UD 2 ASP.NET Core. Características. El lenguaje C#.
2.3 Métodos.

En la tercera parte de la unidad veremos cómo crear métodos en C#.

Se recomienda utilizar como referencia el capítulo 3 del siguiente libro:

https://www.syncfusion.com/ebooks/csharp

Además del apartado correspondiente en la guía de programación de C#, parte de la documentación oficial del lenguaje:

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/methods

Métodos.

Un método es un bloque de código que contiene una serie de instrucciones. En C#, todas las instrucciones ejecutadas se realizan en el contexto de un método. El método Main es el punto de entrada para cada aplicación de C# y se llama cuando se inicia el programa.

Los métodos se definen para agrupar un fragmento de código, de modo que no necesitemos escribir el mismo código varias veces.

Firma de un método.

Los métodos se declaran el nivel de acceso, como `public` o `private`, modificadores opcionales, el valor de retorno, el nombre del método y los parámetros del método. Todas estas partes forman la firma del método.

Los nombres de un método siguen las mismas normas que los identificadores en C#, pero ojo, las normas de estilo de C#, al contrario que otros lenguajes como puede ser Java, indican que los nombres de métodos deben utilizar la notación Pascal case, por lo tanto los nombres de los métodos en C# comienzan por una letra **mayúscula**.

Los parámetros de método se encierran entre paréntesis y se separan por comas. Los paréntesis vacíos indican que el método no requiere parámetros.

Acceder a un método.

Llamar a un método en un objeto es como acceder a un campo. Después del nombre del objeto, se añade el operador punto, el nombre del método y los argumentos entre paréntesis y separados por comas.

Parámetros.

La definición del método especifica los nombres y tipos de todos los parámetros necesarios. Si el código de llamada llama al método, proporciona valores concretos denominados argumentos para cada parámetro. Los argumentos deben ser compatibles con el tipo de parámetro, pero el nombre del argumento (si existe) utilizado en el código de llamada no tiene que ser el mismo que el parámetro con nombre definido en el método.

Parámetros con nombre.

Los parámetros con nombre le liberan de la necesidad de recordar o buscar el orden de los parámetros de la lista de parámetros de los métodos llamados. El parámetro de cada argumento se puede especificar por nombre de parámetro.

Por ejemplo, se podría llamar de la manera habitual a una función que imprime los detalles de un pedido mediante el envío de argumentos por posición, en el orden definido por la función.

```
PrintOrderDetails("Gift Shop", 31, "Red Mug");
```

Si no se recuerda el orden de los parámetros pero se conoce sus nombres, se puede enviar los argumentos en cualquier orden.

```
PrintOrderDetails(orderNum: 31, productName: "Red Mug", sellerName:
"Gift Shop");

PrintOrderDetails(productName: "Red Mug", sellerName: "Gift Shop",
orderNum: 31);
```

Los argumentos con nombre también mejoran la legibilidad del código al identificar lo que cada argumento representa.

Parámetros por defecto

La definición de un método si sus parámetros son necesarios o son opcionales. Todas las llamadas deben proporcionar argumentos para todos los parámetros necesarios, pero pueden omitir los argumentos para los parámetros opcionales.

Cada parámetro opcional tiene un valor predeterminado como parte de su definición. Si no se envía ningún argumento para ese parámetro, se usa el valor predeterminado.

Los parámetros opcionales se definen al final de la lista de parámetros después de los parámetros necesarios. Si el autor de la llamada proporciona un argumento para algún parámetro de una sucesión de parámetros opcionales, debe proporcionar argumentos para todos los parámetros opcionales anteriores. No se admiten espacios separados por comas en la lista de argumentos.

Por ejemplo, en el código siguiente, el método *ExampleMethod* se define con un parámetro necesario y dos opcionales.

```
public void ExampleMethod(int required,
    string optionalstr = "default string", int optionalint = 10)
```

Devolver valores.

Los métodos pueden devolver un valor al autor de llamada. Una instrucción con la palabra clave *return* seguida de un valor que coincide con el tipo de valor devuelto devolverá este valor al autor de llamada del método.

La palabra clave *return* también detiene la ejecución del método. Si el tipo de valor devuelto es *void*, una instrucción *return* sin un valor también es útil para detener la ejecución del método. Sin la palabra clave *return*, el método dejará de ejecutarse cuando alcance el final del bloque de código. En los métodos con un tipo de valor devuelto es obligatorio utilizar *return* para devolver un valor.

Ejemplo.

Antes de continuar haciendo ejercicios, vamos a adelantar una pequeña parte del contenido del próximo tema, de modo que podamos hacer nuestros ejercicios más dinámicos.

Hasta ahora en todos nuestros ejercicios realizamos una petición única a nuestra aplicación, que muestra entonces el resultado, que lógicamente será siempre el mismo. Para que una aplicación sea útil, el valor mostrado debe ser dinámico, y responder a la solicitud del usuario, que no siempre será igual.

Existen diferentes mecanismos para que el usuario pueda proporcionar información de entrada a la aplicación, como veremos en la siguiente unidad. El más común en una aplicación web es el uso de formularios.

Otro de los modos de enviar información a nuestra aplicación web, es mediante el uso de parámetros en la url. Vamos a ver de forma sencilla cómo pasar estos parámetros y recoger estos en nuestra aplicación, de modo que podamos desarrollar ejercicios un poco más complejos.

Comenzaremos por algo muy sencillo, haremos que nuestra aplicación reciba nuestro nombre como parámetro en la url y muestre una página que nos salude.

Como se explicó con anterioridad, nuestra sencilla aplicación base contiene un único controlador *HomeController*, con un único método *Index()* que recibe y responde a nuestras peticiones.

Por defecto este método no está definido para recibir ningún parámetro, como queremos que sea capaz de recibir nuestro nombre, añadiremos un parámetro de tipo string. Una vez recibido el parámetro a través de la url, lo que haremos es meter el valor recibido en nuestro ViewData de modo que la página de la vista pueda obtener ese dato y ser capaz de utilizarlo para mostrar un saludo.

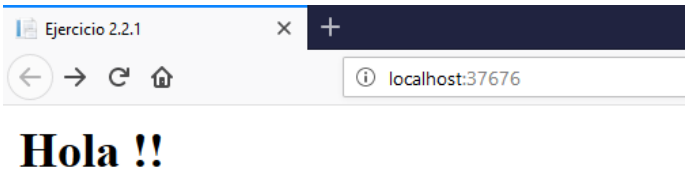
```
public IActionResult Index(string name)
{
    ViewData["name"] = name;

    return View();
}
```

Nuestra página de vista será en este caso muy simple:

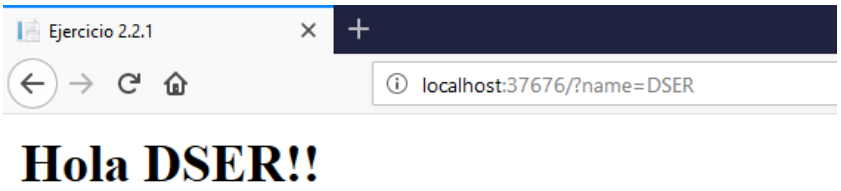
```
<html>
<head>
    <title>Ejemplo unidad 2.3</title>
</head>
<body>
    <h2>Hola @ViewData["name"]!!</h2>
</body>
</html>
```

Si ejecutamos ahora nuestra aplicación veremos que no funciona como esperábamos..



¿Cuál es el problema? Pues ninguno, todo funciona correctamente, pero como seguramente ya te hayas dado cuenta, lo que ocurre es que no hemos pasado a nuestra aplicación ningún parámetro. Efectivamente si nos fijamos en la url vemos que nos falta el parámetro, por lo que nuestro método ha pasado una cadena vacía a la vista.

En el navegador agregamos un parámetro llamado name al final de la url: `/?name=DSER`



Ahora vemos que todo ha funcionado como esperábamos. ASP.NET Core responde a la petición web, ejecutando el método correspondiente. Como en nuestro caso ve que el método espera un parámetro llamado name, de tipo string, busca ese parámetro. Más adelante veremos que ese parámetro puede llegar de diversas maneras, pero en este caso lo hemos especificado en la url. ASP.NET Core toma el valor de los parámetros especificados en la url y los pasa al método del controlador pasando el valor correspondiente a los parámetros del método, realizando la conversión del tipo de datos en caso necesario. De este modo vemos que es muy sencillo pasar valores a nuestra aplicación. A partir de ahora utilizaremos este mecanismo para nuestros ejercicios.

Ejercicio 2.3.1.

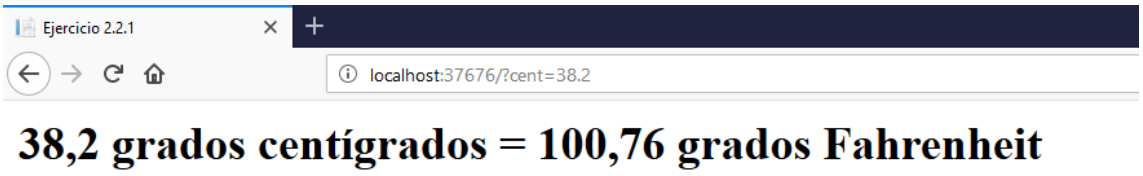
Queremos una aplicación web que convierta una temperatura en grados centígrados a grados Fahrenheit. Para ello debemos pasar la temperatura en grados como un parámetro de nombre *cent* en la url.

Ejemplo: `http://localhost:37676/?cent=38`

Nuestro HomeController debe tener un método que calcule y devuelva el valor en grados Fahrenheit a partir de los grados centígrados que recibirá como parámetro. La fórmula para la conversión es la siguiente:

Grados Fahrenheit = 9/5 * grados centrígrados + 32

El resultado debe ser similar al siguiente:



Ejercicio 2.3.2.

Queremos calcular el número de Tarot de una persona, que se obtiene sumando las cifras de la fecha de nacimiento una y otra vez, hasta que se reduzca a un único dígito.

Por ejemplo: 12/12/1900 = 1 + 2 + 1 + 2 + 1 + 9 = 16 = 1 + 6 = **7**

Para esto pasaremos a la aplicación 3 parámetros, day, month y year. Debemos crear una función que sea capaz de calcular el número del Tarot a partir de estos datos, el día, mes y año de nacimiento.

La salida será similar a la que se muestra:

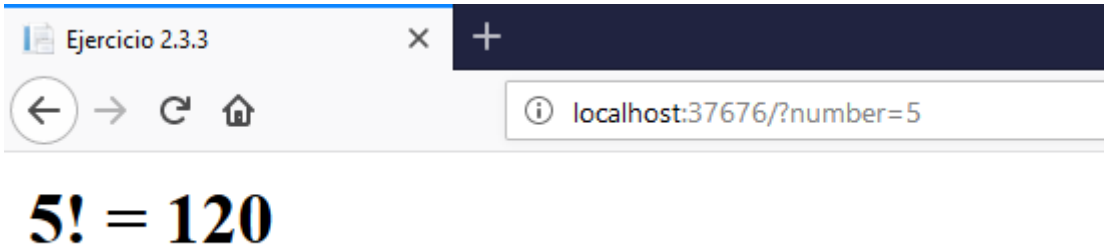
Ejercicio 2.3.3.

Para este ejercicio incluiremos un parámetro *number* en la url, de modo que pasemos un número entero a nuestra aplicación. El resultado será una página con el factorial de dicho número.

Para ello utilizaremos un método en el *HomeController*, que dado un número entero nos devuelva su factorial:

n! = 1 si n<=1
n! = n * (n-1)! si n > 1

El resultado sería similar al siguiente:



Ejercicio 2.3.4.

En este caso lo que queremos es una página donde se muestren todos los números primos entre uno y un número pasado como parámetro.

En este caso adoptaremos un enfoque diferente. Vamos a definir un método estático en el controlador, de modo que pueda ser ejecutado desde la vista, donde recorreremos los números y los mostraremos si son primos. Esto es así porque todavía no hemos visto de que modo pasar colecciones de datos a la vista.

La salida sería como la que se muestra a continuación: