

# Ruteo

---

Ruteo + URL Semántica

# Ejemplo

---

Creemos una calculadora básica usando PHP.



Esta calculadora tiene que:

- Realizar operaciones básicas
- Mostrar el número PI
- Mostrar el About de quienes crearon la calculadora, y se debe poder ver un desarrollador en particular.



CÓDIGO:

[TANDIL](#)

[TRES ARROYOS](#)

# URLs

---

## Cómo quedaron las URL's?

- About & PI

- [www.calculadora.com/about.php](http://www.calculadora.com/about.php)
- [www.calculadora.com/about.php?member="juan"](http://www.calculadora.com/about.php?member=juan)
- [www.calculadora.com/pi.php](http://www.calculadora.com/pi.php)

- Calculadora

- [www.calculadora.com/calculadora.php?operacion=suma&a=5&b=9](http://www.calculadora.com/calculadora.php?operacion=suma&a=5&b=9)
- [www.calculadora.com/calculadora.php?operacion=resta&a=300&b=19](http://www.calculadora.com/calculadora.php?operacion=resta&a=300&b=19)
- [www.calculadora.com/calculadora.php?operacion=division&a=9&b=3](http://www.calculadora.com/calculadora.php?operacion=division&a=9&b=3)

# ¿Que problema tiene esto?

1. Por cada acción nueva que quiero agregarle al sistema, tengo que **crear un archivo nuevo php**.
  - Necesitamos una forma de que cada acción NO se mapee directamente a un archivo físico. (ENRUTAMIENTO)
2. Las URL's no se entienden y esto es **malo para SEO**
  - Es importante que los sistemas utilicen URL's semánticas (amigables).

# URL's Semánticas

---

Las URL semánticas o amigables son aquellas URL que son entendibles para el usuario.

<http://www.exa.unicen.edu.ar/index.php?hl=es&p=ingresantes>



<http://www.exa.unicen.edu.ar/es/ingresantes>

# URL's Semánticas

---

URLs semánticas (amigables o *pretty urls*)

- Fáciles de **entender** para los usuarios
- Mejoran el **posicionamiento** web
- Proporcionan información sobre la **estructura** del sitio
- Fáciles de **compartir**, ej: whatsapp, llamada, divulgación
- Más **estéticas**, ej: imprimirlas en folletos, facebook, etc.

Ejemplo aplicado en nombre de usuario en Twitter:

<https://twitter.com/starwars>

# Ejemplo

---

Modifiquemos nuestra calculadora para que acepte URL's semánticas:



Queremos que las URL de la calculadora sean:

[www.calculadora.com/sumar/2/3](http://www.calculadora.com/sumar/2/3)

[www.calculadora.com/sumar/3/4](http://www.calculadora.com/sumar/3/4)

[www.calculadora.com/restar/10/4](http://www.calculadora.com/restar/10/4)

No puedo tener un archivo PHP para todas las combinaciones posibles de una **operación**, por lo tanto es necesario primero **enrutar** la aplicación.

# Routing

---

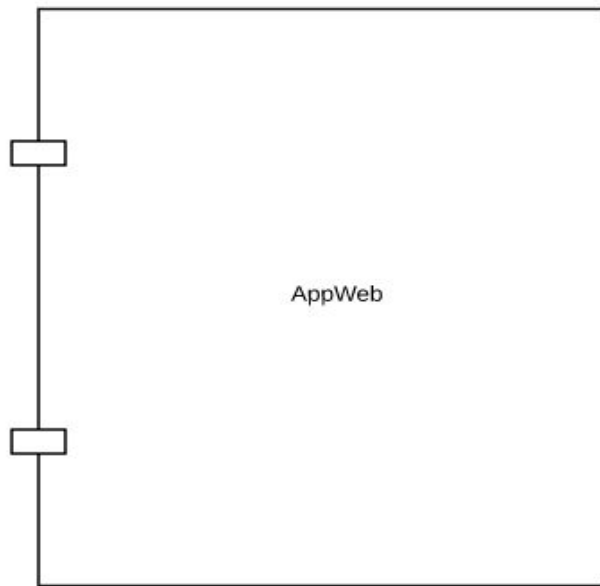
El **routing** (ruteo o enrutamiento) en un Sistema WEB es el mecanismo por el cual cada solicitud del usuario especificada por una URL y un método HTTP es dirigida a un componente de código encargado de atenderlas.

- Se encarga de determinar el PATH a donde redireccionaremos.
- Implica romper la lógica de “cada URL es un archivo”.

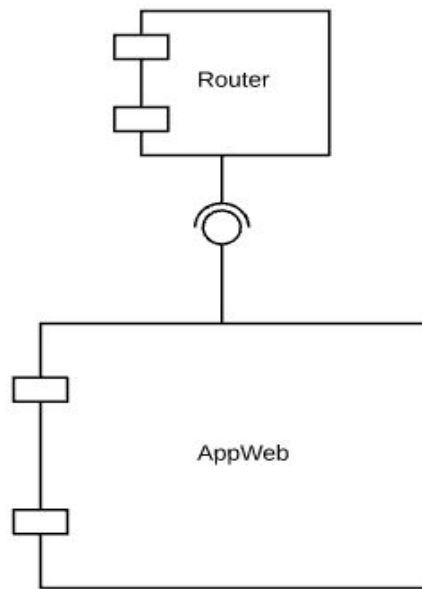


# Routing

Necesitamos un componente principal (**roteador**) que atienda TODOS los request.



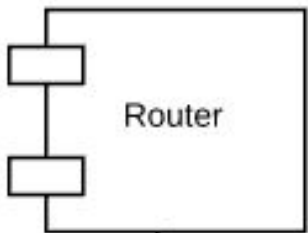
ANTES



AHORA

# Router

---



Vamos a crear un archivo “router.php” que encapsule el comportamiento del componente ruteador:

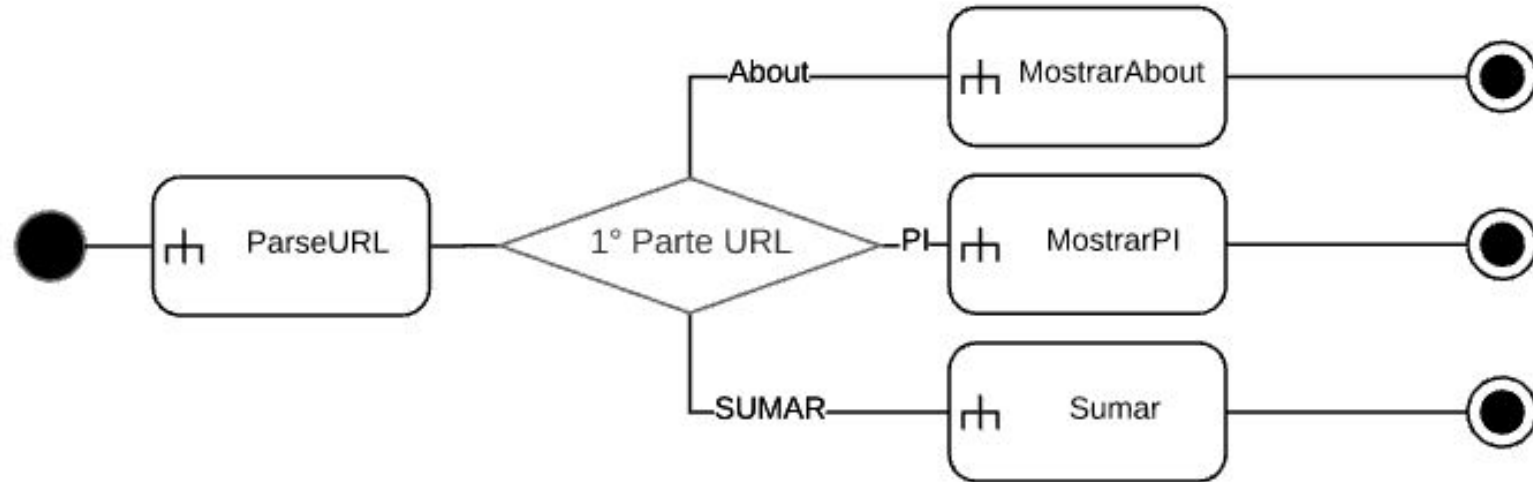
Debe leer una **acción** y una lista de **parámetros** => **:action/[:a/:b]**

Ejemplo:

- [www.calculadora.com/router.php?action=sumar/2/3](http://www.calculadora.com/router.php?action=sumar/2/3)
- [www.calculadoraonline.com/router.php?action=pi](http://www.calculadoraonline.com/router.php?action=pi)
- [www.calculadoraonline.com/router.php?action=about/juan](http://www.calculadoraonline.com/router.php?action=about/juan)

# Ruteo

---



# route.php



```
<?php
```

```
// lee la acción
if (!empty($_GET['action'])) {
    $action = $_GET['action'];
} else {
    $action = 'home'; // acción por defecto si no envían
}

// parsea la acción Ej: suma/1/2 --> ['suma', 1, 2]
$params = explode('/', $action);

// determina que camino seguir según la acción
switch ($params[0]) {
    case 'home':
        home();
        break;
    case 'sumar':
        sumar($params[1], $params[2]);
        break;
    case 'restar':
        restar($params[1], $params[2]);
        break;
    case 'pi':
        showPi();
        break;
    default:
        echo('404 Page not found');
        break;
}
```

# Manos a la obra

---

Modifiquemos nuestra calculadora para que enrutarla y que entienda URLs semánticas.

LIVE  CODING



<https://gitlab.com/unicen/Web2/livecoding2019/bolivar/calculadora/commit/1a88443afb0b41d22bc8fc9fea683a972579e86d>  
<https://gitlab.com/unicen/Web2/livecoding2019/tandil/calculadora/commit/d7d2471eec0eb0db8025b76cdebb8246e5c00e2f>

# Reescribiendo Rutas

---

Ya tenemos lo más importante, el ruteo! Pero las URL's siguen siendo **casi** como antes:

**/route.php?action=sumar/2/5**

Cómo hacemos para que esas rutas sean “amigables”?

## Reglas Apache

Usamos reglas de servidor (Apache en este caso) para “**enmascarar**” la url y no mostrar el archivo router.php en la url.



**Apache**

# Reglas Apache

En el archivo **.htaccess** indicamos qué URLs van a que archivo PHP. En general vamos a redirigirlas a un código que sepa procesar la URL (router.php).




# .htaccess

---

El archivo **.htaccess** es un **archivo de configuración** de Apache HTTP web server.

→ Permite configurar opciones a nivel de directorio

## Aplicaciones

- Prevenir hotlinking
- Bloquear usuarios por IP
- Documentos de error
- Redirigir durante mantenimiento
- Ocultar listado del directorio
- **Ruteo** 



# .htaccess

Este archivo va en la carpeta base o la de URL base a rutear.

Redirigimos la solicitud a un único archivo.

```
<IfModule mod_rewrite.c>
```

```
    RewriteEngine On
```

```
    RewriteCond %{REQUEST_FILENAME} -f [OR]
```

```
    RewriteCond %{REQUEST_FILENAME} -d
```

```
    RewriteRule \.(:css|js|jpe?g|gif|png)$ - [L]
```

```
    ...
```

```
</IfModule>
```

Permiso reescritura  
(ruteo) de URLs

Si existe el archivo o  
directorio entonces se  
procede a la siguiente Rule

Dejamos que el contenido  
estático sea accedido con el  
método por defecto

Expresión Regular

# .htaccess

---

```
<IfModule mod_rewrite.c>
```

```
...
```

# Con "." decimos que capture todos los caracteres restantes.

# Los paréntesis agregan lo capturado a una variable: \$1 por ser la primera.

```
RewriteRule ^(.*)$ route.php?action=$1 [QSA,L]
```

```
</IfModule>
```



Expresión Regular

(en este caso representa cualquier cadena), es decir, cualquier URL.

<http://regexr.com/3ghl2>

# URL's

---

Ahora podemos cambiar todas nuestras URL's por las nuevas:

ANTES

```
<ul>
  <li><a href="index.php">Calculadora</a></li>
  <li><a href="pi.php">El número Pi</a></li>
  <li><a href="about.php">About</a></li>
</ul>
```

**AHORA**

```
<ul>
  <li><a href="home">Calculadora</a></li>
  <li><a href="pi">El número Pi</a></li>
  <li><a href="about">About</a></li>
</ul>
```

# URL's y acceso a datos estáticos

---

Cuando queremos entrar a [www.calculadora/about/juan](http://www.calculadora/about/juan) no nos muestra la imagen de Juan.

→ El problema es que está intentando encontrar la imagen en: /about/images/juan.jpg.

¿Cómo podemos arreglar esto?

- Rutas absolutas?
- Cambiamos de lugar las imágenes?

# Tag Base



## Tag Base <base>

En lugar de usar rutas absolutas, usamos el tag **base**, que va dentro de **head**.

- Nos indica cual es la **base** de nuestro sitio. Ej: [www.calculadora.com](http://www.calculadora.com)
- En nuestro caso será algo así: <http://localhost/calculadora/>

<head>

<base href="http://localhost/calculadora/" target="\_blank">

....

ESA MALA PRÁCTICA



*Pero esa url solo funciona en nuestra computadora!!!*

# ¿Como saber la base de nuestro sitio?

---

PHP nos da datos del server en la variable **\$\_SERVER**.

Para crear la base de nuestro sitio:

```
define('BASE_URL', '//' . $_SERVER['SERVER_NAME'] . ':' .  
$_SERVER['SERVER_PORT'] . dirname($_SERVER['PHP_SELF']) . '/');
```

En el head html:

```
<base href="'.BASE_URL.'">
```

- **SERVER\_NAME**: Nombre del server (localhost)
- **SERVER\_PORT**: Nro puerto server (por default no se vé)
- **PHP\_SELF**: El script que se está ejecutando.
- **dirname()**: Nos devuelve el directorio del script que le pasemos por parametro.

**“es difícil pero bonito...”**

**DEMO**

**A MEDIDA QUE DAMOS LAS CLASES DEJAMOS ACA  
LINKS AL COMMIT DE GIT HASTA ESTE PUNTO**

**TRES ARROYOS:** <https://gitlab.com/unicen/Web2/livecoding2020/tres-arroyos/calculadora>

**BOLIVAR:** <https://gitlab.com/unicen/Web2/livecoding2019/bolivar/calculadora>

**TANDIL:** <https://gitlab.com/unicen/Web2/livecoding2019/tandil/calculadora>

Y si queremos un router más  
prolijo?





# Routing

---

Una alternativa, agregar las acciones disponibles en un archivo de configuración

## config/ConfigApp.php

```
class ConfigApp{

    public static $ACTION = "action";
    public static $PARAMS = "params";
    public static $ACTIONS = [
        'home' => "mostrar_about",
        'about' => "mostrar_about_by_name",
        'pi' => "number_pi",
        'sumar' => "sumar"
    ];
}
```

# Routing

---

Versión mejorada!

- Se utilizan las constantes ACTION y PARAMS
- Retorna un array donde:
  - Array[0]: Tiene la Accion
  - Array[ 1]: Es la lista de parametros que vienen despues de la acción

```
function parseUrl($url){  
    $arr_data = explode ("/", $url);  
    // Se guarda el nombre de la accion  
    $arrayReturn[ConfigApp::$ACTION] = $arr_data[0];  
    // Se guarda la lista de parametros como un array  
    $arrayReturn[ConfigApp::$PARAMS] = isset($arr_data[1]) ? array_slice($arr_data, 1) : null;  
  
    return $arrayReturn;  
}
```

routeAvanzado.php

# Routing

---

Versión mejorada. ¿Por qué creen que es mejor?

- Meta-programacion
  - \$methodName

```
// Parsear la URL para identificar Actions y Parametros
$urlData = parseUrl($_GET[ConfigApp::$ACTION]);
// Nombre de la Accion a ejecutar
$actionName = $urlData[ConfigApp::$ACTION];

if (array_key_exists($actionName,ConfigApp::$ACTIONS)){
    $params = $urlData[ConfigApp::$PARAMS];
    $methodName = ConfigApp::$ACTIONS[$actionName];

    if(isset($params) && $params != null){
        // Invocar el metodo con el array de parametros
        $methodName($params);
    } else {
        $methodName();
    }
} else {
    // No existe La Action entonces muestro La Home
    mostrar_about();
}
```

routeAvanzado.php

## URL's amigables con parámetros GET

---

Es recomendable mantener la **url de los recursos base** lo más “limpia” posible. A veces podemos tener algunos parámetros que aún se vean como parámetros GET.

**Búsqueda** avanzada, **filtros** complejos y **ordenamiento** de recursos son candidatos perfecto para mantener urls amigables en combinación con parámetros get.

# Diseño de URLs

---

Ejemplos:

- misitio.com/**tickets**?sort=price
- misitio.com/**tickets/search**?term=Roger Waters
- misitio.com/**tickets**?status=open



**RECURSO PRINCIPAL**  
Indexado por buscadores

**MODIFICADORES (Query Params)**  
Modifican el orden o filtran el  
recurso principal

# SEO: URLs Canónicas

---

Si hay muchas URL que llevan a lo mismo puede perjudicar el SEO porque el buscador detecta contenido duplicado. Hay que indicarle al buscador que son lo mismo. Se usa el rel “canonical” en el head.

```
<head>
```

```
...
```

```
<link rel="canonical"
```

```
href="/productos/buscar?q=tareas" />
```

```
</head>
```

## Links reales en el browser

```
/productos/buscar?q=tareas&page=1
```

```
/productos/buscar?q=tareas&page=1&order=desc
```

# Referencias

---

- [Buenas practicas en API RESTs con Pretty URLs](#)
- [URLs Canonicas](#)
- [Htaccess Documentación](#)
- [Ejemplo Completo](#)
- [15 Buenas Prácticas](#)
- [Infografia SEO URLs](#)