo que es útil para conservar el estado de un objeto entre distintas invocaciones de una aplicación.
- La [serialización de SOAP y XML](#) solo serializa propiedades y campos públicos y no preserva la fidelidad de tipo.
- La [serialización de JSON](#) solo serializa propiedades públicas y no preserva la fidelidad de tipo.

Json – En Net Core

Serialización de Json en .Net

En .Net vamos a encontrar dos espacios de nombre que nos permitirán trabajar con Archivos Json:

- **System.Text.Json:** Incorporado en el propio sdk y no necesita descargar librerías
- **Newtonsoft.Json:** Es una librería externa y se la debe obtener de algún repo, comúnmente se utiliza nuget



<https://medium.com/@serasiyasavan14/newtonsoft-json-vs-system-text-json-a-deep-dive-into-json-serialization-for-net-developers-f4fc4d0815b9>

JSON – Serialización

Componentes principales dentro del espacio de nombres System.Text.Json :

- **JsonSerializer:** Provee la funcionalidades para serializar Objetos de .Net a JSON y también para deserializar JSON en Objetos .Net
- **JsonDocument:** Provee acceso para examinar la estructura de un documento JSON.
- **JsonElement:** Representa un valor específico JSON dentro de un documento JSON.
- **Utf8JsonWriter:** Proporciona una API de alto rendimiento para la escritura no almacenada en caché de solo reenvío de texto JSON codificado en UTF-8.
- **Utf8JsonReader:** Proporciona una API de alto rendimiento para el procesamiento de texto Json codificado en UTF8 para forward-only, token-by-token.

Json – Serialización y deserealización

Pasos usando la Serialización con System.Text.Json

Buscamos convertir un **objeto C#** a una **cadena JSON**.

1. Definir una Clase: Crear una clase que represente la estructura de datos que quieres serializar.

```
public class Producto
{
    public string Nombre { get; set; }
    public int Precio { get; set; }
}
```

2. Crear una Instancia del Objeto: Crea una instancia de la clase y asigna valores a sus propiedades.

```
Producto papas = new Producto{ Nombre = "Papas Fritas", Precio = 30 };
```

3. Serializar el ObjetoUtiliza JsonSerializer: Serialize para convertir el objeto a una cadena JSON.

```
string jsonString = JsonSerializer.Serialize(papas);
```

4. Imprimir el ResultadoMuestra la cadena JSON resultante.

```
Console.WriteLine(jsonString);
```

Json – Serialización y deserealización

```
using System;  
using System.Text.Json;
```

```
public class Producto  
{  
    public string Nombre { get; set; }  
    public int Precio { get; set; }  
}
```

```
Producto papas = new Producto { Nombre = "Papas Fritas", Precio = 30 };  
string jsonString = JsonSerializer.Serialize(papas);  
Console.WriteLine(jsonString);
```



**Clase
Producto**

Salida:

```
> { "Nombre" : "Papas Fritas" , " Precio" : 30 }
```

Json – Serialización y deserealización

Pasos usando la deserialización con System.Text.Json

Buscamos convertir una cadena JSON a un objeto C#

1. Definir la Clase: Define la misma clase que utilizaste para serializar los datos.

```
public class Producto
{
    public string Nombre { get; set; }
    public int Precio { get; set; }
}
```

2. Cadena JSON: requerimos de una cadena JSON que necesitemos deserializar. La estructura debe coincidir con la clase.

```
string jsonString = "{\"Nombre\":\"Papas Fritas\",\"Precio\":30}";
```

3. Deserializar la Cadena JSON: JsonSerializer.Deserialize para convertir la cadena JSON en una instancia de la clase.

```
Producto producto = JsonSerializer.Deserialize<Producto>(jsonString);
```

4. Mostrar los valores deserializados:

```
Console.WriteLine($"Nombre: {producto.Nombre}, Precio: {producto.Precio}");
```

Json – Serialización y deserealización

```
using System;  
using System.Text.Json;
```

```
public class Producto  
{  
    public string Nombre { get; set; }  
    public int Precio { get; set; }  
}
```

```
string jsonString = "{\\\"Nombre\\\":\\\"Papas Fritas\\\",\\\" Precio \\\":30}";  
Producto producto = JsonSerializer.Deserialize<Producto>(jsonString);  
Console.WriteLine($"Nombre: {producto.Nombre}, Precio: {producto.Edad}");
```

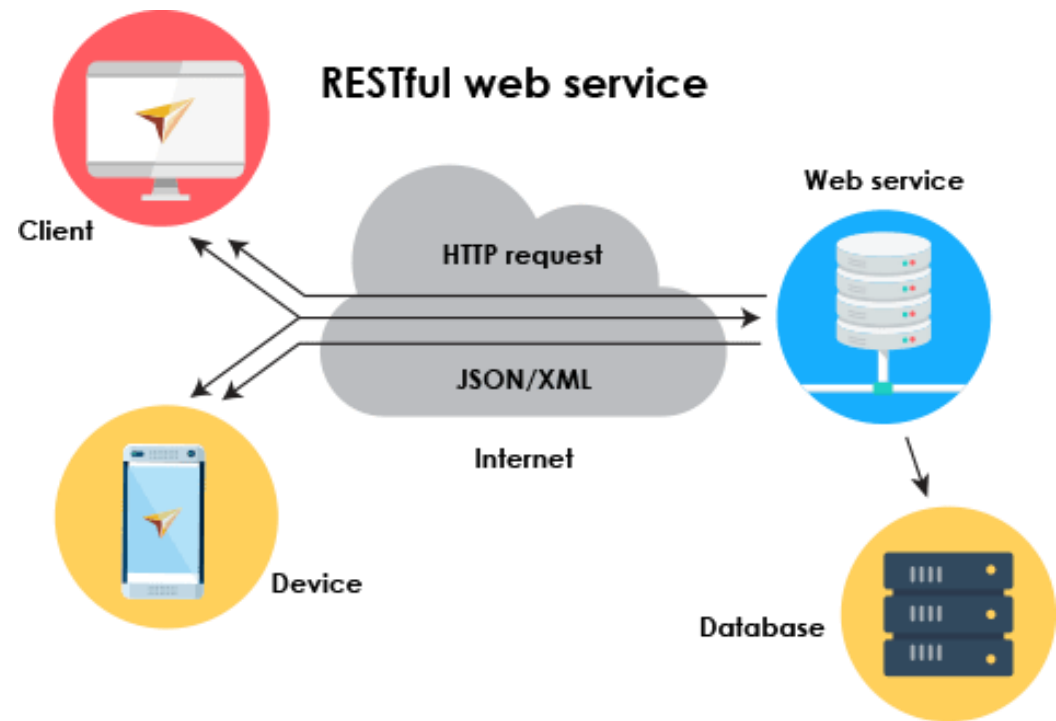


JSON

Salida:

```
> { "Nombre" : " Papas Fritas" , " Precio" : 30 }
```


Servicios web

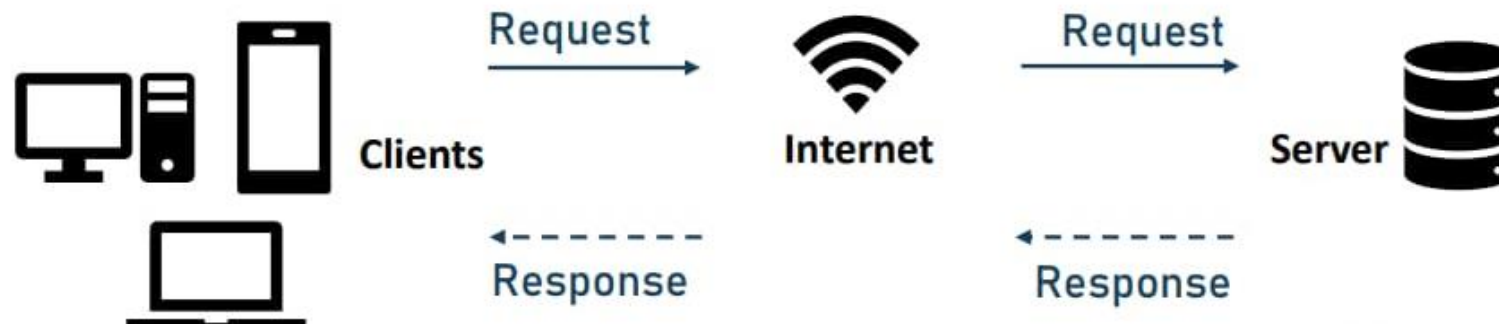


Servicios webs

Un servicio web es una aplicación que interactúa con otras aplicaciones a través de la web utilizando estándares y protocolos abiertos.

Los servicios webs permiten que aplicaciones desarrolladas en diferentes lenguajes y plataformas se comuniquen entre sí.

Alguno de los estándares Abiertos que se utilizan son **XML**, **JSON**, **WSDL** (Web Services Description Language), y **UDDI** (Universal Description, Discovery, and Integration).



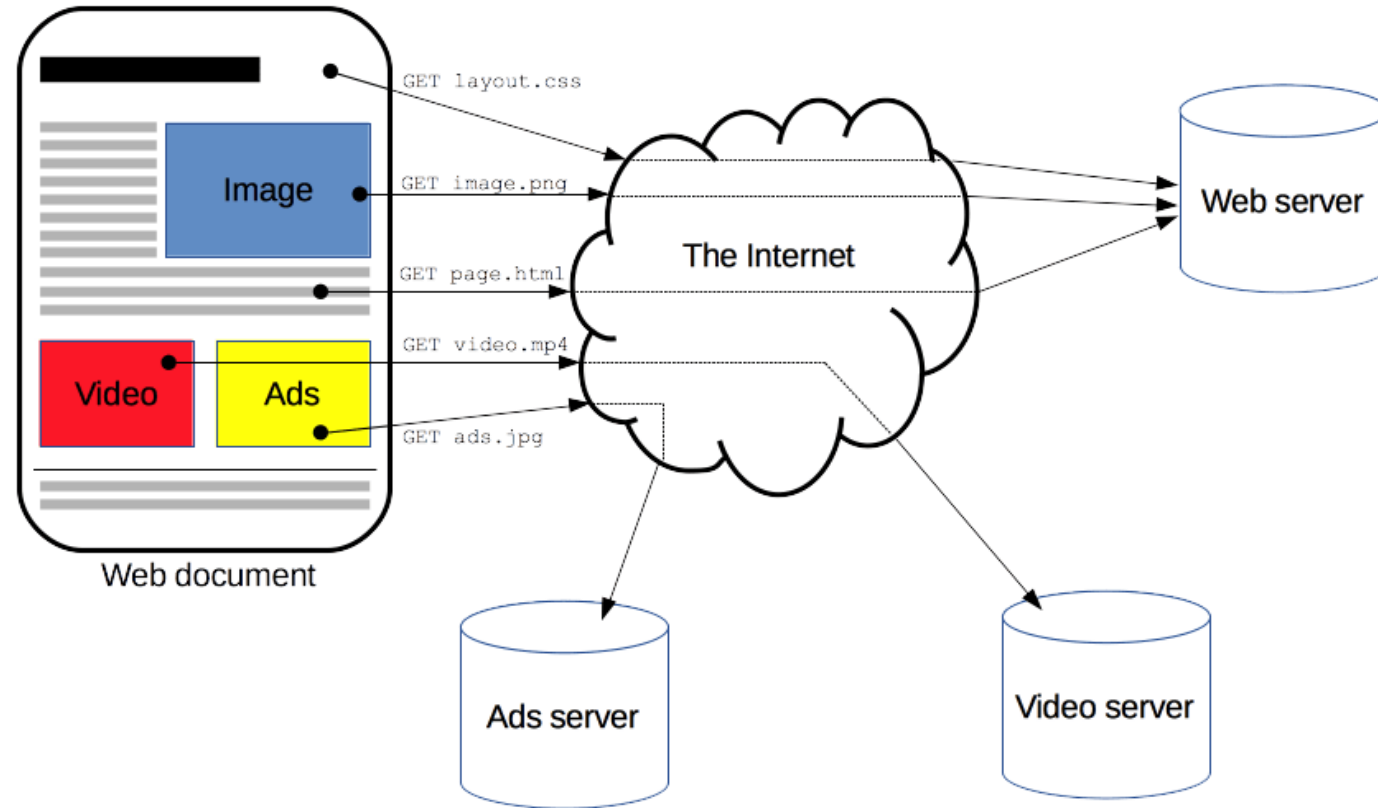
Servicios webs

Para que se utilizan

- **Integración de Sistemas:** Facilitan la integración de diferentes sistemas dentro de una organización o entre organizaciones distintas.
- **Intercambio de Datos:** Permiten el intercambio de datos y la comunicación entre aplicaciones web y móviles.
- **Automatización de Procesos:** Automatizan procesos de negocio mediante la comunicación entre distintos servicios.
- **Escalabilidad y Flexibilidad:** Ayudan a crear aplicaciones escalables y flexibles mediante el uso de microservicios y API.

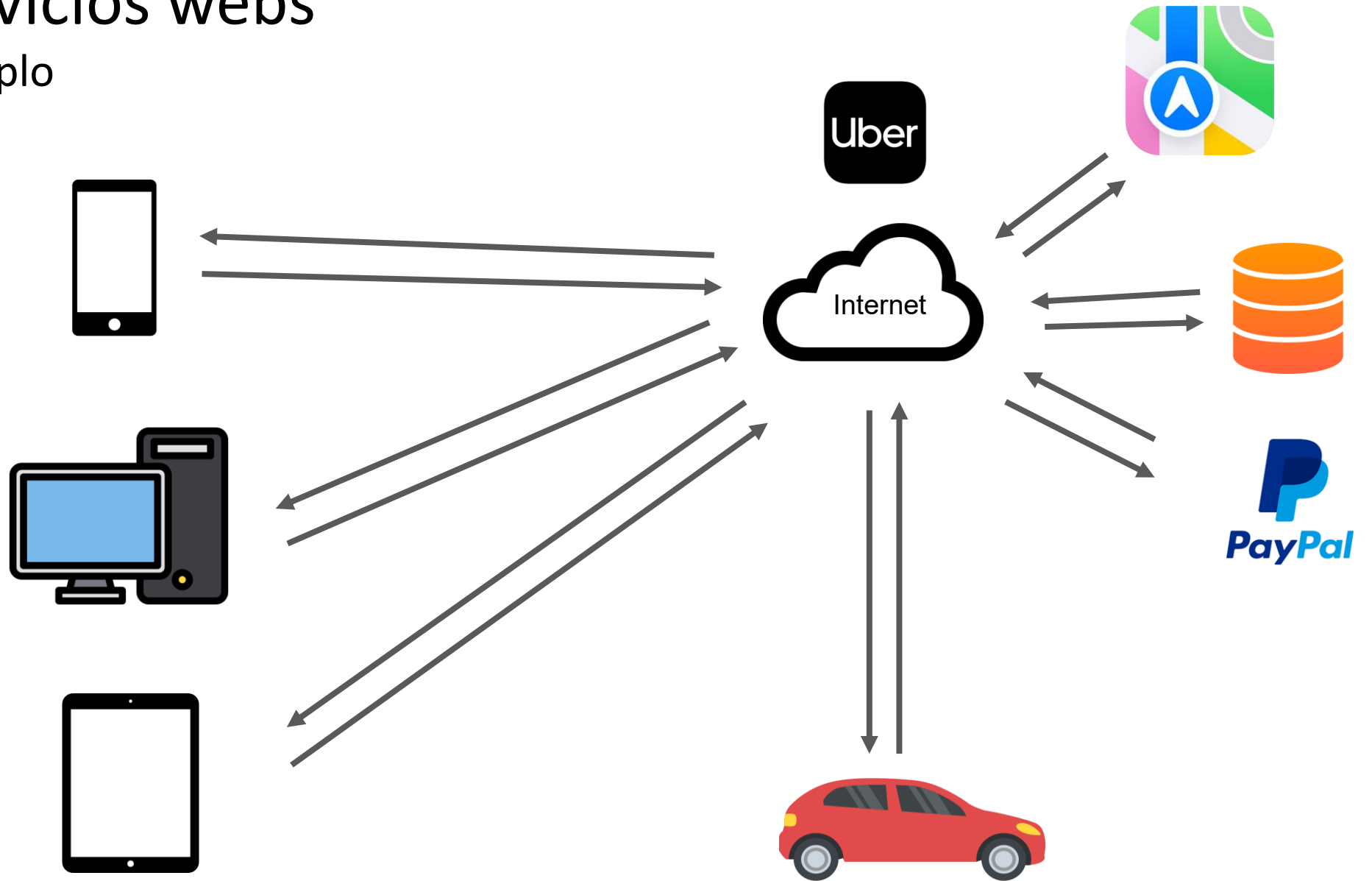
Acceso Remoto a Funcionalidades: Proveen acceso a funcionalidades y datos a través de la web desde cualquier ubicación.

Servicios webs



Servicios webs

Ejemplo

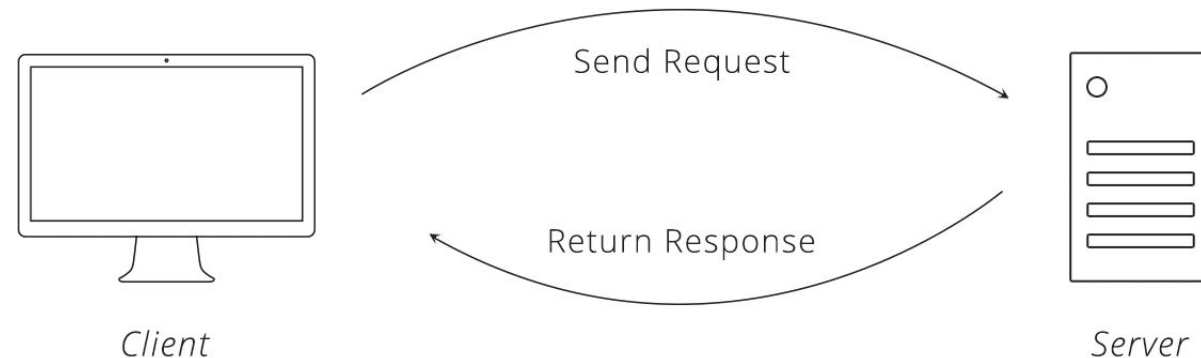


Servicios webs

Protocolo HTTP

La comunicación en HTTP se centra en un concepto llamado ciclo de solicitud-respuesta. El cliente envía al servidor una solicitud para hacer algo. El servidor, a su vez, envía al cliente una respuesta diciendo si el servidor puede o no hacer lo que el cliente pidió.

Por lo tanto, hay dos tipos de mensajes: *peticiones (request)*, enviadas por el cliente al servidor, para pedir el inicio de una acción; y *respuestas (response)*, que son la respuesta del servidor



Servicios web – con .Net

Servicios webs

HttpClient

Se utiliza para enviar solicitudes HTTP a un servidor web que pueden ser GET, POST, etc.

Características

- **Asincronía:** Soporta operaciones asincrónicas, permitiendo realizar solicitudes y procesar respuestas sin bloquear el hilo principal.
- **Configurabilidad:** Ofrece múltiples opciones de configuración, como establecer encabezados personalizados, tiempo de espera, autenticación y manejo de cookies.
- **Reutilización de Instancias:** Reutilizar una única instancia de HttpClient a lo largo de la vida de la aplicación reduce la sobrecarga de recursos y mejora el rendimiento.
- **Manejo de Redirecciones y Cookies:** Puede manejar automáticamente redirecciones HTTP y gestionar cookies, simplificando la interacción con APIs web complejas.
- **Soporte para Contenidos Diversos:** Facilita el envío y recepción de diversos tipos de contenido, como JSON, XML, archivos y formularios, útil para interactuar con distintas APIs.

Servicios webs

Usar HttpClient

1. Crear una Instancia de HttpClient:

```
HttpClient client = new HttpClient();
```

2. Enviar Solicitud GET: Se envía una solicitud GET a la URL especificada y se verifica que la respuesta sea exitosa.

```
HttpResponseMessage response = await client.GetAsync(url); // en url especificamos la dirección  
web donde consultamos el servicio  
response.EnsureSuccessStatusCode();
```

3. Leer y Deserializar la Respuesta:

```
string responseBody = await response.Content.ReadAsStringAsync();  
List<Clima> listclima = JsonSerializer.Deserialize<List<Clima>>(responseBody);
```

4. Mostrar Información:

```
foreach (var Prov in listclima)  
{  
    Console.WriteLine("Nombre: " + Prov.name + " Temperatura: " + Prov.weather.temp);  
}
```

Servicios webs

Usar HttpClient

```
private static readonly HttpClient client = new HttpClient();  
await GetClima();
```

```
private static async Task GetClima()  
{  
    var url = "https://ws.smn.gob.ar/map_items/weather/";  
    HttpResponseMessage response = await client.GetAsync(url);  
    response.EnsureSuccessStatusCode();  
    string responseBody = await response.Content.ReadAsStringAsync();  
    List<Root> listclima = JsonSerializer.Deserialize<List<Root>>(responseBody);  
    foreach (var Prov in listclima)  
    {  
        Console.WriteLine("Nombre: " + Prov.name + " Temperatura: " + Prov.weather.temp);  
    }  
}
```

Servicios webs

Usar HttpClient

```
private static readonly HttpClient client = new HttpClient();
```

```
await GetClima();
```

```
private static async Task<List<Root>> GetClima()
```

```
{
```

```
    var url = "https://ws.smn.gob.ar/map_items/weather/";
```

```
    HttpResponseMessage response = await client.GetAsync(url);
```

```
    response.EnsureSuccessStatusCode();
```

```
    string responseBody = await response.Content.ReadAsStringAsync();
```

```
    List<Root> listclima = JsonSerializer.Deserialize<List<Root>>(responseBody);
```

```
    foreach (var Prov in listclima)
```

```
    {
```

```
        Console.WriteLine("Nombre: " + Prov.name + " Temperatura: " + Prov.weather.temp);
```

```
    }
```

```
}
```

Importante:
Es necesario
ya que la
solicitud es
asincrónica