

# Templates Engine

---

# Problema #1

---

Mostramos los datos de los animales mamíferos en una tabla.

```
function showMamiferos($animales) {  
    echo "<table>";  
    foreach($animales as $animal){  
        if(es_mamifero($animal)) {  
            echo "<tr>  
                <td>" . $animal->nombre . "</td>  
                <td>" . $animal->peso . "</td>  
                <td>" . $animal->tamano . "</td>  
            </tr>";  
        }  
    }  
    echo "</table>";  
}
```

El cliente nos dice que ahora quiere que sea una lista. Debemos reescribir esa ensalada de PHP y HTML.

**Pero la lógica en sí no cambió!**

Lo mismo pasa si quiero dos páginas con los mismos datos mostrados de diferente forma (ej, lista y tabla)



# Hasta ahora

---

PHP es un intérprete de scripts.

- Aunque podemos mezclar HTML y PHP y funciona, es difícil de mantener si tenemos mezcla de lenguajes.
- PHP funciona bien para programar, pero no para manejar **html**.

SOLUCIÓN  
**TEMPLATE ENGINE**

# Template engine (Motor de templates)

---

Los “Template Engine” son herramientas que se utilizan para separar la **lógica del programa** y la **presentación del contenido** en dos partes independientes.

## VENTAJAS

- Facilita el desarrollo tanto de la lógica como de la presentación.
- Mejora la flexibilidad.
- Facilita la modificación y el mantenimiento.

# Template engine - ¿Por qué son importantes?

---

¡Separa la **lógica** de la **vista**!

En otras palabras...

El código (PHP) se separa de la presentación (HTML)



# ¿A qué llamamos lógica?



# Lógica

---

La **lógica de una aplicación** es la parte del código que realiza todo lo referido a la obtención, almacenamiento y procesamiento de los datos para entregarlos a una **vista** que sabe cómo visualizarlos.

Se dice que es el “detrás de escena” necesario para poder presentar los datos en pantalla.



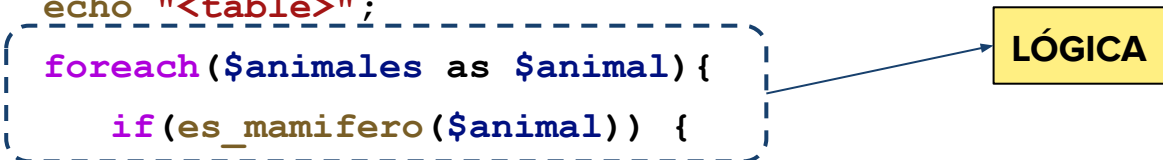


# Problema #1

---

## En el problema #1 ¿Qué es lógica y que es vista?

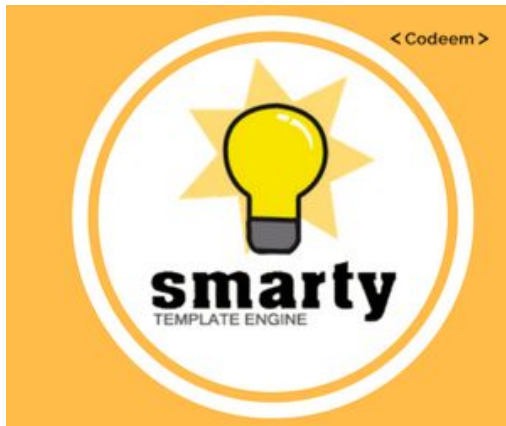
```
function showMamiferos($animales) {  
    echo "<table>";  
    foreach($animales as $animal){  
        if(es_mamifero($animal)) {  
            echo "<tr>  
                <td>" . $animal->nombre . "</td>  
                <td>" . $animal->tamano . "</td>  
            </tr>";  
        }  
    }  
    echo "</table>";  
}
```



A diagram illustrating the concept of logic in programming. A dashed blue box highlights the inner loop of the PHP function `showMamiferos`, specifically the `foreach` loop and the `if` statement that checks `es_mamifero($animal)` and echoes the table rows. A blue arrow points from this dashed box to a yellow rectangular box labeled **LÓGICA** (Logic).

# Template Engine - Alternativas

---



SMARTY

<https://www.smarty.net/>



DWOO

<http://dwoo.org/>



TWIG

<https://twig.symfony.com/>



VOLT PHALCON

<https://docs.phalconphp.com/ar/3.2/volt>

{Smarty}

---

# ¿Qué es Smarty?

---

- **Smarty** es un motor de plantillas para PHP.
- Es rápido y eficiente.
- La plantilla es compilada solo una vez.



# Con Smarty

---

Smarty usa una combinación de etiquetas HTML y **etiquetas de plantilla** para formatear la presentación del contenido.

**Etiquetas de plantilla** -> utilizan el formato de **{tags}**

- La idea siempre es:
  - Mantener separada la presentación (menor acoplamiento posible).
  - Mismo objetivo que CSS separado del HTML!
  - El menor overhead posible.



# Usar Smarty

---

## CREAR LA LÓGICA DE LA APLICACIÓN

Obtiene la información (BBDD, files, etc), se la procesa y se asigna el contenido a mostrar en variables.

**.PHP**

**NO TIENE INFORMACIÓN DE CÓMO VA A SER MOSTRADO EL CONTENIDO**



## CREAR EL TEMPLATE DE PRESENTACIÓN

Recibe la información y se muestra mediante una combinación de tags html y tags de plantillas.

**.TPL**

**NO SE ENCARGA DE OBTENER NI PROCESAR EL CONTENIDO**

## ¿Cómo lo uso?

### 1. Lo descargo e incluyo en mi sitio

<http://www.smarty.net/download>

### 2. Lo incluyo

```
require_once('libs/smarty/Smarty.class.php');
```

### 3. Lo instancio

```
$smarty = new Smarty();
```

### 4. Creo el template

```
templates/animales.tpl
```

### 5. Asigno variables

```
$smarty->assign('titulo',"Lista de mamíferos");
```

```
$smarty->assign('animales', $mamiferos);
```

### 6. Renderizo el template

```
$smarty->display('templates/animales.tpl'); // muestro el template
```

# Problema #1 - Con Smarty

animales.php

```
<?php
require_once('libs/smarty/Smarty.class.php');

function showMamiferos($animales) {
    $mamiferos = array(); // arreglo para guardar solo los mamiferos

    foreach($animales as $animal) {
        if(es_mamifero($animal))
            array_push($mamiferos, $animal);
    }

    // inicializo Smarty y asigno las variables para mostrar
    $smarty = new Smarty();
    $smarty->assign('titulo',"Lista de mamíferos");
    $smarty->assign('animales', $mamiferos);

    $smarty->display('templates/animales.tpl'); // muestro el template
}
```



# Problema #1 - Con Smarty (mostrando un arreglo)

---

animales.tpl

```
<h1>{$titulo}</h1>
```

```
<table>
```

```
  {foreach from=$animales item=animal}
```

```
    <tr>
```

```
      <td>{$animal->nombre}</td>
```

```
      <td>{$animal->peso}</td>
```

```
      <td>{$animal->tamano}</td>
```

```
    </tr>
```

```
  {/foreach}
```

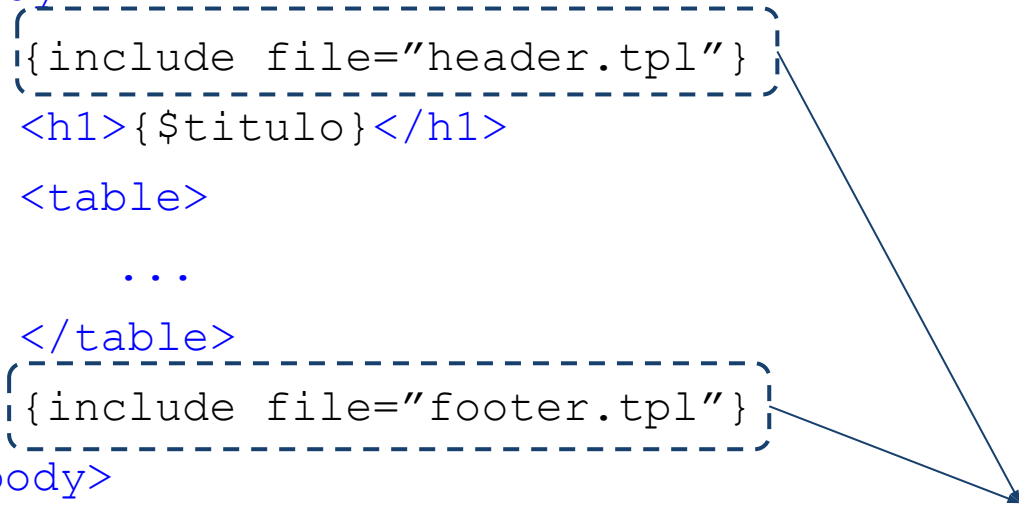
```
</table>
```

# Plantilla - Include

---

```
<html>
<head>
  <title>{$titulo}</title>
  <link rel="stylesheet" type="text/css"
href="css/bootstrap.css">
</head>

<body>
  {{include file="header.tpl"}}
  <h1>{$titulo}</h1>
  <table>
    ...
  </table>
  {{include file="footer.tpl"}}
</body>
<html>
```



**PODEMOS INCLUIR  
TEMPLATES DENTRO DE  
OTROS!!!**

# Características

---

Smarty se enfoca en tener:

- **Plantillas rápidas**
- **Poco código**
- **Mantenibles**

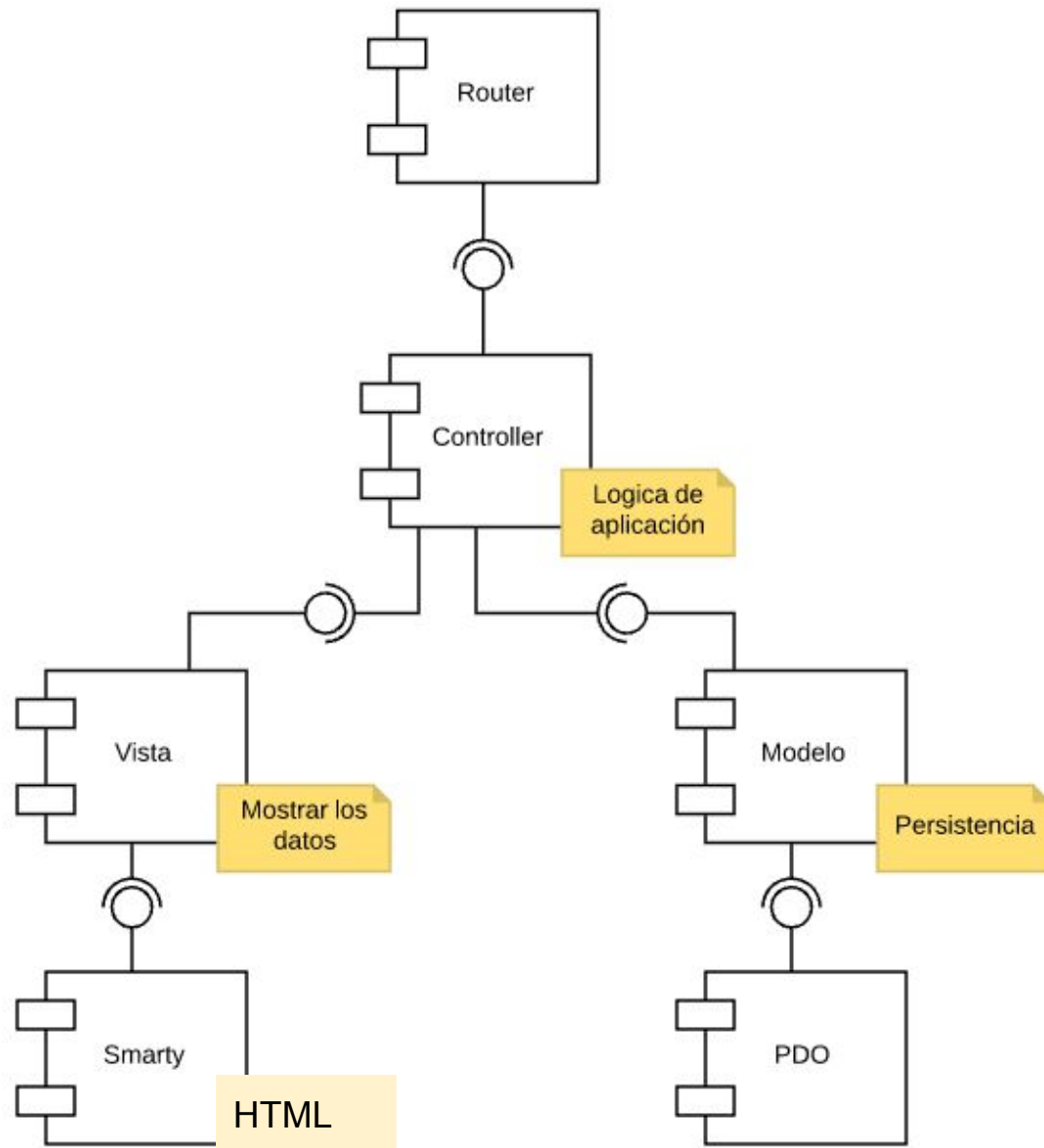
Smarty ofrece una enorme cantidad de funciones y herramientas para facilitarle el trabajo al desarrollador:

- Funciones
- Modificadores
- Plugins
- etc

AHORA A LEER DOCUMENTACIÓN

<https://www.smarty.net/docs/en/>

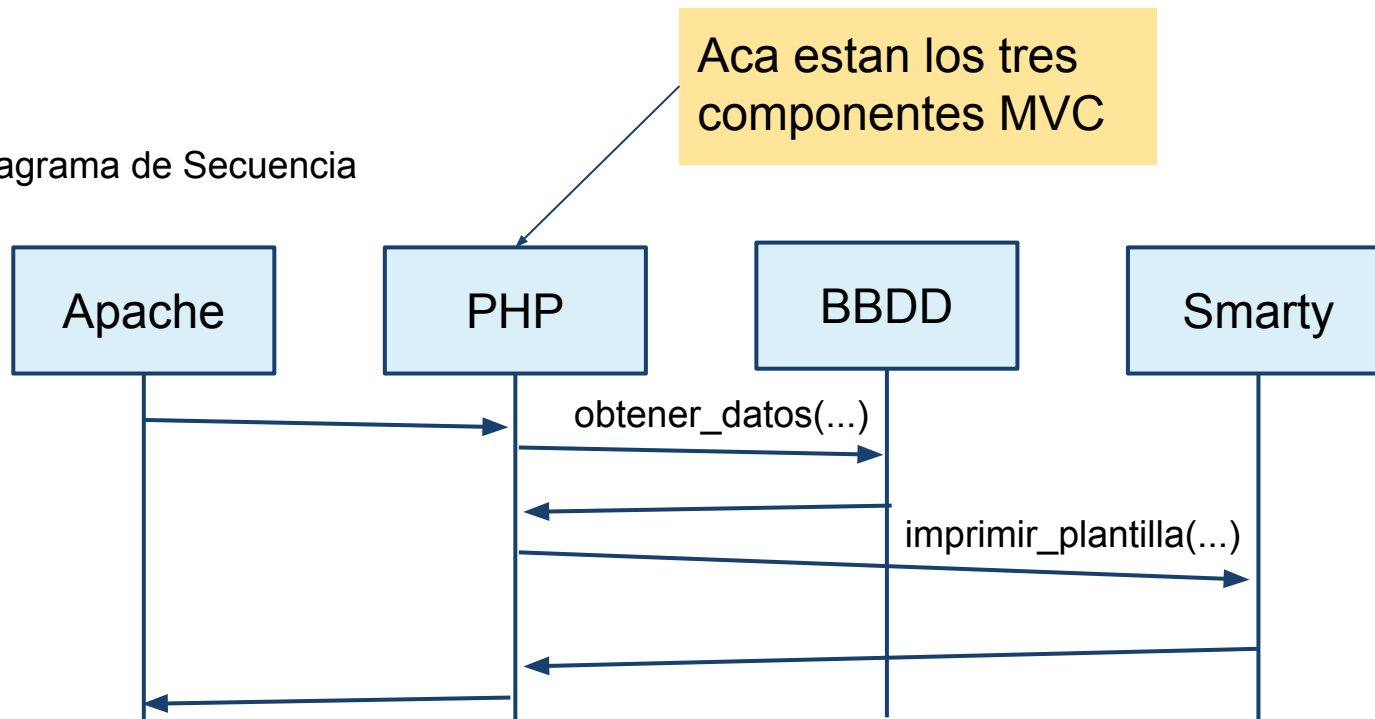
# Arquitectura



# Ejecución

- Accedemos a la página animales.php
  - PHP ejecuta la lógica de negocio y de Smarty.
  - Smarty se encarga de abrir la plantilla y plasmar los datos.

Diagrama de Secuencia



Manos a la obra!

## Ejemplo - Lista de tareas

---

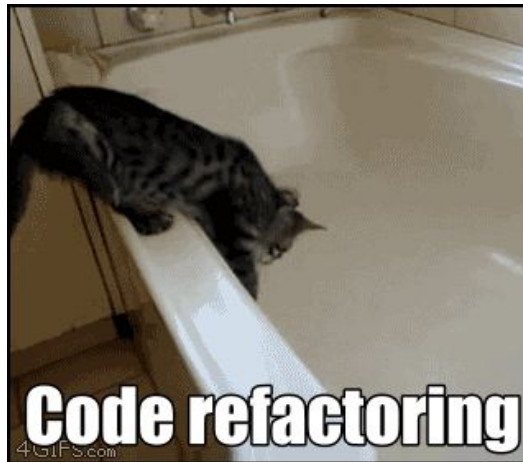
Vamos a modificar la aplicación que permite administrar una lista de tareas para separar la lógica del HTML usando Smarty.



**KEEP  
CALM  
AND  
REFACTOR  
CODE**

La refactorización (refactoring) es una técnica de la ingeniería de software para **reestructurar un código fuente**, mejorando su estructura interna **sin cambiar su comportamiento externo**.

“Código más bonito, que hace lo mismo.”





# Instalar Smarty

- **¿Dónde se baja?**

Desde el sitio oficial: <http://www.smarty.net/download>

Asegurémonos de bajar la última versión 3.+

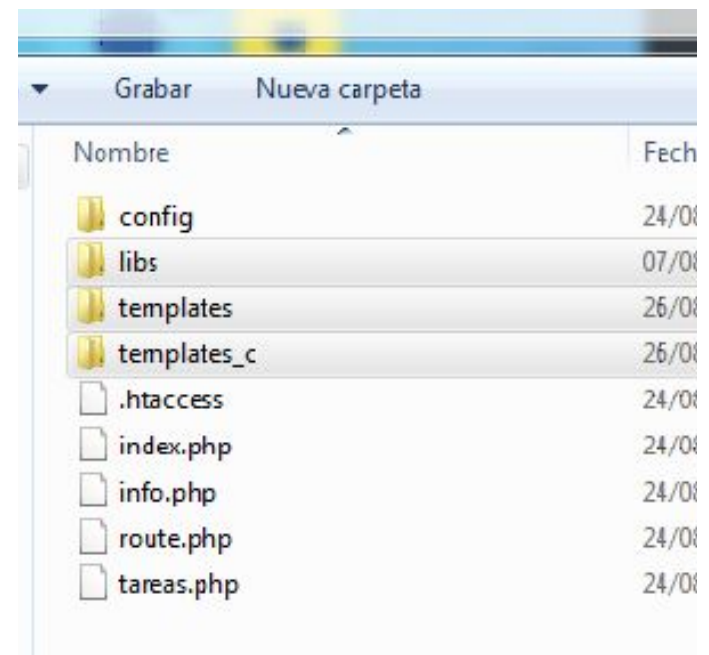
- **¿Cómo lo incluyo en mi proyecto?**

**1. Copiamos el contenido de la carpeta “libs” en nuestro proyecto -> libs/smarty**

**2. Creamos una carpeta “templates”**

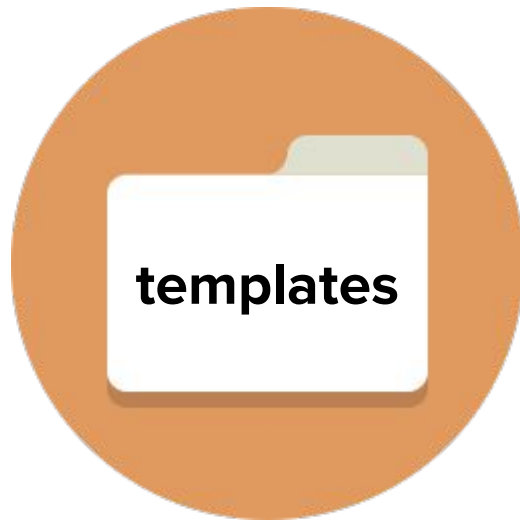
**3. Creamos una carpeta “templates\_c”**

En linux hay que darle permisos de escritura a apache sobre esta carpeta



# Instalar Smarty - Templates folder

---

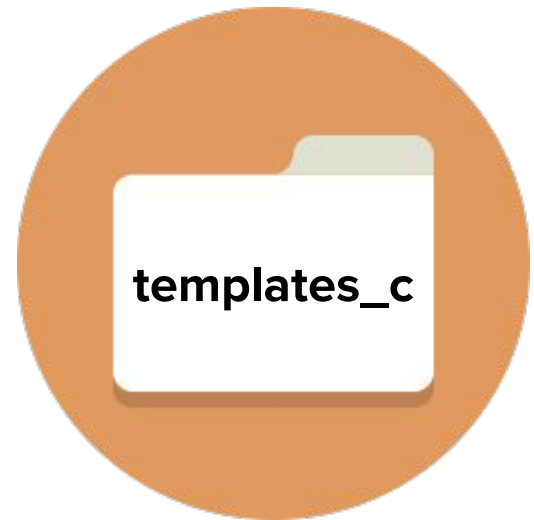
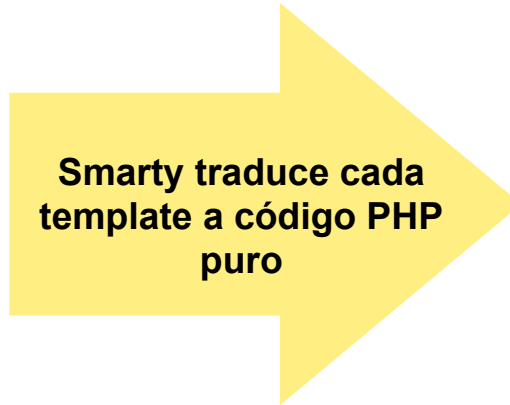


**ESTA CARPETA VA A CONTENER LAS PLANTILLAS DE NUESTRO PROYECTO**

Las creamos nosotros usando una combinación de tags html y tags de plantilla.



SON UNA PARTE FUNDAMENTAL DE NUESTRO PROYECTO



**ESTA CARPETA VA A CONTENER LAS PLANTILLAS COMPILADAS**

Se compilan automáticamente cada vez que el template original se crea o modifica.



NO SE SUBEN A GIT. SMARTY SE ENCARGA DE GENERARLAS Y MANTENERLAS

# Vista con Smarty

---

Modifiquemos la Vista de Tareas para **separar lógica de HTML**.

1. Crear **header.tpl**
2. Crear **footer.tpl**
3. Crear **taskList.tpl** (que incluya header.tpl y footer.tpl)
4. Modificar el método de la vista para que llame al template **taskList.tpl**

# taskList.tpl

```
<div class="container">
  <div class="page-header">
    <h1>Lista de Tareas</h1>
  </div>
  <div class="row">
    <div class="col-md-6">
      <label class="control-label" for="nombre">Tarea</label>
      <ul class="list-group">
        {foreach $tareas as $tarea}
          <li class="list-group-item">
            {$tarea}
          </li>
        {/foreach}
      </ul>
    </div>
  </div>
</div>
```

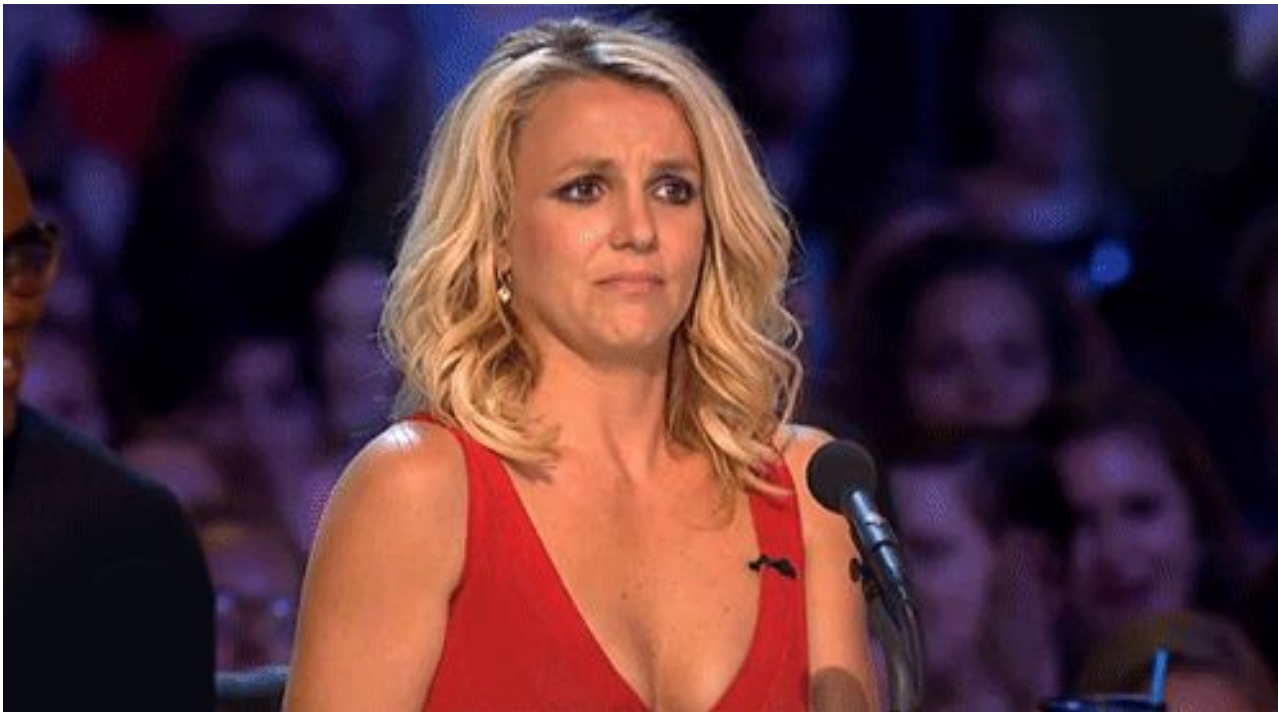
← Titulo

← Lista de Tareas

# Modificadores

---

Ahora queremos que los títulos o descripción se vean en mayúsculas y solo se muestren los primeros 20 caracteres.



# Modificadores

---

## Todo en mayúsculas

`{$nombre|upper}`

## Todo en minúsculas

`{$nombre|lower}`

## Capitalizado

`{$nombre|capitalize}`

## Sin decimales

`{$sueldo|string_format:"%d"}`

## Con N decimales

`{$sueldo|string_format:"%.Nf"}`

## Texto separado de a N letras

`{$overview|wordwrap:30:"<br />\n"}`

## Truncado

`{$titulo|truncate:30}`

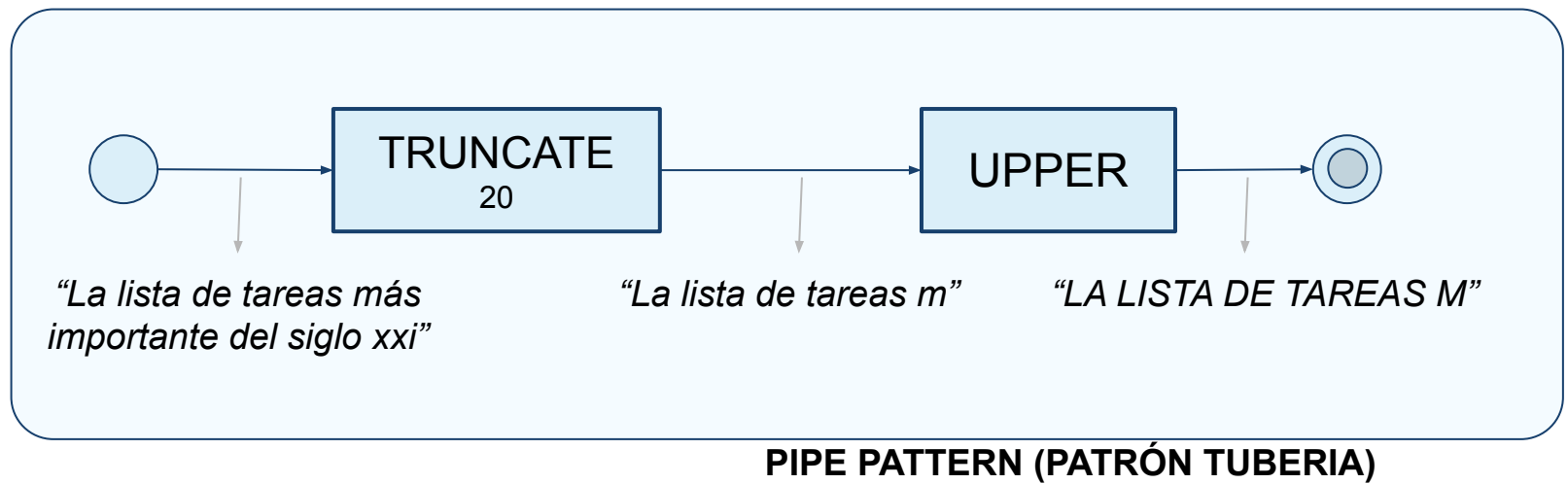
# Modificadores

## PHP

```
<?php echo strtoupper(substr($title,0,20)); ?>
```

## Smarty

```
{ $title|truncate:20|upper }
```



**Más info:** <https://www.smarty.net/docs/en/language.modifiers.tpl>

# Buscando errores

---



# Debug

---

## Debug

- Tiempos de compilación de la plantilla.
- Valores de variables.

## Uso:

```
$smarty->debugging = true;
```

`O`

`{debug}`



# Debug - Salida

Variable  
asignada  
con assign

Variables  
instanciadas  
en los foreach

## Smarty 3.1.27 Debug Console - "file:index.tpl"

### assigned template variables

<b>\$SCRIPT_NAME</b>	Value	
Origin: "Global"	"/web2tudai/Smarty/index.php"	
<b>\$animal</b>	Value	Attributes
Origin: "file:index.tpl"	Array (5) Nombre => "Ballena" Familia => "Cetaceo" Clase => "Mamifero" 0 => "Acuatico" 1 => "Azul"	Array (1) _loop => true
<b>\$animales</b>	Value	
Origin: "Smarty object"	Array (2) 0 => Array (5) Nombre => "Leon" Familia => "Felino" Clase => "Mamifero" 0 => "Terrestre" 1 => "Amarillo" 1 => Array (5) Nombre => "Ballena" Familia => "Cetaceo" Clase => "Mamifero" 0 => "Acuatico" 1 => "Azul"	
<b>\$propiedad</b>	Value	Attributes
Origin: "file:index.tpl"	"Azul"	Array (1) _loop => true
<b>\$smarty</b>	Value	
Origin: "file:index.tpl"	null	

### assigned config file variables

# Caching

---

- Acelerar la carga.
- Al llamar a *display()* se guardará una copia de la salida a un archivo.
- El siguiente llamado usará la copia guardada en vez de calcular la página de nuevo.
- Al tomar una pagina ya hecha, los datos no serán actualizados.
- Uso:

```
$smarty->caching = true;
```

```
$smarty->cache_lifetime = 120; // en segundos
```

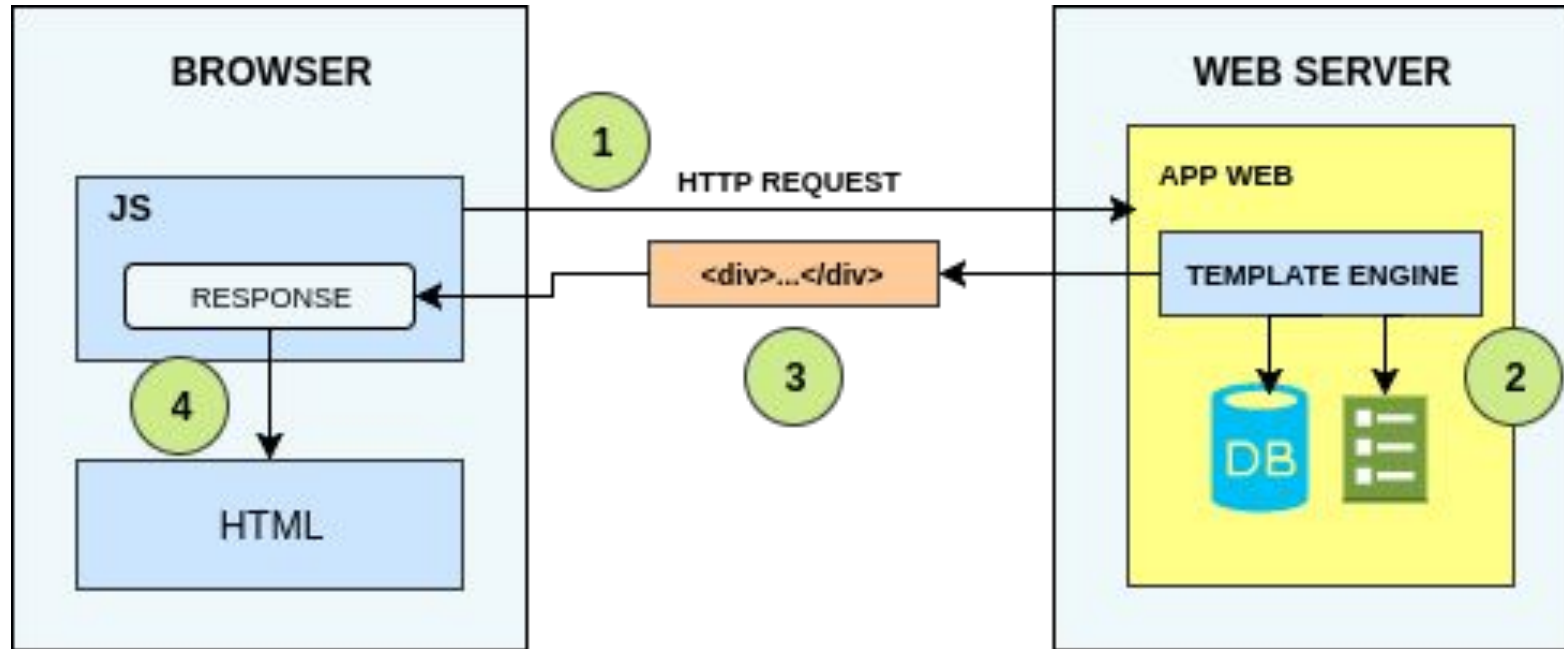
**NO RECOMENDABLE EN DESARROLLO!!!**

# Errores en una Arquitectura Cliente-Servidor

---

# AJAX y PHP

## Ejemplo con Partial Render



1. El cliente hace un llamado al servidor usando AJAX.
2. El servidor procesa el llamado, busca la información y genera el contenido usando Smarty.
3. El servidor devuelve **un fragmento** de HTML.
4. El cliente recibe el fragmento HTML y lo inserta en la página.

Si tenemos un error  
**¿Dónde lo buscamos?**

**PHP**  
**HTML**

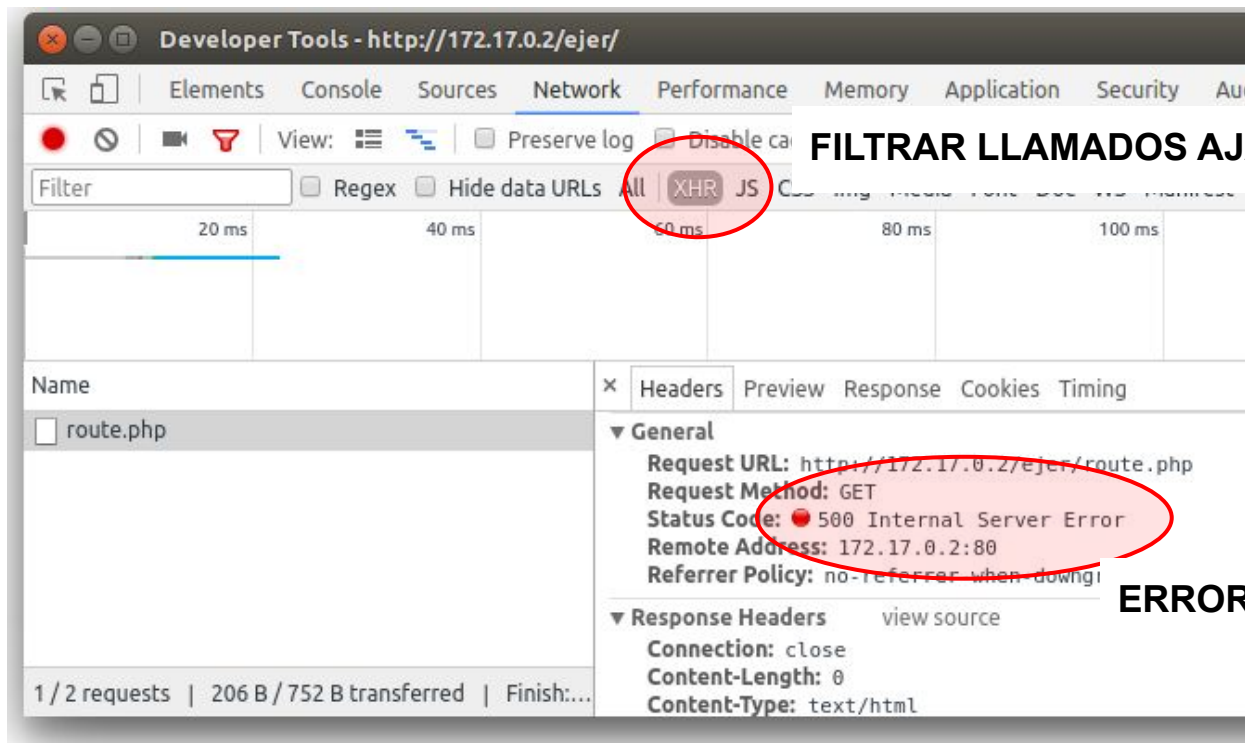


**JS**  
**SMARTY**

# AJAX y PHP

Existen dos posibilidades:

- o está del lado del servidor (PHP + Smarty)
- o está del lado del cliente (JS + HTML)



**FILTRAR LLAMADOS AJAX**

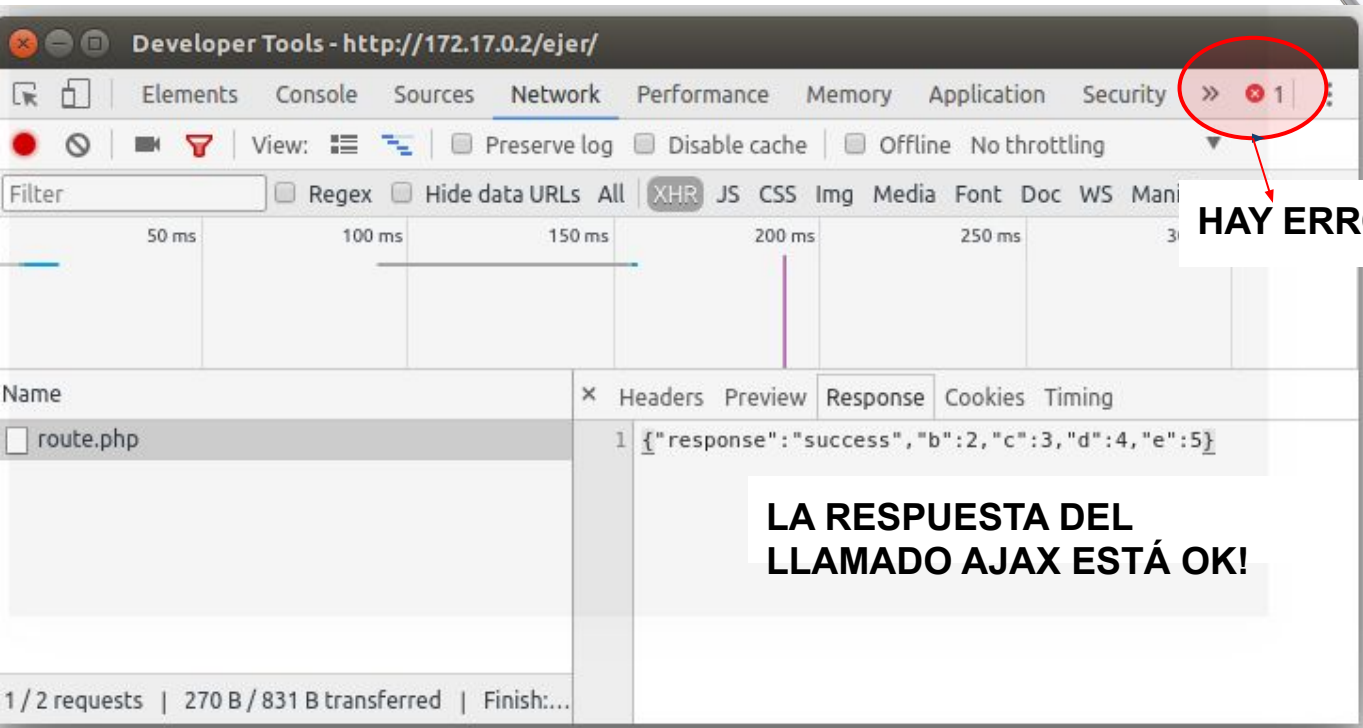


**ERROR EN EL SERVIDOR!!!**

# AJAX y PHP

Existen dos posibilidades:

- o está del lado del servidor (PHP + Smarty)
- o está del lado del cliente (**JS + HTML**)



HAY ERRORES JS!!!

LA RESPUESTA DEL  
LLAMADO AJAX ESTÁ OK!





# AJAX y PHP

---

A veces el código compila, pero no anda. La estrategia siempre es aislar el error, ver de cuál lado está.

# Alternativas

---

Existen muchos motores de templates que podemos usar:

## **BLADE**

<https://github.com/jenssegers/blade>

## **TWIG**

<https://twig.symfony.com/>

## **MOUSTACHE**

<https://github.com/bobthecow/mustache.php>

# Bibliografía

---

Smarty - <http://www.smarty.net>

Template Processor -

[https://en.wikipedia.org/wiki/Template\\_processor](https://en.wikipedia.org/wiki/Template_processor)

Web Template System -

[https://en.wikipedia.org/wiki/Web\\_template\\_system](https://en.wikipedia.org/wiki/Web_template_system)