# WEB API para calcular la resistencia equivalente de un circuito en serie o paralelo

**Santiago Dueñas Martínez**

**Trabajo presentado como requisito de evaluación parcial en la asignatura de Electromagnetismo al profesor**

**Juan Pacheco Fernández**

**Facultad de ingenierías y tecnologías**

**Ingeniería de Sistema**

**Universidad Popular del Cesar**

**Valledupar – Cesar**

**2019**

**WEB API para calcular la resistencia equivalente de un circuito en serie o paralelo**

## Objetivo general

Analizar e identificar los principios y procedimientos para calcular la resistencia equivalente de un circuito utilizando una WEB API.

## Conceptos físicos presentes en los circuitos con resistores

Los resistores o resistencias, son elementos que se oponen al paso de la corriente eléctrica en un circuito. Las resistencias son componentes habituales en los circuitos electrónicos. Su valor se encuentra marcado con un valor indicado o mediante un código de colores aceptado internacionalmente, dado que se fabrican en innumerables tamaños, valores y potencias.

## Agrupación de Resistores

**Agrupación en Serie:** Cuando dos o más resistores se conectan extremo con extremo a lo largo de una sola trayectoria, como se muestra en la figura 1, se dice que están conectados en serie

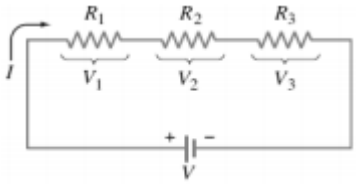


**Figura 1:** Resistores conectados en serie

Cualquier carga que pase a través de $R_1$ en la figura 1 también pasará a través de $R_2$ y luego a $R_3$. Por ende, la misma corriente I pasa a través de cada resistor. (Si no fuera así, esto implicaría que la carga no se conserva).

**Agrupación en paralelo:** Dos o más resistores se conectan en paralelo, si su conexión es como se muestra en la figura 2

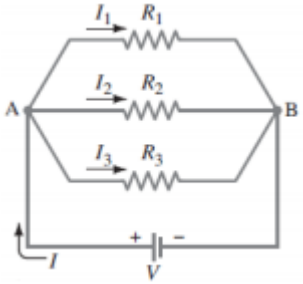


**Figura 2:** Resistores conectados en paralelo

En un circuito en paralelo como el de la figura 2, la corriente total I que sale de la batería se divide en tres trayectorias separadas. Sean $I_1$, $I_2$ e $I_3$ las corrientes a través de cada uno de los resistores, $R_1$, $R_2$ y $R_3$, respectivamente.

**Web API**

Una API es una interfaz de programación de aplicaciones (del inglés API: Application Programming Interface). Es un conjunto de rutinas que provee acceso a funciones de un determinado software.

Son publicadas por los constructores de software para permitir acceso a características de bajo nivel o propietarias, detallando solamente la forma en que cada rutina debe ser llevada a cabo y la funcionalidad que brinda, sin otorgar información acerca de cómo se lleva a cabo la tarea. Son utilizadas por los programadores para construir sus aplicaciones sin necesidad de volver a programar funciones ya hechas por otros, reutilizando código que se sabe que está probado y que funciona correctamente.

En la web, las API's son publicadas por sitios para brindar la posibilidad de realizar alguna acción o acceder a alguna característica o contenido que el sitio provee. Algunas de las más conocidas son las API's de:

- Google Search
- Flickr
- Del.icio.us
- Amazon
- Google Maps

**ASP.NET Core**

ASP.NET Core es un marco multiplataforma de código abierto y de alto rendimiento que tiene como finalidad compilar modernas aplicaciones conectadas a Internet y basadas en la nube. Con ASP.NET Core puede hacer lo siguiente:

- Compilar servicios y aplicaciones web, aplicaciones de IoT y back-ends móviles.
- Usar sus herramientas de desarrollo favoritas en Windows, macOS y Linux.
- Efectuar implementaciones locales y en la nube.
- Ejecutarlo en .NET Core o en .NET Framework.

**C#**

Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO

(ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

## Desarrollo

Se creo el proyecto bajo el nombre de “Electro” mediante la terminal de Windows y se procedió a crear los modelos que nos ayudaran a que el usuario pueda ingresar los datos de manera correcta, en este caso se crearon:

- Paralelo
- Resistencia

Donde paralelo hace la representación de todas las sumas que tiene cada resistencia, siendo está la ecuación: 1/R.

Y resistencia solamente almacena todas resistencia calculadas.

En este caso, no se hace un modelo a los circuitos en serie dado que solamente es hacer una suma.

```
namespace Electro.Models
{
    3 references
    public class Paralelo
    {
        1 reference
        public int Id {set;get;}
        3 references
        public float Resistenica {set;get;}
    }
}
```

```
namespace Electro.Models
{
    10 references
    public class Resistencia
    {
        7 references
        public int Id {get;set;}
        6 references
        public float Resisten {get;set;}

        5 references
        public bool Ok {get;set;}
    }
}
```

**Figura 3:** Clase Paralelo (modelo) y clase Resistencia

Y se procedió hacer el controlador para poder obtener nuestra API funcionando correctamente, en este caso, solamente vale la pena explicar como funciona calcular el método que almacena y calcula los circuitos en paralelo, dado que en serie solamente es una suma.

Se realizaron 3 métodos,

“PostResistenciaParaleloAdd(Paralelo item)”

El cual recibe un objeto tipo “Paralelo”, lo primero que hace es contar cuántos circuitos han sido calculados para poder incrementar el “Id” (el atributo que nos permite saber el orden donde se han guardado los circuitos calculados), segundo, crea un objeto “Resistencia”, se le asignan los valores correspondientes a este objeto, para después buscar un valor para el atributo “Id” del modelo “Paralelo” para asignarle un valor integral en

la base de datos y seguir ingresando valores temporales en “Paralelo”, para finalizar el método, se invoca el método:

“PostResistenciaParalelloSum(Resistencia item)”

Después de haber sido invocado lo que procede hacer este método es hacer la suma de los valores que tenga el atributo “resistencia” y crear un objeto temporal en la base de datos donde el atributo “Ok” va estar en “false” si el objeto esta creado entonces solamente lo modifica con la suma.

“PostResistenciaParaleloResult()”

Para finalizar, este solamente utiliza la siguiente formula: 1/R, siendo R la sumatoria de todas las resistencias almacenadas, mostrando el resultado.

```
[HttpPost("ParaleloAdd")]
0 references
public async Task<ActionResult<Paralelo>> PostResistenciaParaleloAdd(Paralelo item){
    int num =  _context.Resistencias.Count();
    item.Id= num+1;
    item.Resistenica = 1/item.Resistenica;
    _context.Paralelos.Add(item);
    Resistencia resistencia = new Resistencia();
    for (int i=1; i ==_context.Resistencias.Count(); i++){
        var auxResis = await _context.Resistencias.FindAsync(i);
        if (auxResis.Ok==false){
            resistencia.Id = auxResis.Id;
        }
        else {
            resistencia.Id = _context.Resistencias.Count()+1;
        }
    }
    resistencia.Ok = false;
    resistencia.Resisten = item.Resistenica;
    await this.PostResistenciaParalelloSum(resistencia);
    await _context.SaveChangesAsync();
    return Ok();
}

[HttpPost("ParaleloSum")]
1 reference
public async Task<ActionResult<Resistencia>> PostResistenciaParalelloSum(Resistencia item){
    var resaux = await _context.Resistencias.FindAsync(item.Id);
    if (resaux==null){
        _context.Resistencias.Add(item);
        await _context.SaveChangesAsync();
        return Ok();
    } else {
        resaux.Resisten=resaux.Resisten+item.Resisten;
        await _context.SaveChangesAsync();
        return Ok();
    }
}
```

```
[HttpPost("ParaleloResult")]
0 references
public async Task<ActionResult<Resistencia>> PostResistenciaParaleloResult(){
    int id=0;
    for (int i=1; i==_context.Resistencias.Count(); i++){
        var aux = await _context.Resistencias.FindAsync(i);
        if (aux.Ok==false){
            id = i;
        }
        else {
            return NotFound();
        }
    }
    var item = await _context.Resistencias.FindAsync(id);
    if (item!=null){
        item.Resisten = 1/item.Resisten;
        item.Ok=true;
        int valor = _context.Paralelos.Count();
        for (int i=1; i==valor;i++){
            var paralelo = await _context.Paralelos.FindAsync(i);
            _context.Paralelos.Remove(paralelo);
        }
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetResistencia), new {id=item.Id}, item);
    }
    else {
        return NotFound();
    }

}
```

**Figura 4:** Métodos utilizados.

## Prueba

Como no se realizó una interfaz, se decidió utilizar Swagger para utilizar y ver el funcionamiento de la API.

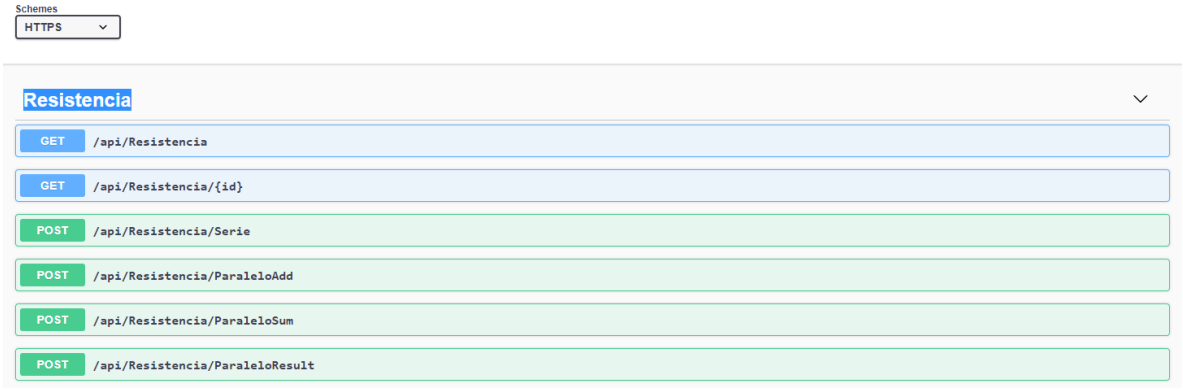


**Figura 5:** Interfaz de Swagger

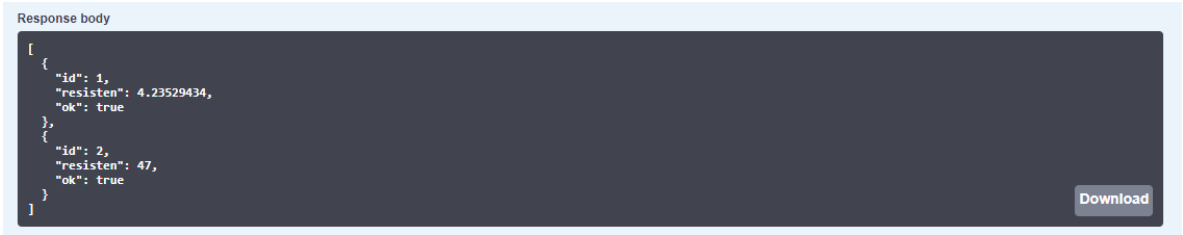


**Figura 6:** Resultado de la prueba

## Bibliografía


1. Serway Raymond, Física para las ciencias y la ingeniería, séptima edición, Tomo II, Editoial Adisson Wesley
2. Halliday, D., Resnick, R. y Krane, K. Física, volumen 2, cuarta edición, Editorial Continental S.A., México D.F. México 1999.
3. Física Conceptual. Paul Hewith, Décima edición , Editoial Adisson Wesley, México 2007.
4. Anya Miyeth Bolaño Arias, https://docs.google.com/a/unicesar.edu.co/viewer?a=v&pid=sites&srcid=dW5pY2VzYXIuZWR1LmNvfGFueWFib2xhbm98Z3g6NDk5NTdlMjAxMzI3M2Y5MQ