



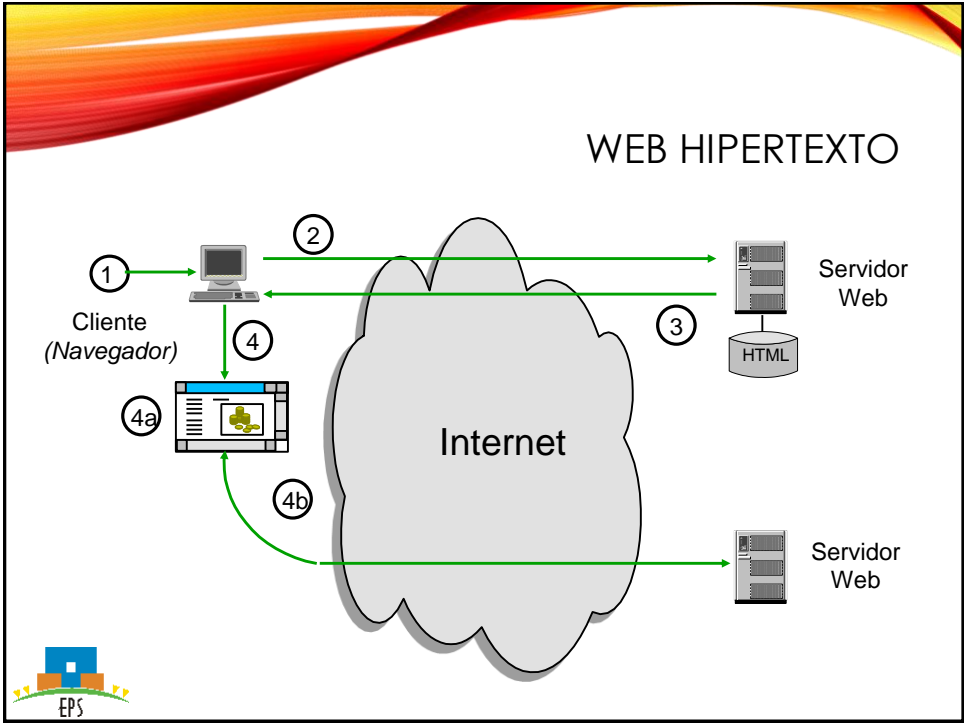
SISTEMAS DISTRIBUIDOS BASADOS EN LA WORLD WIDE WEB

Sistemas informáticos I



2.3 WEB INTERACTIVA. LAS APLICACIONES WEB

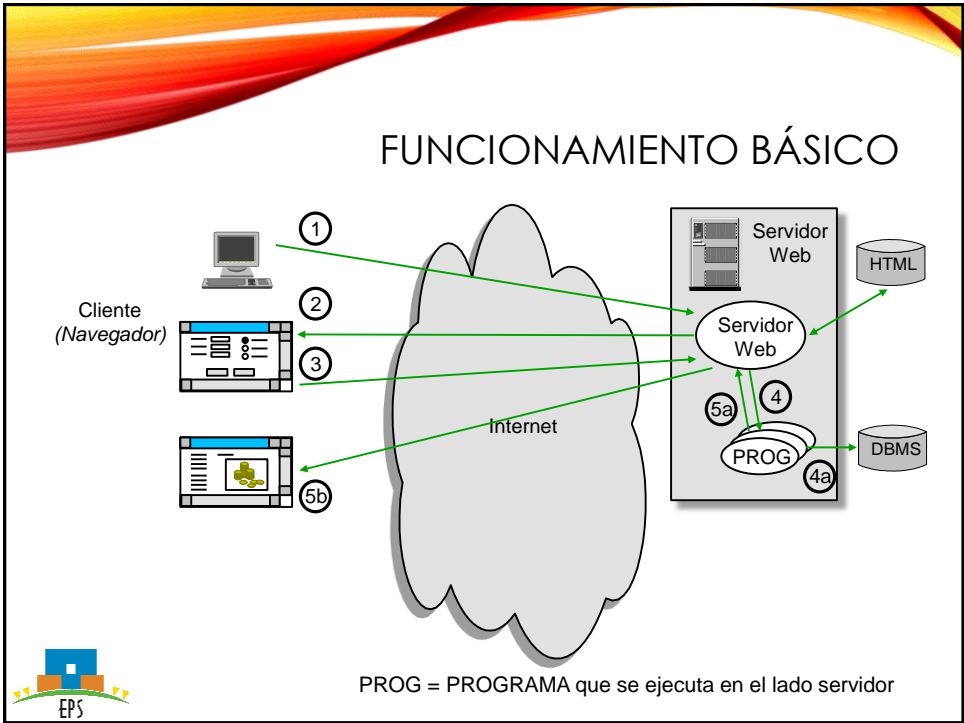
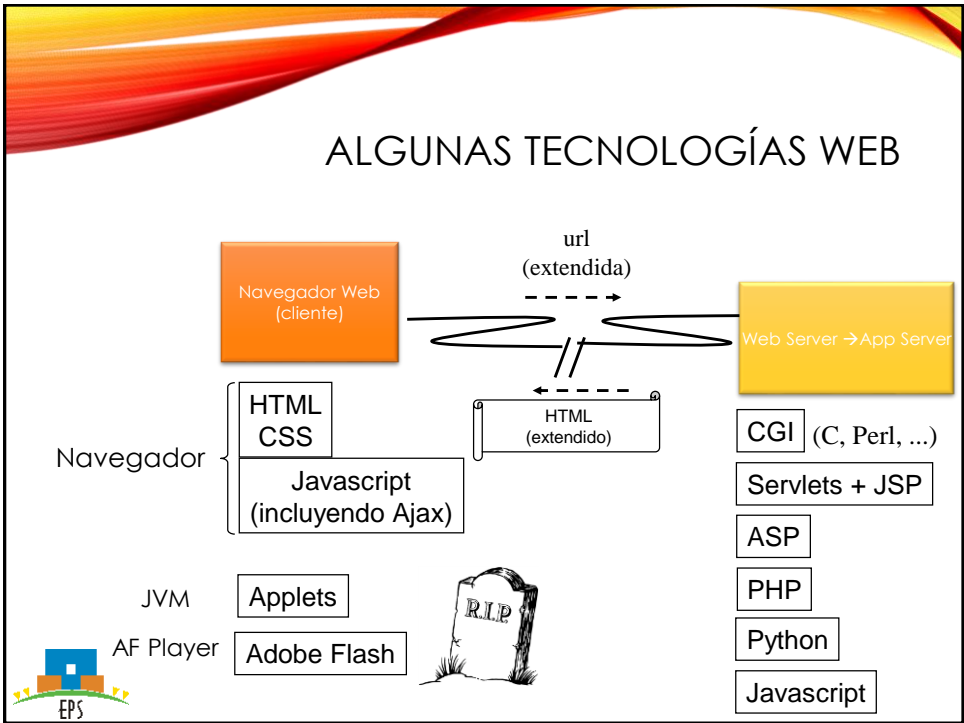
Sistemas distribuidos basados en la World Wide Web



WEB INTERACTIVA

- El modelo Web hipertexto no permite más interacción del usuario que seguir hiperenlaces para obtener **contenido estático** en una lectura no secuencial de documentos
- Necesario establecer comunicación entre programas que se ejecutan en el "servidor" con los datos que proporciona el usuario (del lado del cliente)

EPS



FUNCIONAMIENTO BÁSICO


1. El cliente solicita una página al servidor web
2. El servidor web envía una página que contiene un formulario
3. El cliente envía el formulario al servidor con los datos "suministrados" por el usuario de la aplicación
4. El servidor web:
 - a. Recibe y procesa la petición HTTP/S
 - b. Identifica el recurso o servicio que debe procesar el formulario por su URL
 - c. Extrae los parámetros de la llamada
 - d. Invoca a la funcionalidad correspondiente pasándole los datos recibidos del cliente
5. El programa genera una página con los resultados de su ejecución y la devuelve al cliente a través del servidor web



FORMULARIOS HTML

- Definidos mediante la etiqueta <FORM>
- Contenedor que agrupa los elementos de entrada de datos, mezclados con otros elementos estándar HTML
- Estructura:

```
<FORM ACTION=url METHOD=método de envío ...>  
  <INPUT> | <TEXTAREA> | <SELECT> | <BUTTON>  
  otros elementos HTML  
</FORM>
```
- *action*: URL del recurso que se debe solicitar al enviar el formulario
- *method*: Método de envío de los datos asociados al formulario




https://www.w3schools.com/html/html_forms.asp

"Programa" que atiende la petición en lado servidor

GET o POST

CAMPOS DE FORMULARIO

- Elementos HTML de interacción con el usuario de la aplicación:
 - **INPUT**: text, password, checkbox, radio, hidden, submit, reset
 - En HTML 5 muchos más con sus pertinentes validaciones (p.ej., number o email)
 - **SELECT**: Listas de selección con múltiples valores
 - **TEXTAREA**: Campo de texto multilínea
 - **BUTTON**: Botones de acción
- A cada campo de formulario se le asocia un nombre (atributo `name`) y un valor (atributo `value`)
- La interacción con el usuario puede modificar el valor
- Los resultados se envían al servidor en la petición HTTP mediante pares `<nombre>=<valor>`
 - Según el método elegido para el envío del formulario viajan asociados a la URL de procesamiento o en el cuerpo del mensaje HTTP



FORMULARIOS - EJEMPLO


```
<FORM ACTION="http://host/cgi-bin/login" METHOD="GET">
  <p>
    Usuario:
    <INPUT NAME="usuario" TYPE="text" SIZE="10" MAXLENGTH="8">
  </p>
  <p>
    Contraseña:
    <INPUT NAME="clave" TYPE="password" SIZE="10" MAXLENGTH="8">
  </p>
  <p>
    <INPUT NAME="nuevo" TYPE="radio" VALUE="si" CHECKED>
    Nuevo usuario
  </p>
  <p>
    <INPUT NAME="nuevo" TYPE="radio" VALUE="no">
    Usuario registrado
  </p>
  <center><p>
    <INPUT TYPE="RESET" VALUE="Borrar">
    <INPUT TYPE="SUBMIT" VALUE="Enviar">
  </p></center>
</FORM>
```

Usuario:

Contraseña:

☒ Nuevo usuario

☐ Usuario registrado



DIFERENCIA ENTRE GET Y POST

- En caso de GET:

```
GET /logon?usuario=Superman&clave=loislane&nuevo=si HTTP/1.1
(otras cabeceras HTTP)
(línea en blanco)
```

- En caso de POST:

```
POST /logon HTTP/1.1
CONTENT-TYPE: application/x-www-form-urlencoded
CONTENT-LENGT: xxx
(otras cabeceras HTTP)
(línea en blanco)
usuario=Superman&clave=loislane&nuevo=si
```



PROCESAMIENTO DE FORMULARIOS EN EL SERVIDOR

- La forma habitual de trabajar con formularios es usar un único "programa" que, dependiendo del método de envío, valide y procese el formulario o simplemente lo "muestre"
- Ventajas:
 - Simplifica el código y su mantenimiento
 - Permite validar los datos del formulario en el propio formulario
- Procedimiento:

```
si petición POST (se ha enviado el formulario):
  si hay errores:
    Mostrar formulario con errores
  si no:
    Procesar formulario
si no (petición GET, no se ha enviado el formulario):
  Mostrar formulario
fsi
```



CGI: COMMON GATEWAY INTERFACE

- CGI define un método “estándar” para que un servidor WWW pueda ejecutar programas externos y recoger información de ellos
- Es una extensión del protocolo HTTP
- El programa externo recibe del servidor información:
 - Asociada a la transmisión: Origen, URL, protocolo utilizado...
 - Introducida por el usuario, en un formulario de entrada.
- El paso de información se realiza mediante:
 - Variables de entorno
 - Línea de comandos
 - Entrada/salida estándar



VARIABLES DE ENTORNO

- | | |
|--|---|
| <ul style="list-style-type: none">• No dependientes de la petición:<ul style="list-style-type: none">• SERVER_SOFTWARE• SERVER_NAME• GATEWAY_INTERFACE• Dependientes de la petición:<ul style="list-style-type: none">• SERVER_PROTOCOL• SERVER_PORT• REQUEST_METHOD• PATH_INFO• PATH_TRANSLATED• SCRIPT_NAME• QUERY_STRING• REMOTE_HOST• REMOTE_ADDR | <ul style="list-style-type: none">• Asociadas a la seguridad de acceso:<ul style="list-style-type: none">• AUTH_TYPE• REMOTE_USER• REMOTE_IDENT• Metainformación de la petición:<ul style="list-style-type: none">• CONTENT_TYPE• CONTENT_LENGTH• Variables de la cabecera HTTP:<ul style="list-style-type: none">• HTTP_ACCEPT• HTTP_ACCEPT_LANGUAGE• HTTP_USER_AGENT• HTTP_COOKIE |
|--|---|



VENTAJAS E INCONVENIENTES

- Ventajas:
 - Sencillez de programación
 - Uso de cualquier lenguaje de programación
 - El programa CGI no afecta al funcionamiento del servidor por ejecutarse como un proceso independiente
 - Estándar. Garantiza la portabilidad entre servidores de distintos fabricantes
- Inconvenientes:
 - Lentitud
 - Cada ejecución requiere crear un proceso y finalizarlo, lo que implica reserva de memoria, apertura/cierre de ficheros, conexiones a bases de datos, etc.
 - El programa CGI termina con cada llamada → no es posible el mantenimiento automático de un estado de la comunicación entre peticiones
 - Gestión de **sesiones** en manos de los programas





SESIONES Y COOKIES

- Una sesión es la relación que se establece entre un cliente y un servidor durante un tiempo finito
- Asociado al concepto de sesión, se habla de la sesión como un contexto persistente en el que almacenar/recuperar datos mientras la relación entre cliente y servidor se mantenga activa
 - El típico ejemplo es un "carrito de la compra"
 - El servidor debe mantener información asociada al usuario de la aplicación a lo largo de todo el proceso
- ... pero HTTP es un protocolo "sin memoria" → cookies HTTP
 - Pequeño fragmento de información que el servidor (con el permiso del navegador) almacena en el cliente
 - Cada vez que el navegador solicite una nueva página al servidor envía también la cookie
 - Aunque en origen se crearon para la comunicación entre el cliente y el servidor, en la actualidad, la funcionalidad en el cliente también puede escribir y leer en las cookies
 - Sobre el papel un gran invento. En la práctica, presentan serios problemas de seguridad



MANTENIMIENTO DE SESIÓN EN CGI

- En los CGIs la gestión de sesiones está en manos de los programas
 - Mediante campos de formulario ocultos:
 - El usuario rellena un formulario con una serie de datos
 - El cliente envía al servidor el formulario
 - El programa CGI que lo recibe genera una página HTML en la que añade los valores recibidos como campos ocultos
 - El usuario, al enviar nuevos formularios, proporciona al programa CGI los datos nuevos y los de la petición anterior



WEB API: WEB APPLICATION PROGRAMMING INTERFACES

- Surgen para tratar de evitar los problemas de bajo rendimiento de la interfaz CGI:
 - Los nuevos programas se enlazan junto con el servidor en una librería dinámica
 - El servidor llama a las funciones de librería como tareas dentro del propio proceso servidor
 - El proceso servidor no finaliza: Se mantienen ficheros abiertos, conexiones a bases de datos, etc. entre llamadas a funciones
 - Se proporciona una API de acceso a los datos y estado del servidor
- Inconvenientes:
 - Un fallo en una rutina puede hacer caer el servidor completo
 - Lenguajes de programación normalmente limitados a C y C++ (en la actualidad más variedad)
 - Difícil de programar. Es preciso conocer el funcionamiento interno del servidor para aprovecharla a máximo
- Proporcionadas por casi todos los fabricantes: Netscape (NSAPI), Microsoft (ISAPI), IBM (ICAPI, GWAPI)...



INTERFACES HÍBRIDAS

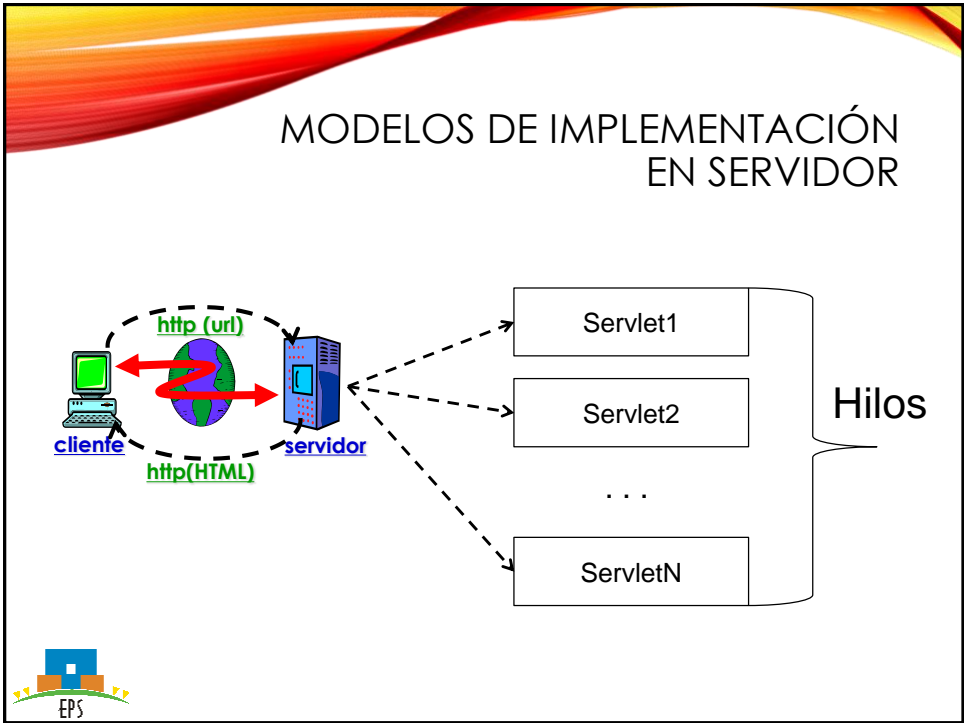
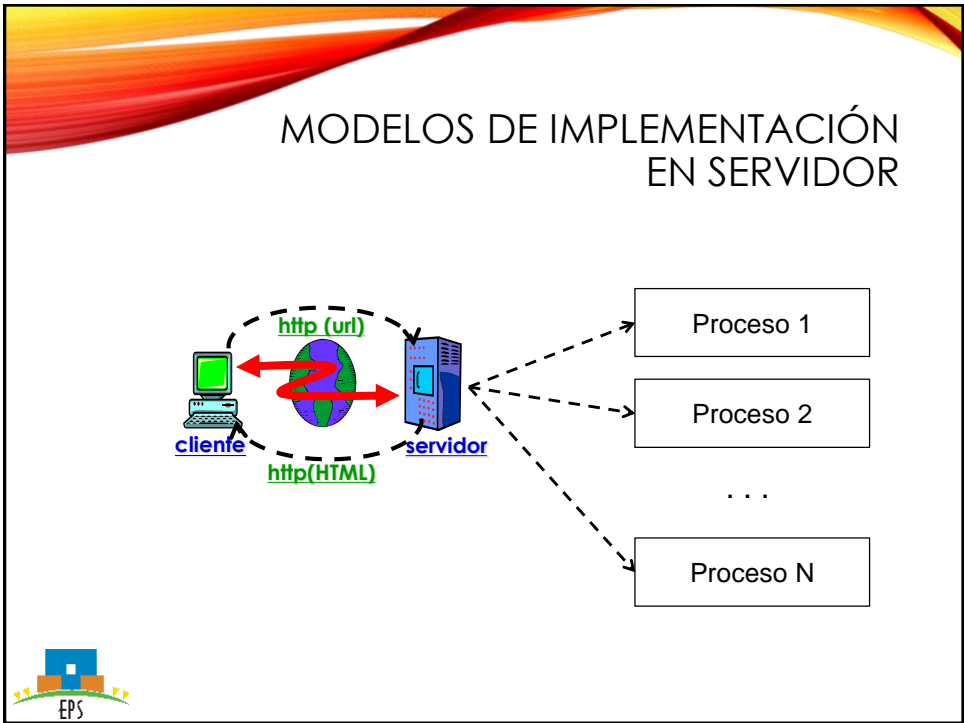
- Intentan conseguir las ventajas de CGI y Web APIs evitando sus inconvenientes
- Los programas se desarrollan de modo independiente al servidor Web, y en cualquier lenguaje
- El servidor Web, durante su inicialización, puede arrancar los programas en procesos independientes
- Los programas arrancados, tras inicializarse, quedan a la espera de recibir peticiones
 - Todo el proceso de inicialización se realiza antes de recibir una petición
- La comunicación mediante variables de entorno y *stdin/stdout* se sustituyen por un mecanismo de comunicación entre procesos más rápido
 - Puede permitir acceso remoto empleando un mecanismo de comunicación apropiado
 - Empleando los mismos elementos que en CGI se consigue facilidad de migración de programas CGI a las nuevas interfaces
- Tras atender una petición, el programa no finaliza: vuelve a esperar la siguiente petición
 - Es posible mantener el estado de la aplicación entre peticiones sucesivas.
- Siguen este modelo:
 - FastCGI, de Open Market, Inc. Comunicación Servidor - Programas por Sockets
 - Netscape Web Application Interface (WAI). Comunicación mediante CORBA



PÁGINAS DINÁMICAS

- Como alternativa a CGI y Web APIs, surgen extensiones del lenguaje HTML que permiten una mayor capacidad de procesamiento
 - En el cliente (*client side scripting*)
 - Inclusión de código en el documento que el cliente interpretará para variar dinámicamente la presentación de la página
 - Proporciona "inteligencia" en el navegador
 - En el servidor (*server side scripting*):
 - Inclusión de código funcional en el fichero HTML que contiene la descripción de la página
 - El servidor lo interpretará para generar dinámicamente la página antes de su envío al cliente





MODELOS DE IMPLEMENTACIÓN EN SERVIDOR

- En PHP, por ejemplo, depende del servidor web asociado
- Con Apache hay dos modelos posibles:
 - Multi-proceso: también llamado pre-forked, porque al iniciarse el servidor crea un pool de procesos que son reusados para atender las distintas peticiones

StartServers	5
MinSpareServers	5
MaxSpareServers	10
MaxClients	150
MaxRequestsPerChild	0


Máximo número de procesos a crear

- Multi-hilo (workers) : aunque el motor PHP está preparado para trabajar con múltiples hilos, hay muchas bibliotecas adicionales que no lo están

StartServers	2
MaxClients	150
MinSpareThreads	25
MaxSpareThreads	75
ThreadsPerChild	25
MaxRequestsPerChild	0

Procesos

Hilos x proceso





MODELOS DE IMPLEMENTACIÓN EN SERVIDOR

- Modelo alternativo: asíncrono
- Es el modelo implementado, por ejemplo, por **NodeJS**:
 - Sistema para poder programa los servidores web en *JavaScript*
 - Un hilo (y proceso, por supuesto)
 - Llamadas no bloqueantes y programación basada en eventos
 - Por ejemplo, si quiero leer un fichero (para responder a petición http), invoco a la función de lectura, dándole el nombre de otra función (*callback*) y sigo ejecutando (posiblemente atendiendo otras peticiones)
 - Cuando se termina de leer el fichero (se produce el evento), el sistema invoca automática la función *callback*



PHP: HYPERTEXT PREPROCESSOR

- Lenguaje de programación de propósito general (interpretado)
- Sintaxis similar C/C++
- Originalmente diseñado para el preprocesamiento de texto plano
- En el contexto web utilizado para generar contenido dinámico del lado del servidor. Otros lenguajes similares son ASP o JSP
 - Los scripts PHP están incrustados en los documentos HTML y el servidor los interpreta antes de servir las páginas al cliente
 - El cliente no ve el código PHP, sino los resultados que produce
 - Software abierto y gratuito
 - Integra acceso a formularios, archivos, bases de datos...
 - El acceso a la información propia del servidor se consigue a través de arrays indexados con el nombre del parámetro al que se desea acceder:
 - **\$_SERVER**: Variables propias del servidor
 - Ejemplos: `$_SERVER[QUERY_STRING]`, `$_SERVER[REMOTE_HOST]`
 - **\$_REQUEST**: Variables asociadas a la petición recibida (campos de formularios)
 - **\$_SESSION**: Información asociada a la sesión de trabajo con el usuario
 - **\$_COOKIE**: Información asociada a las cookies
 - ...



EJEMPLO PHP

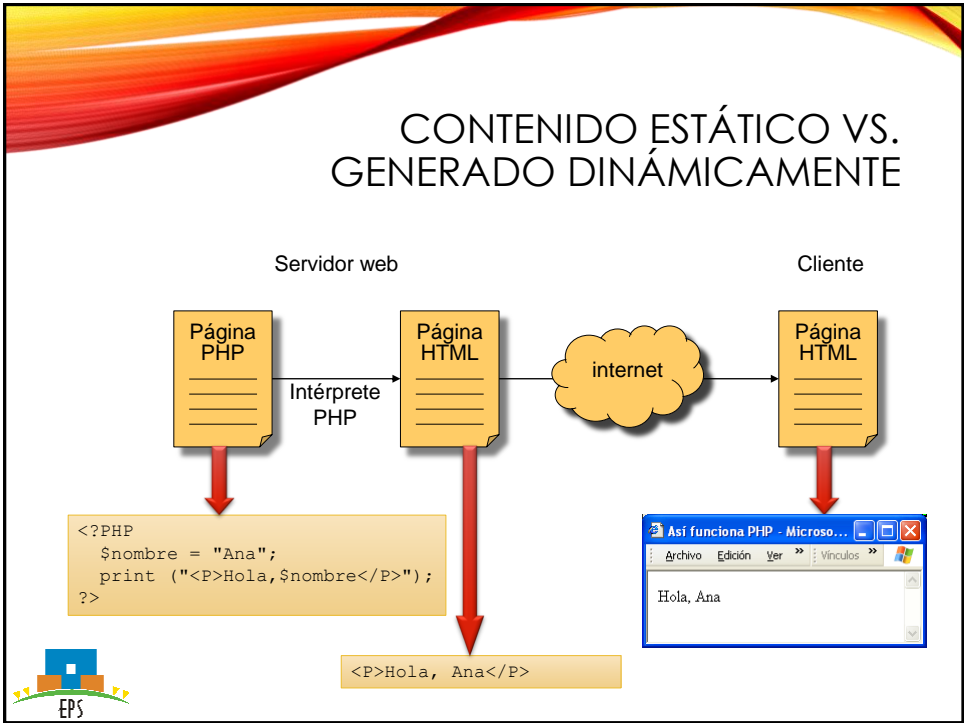
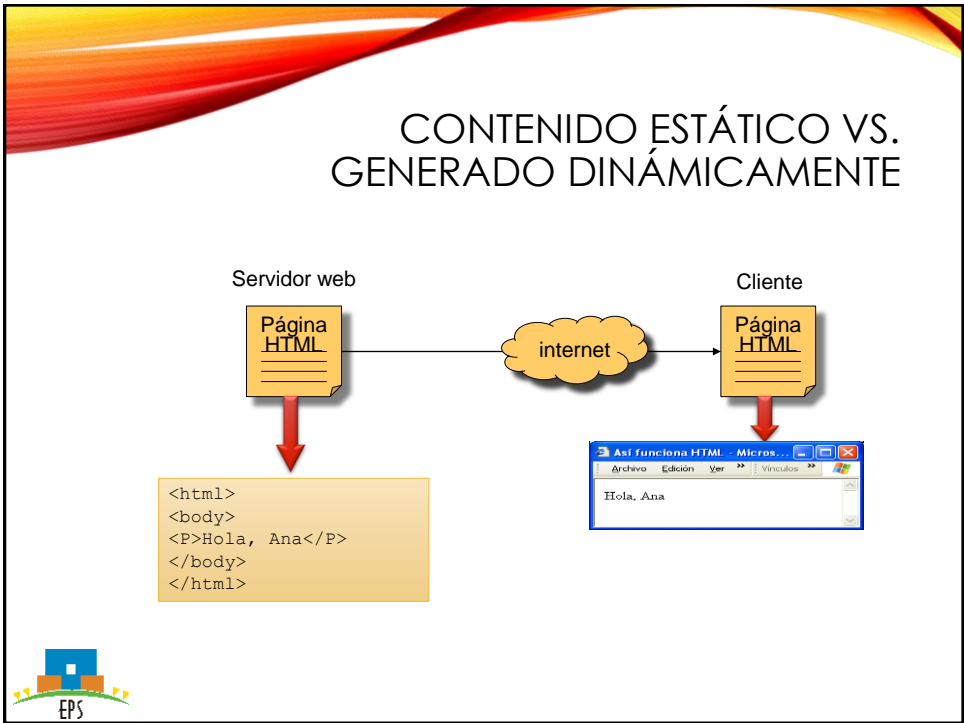
```
<html>
<body>

  <?php echo "Hello World"; ?>

</body>
</html>
```

- Vínculo URL – código
 - <http://www.misitioweb.com/index.php>
 - Busca un fichero llamado 'index.php' y lo interpreta

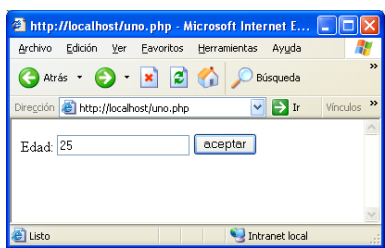




RECEPCIÓN DE PARÁMETROS. LA FUNCIÓN \$_REQUEST

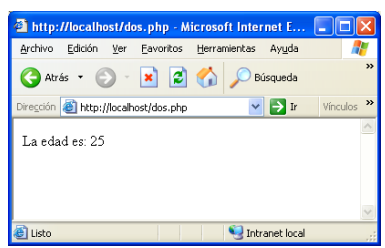
uno.html

```
<HTML>
<BODY>
<FORM ACTION="dos.php" METHOD="POST">
  Edad: <INPUT TYPE="text" NAME="edad">
  <INPUT TYPE="submit" VALUE="aceptar">
</FORM>
</BODY>
</HTML>
```



dos.php

```
<HTML>
<BODY>
<?PHP
  $edad = $_REQUEST['edad'];
  print ("La edad es: $edad");
?>
</BODY>
</HTML>
```



RECEPCIÓN DE PARÁMETROS. LA FUNCIÓN \$_GET

```
<form action="welcome.php" method="get">
  Name: <input type="text" name="fname" />
  Age: <input type="text" name="age" />
  <input type="submit" value="Enviar consulta" />
</form>
```

welcome.php?fname=Usuario&age=29

Name: Age:

```
Welcome <?php echo $_GET["fname"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```



Welcome Usuario!
You are 29 years old.


RECEPCIÓN DE PARÁMETROS. LA FUNCIÓN \$_POST

```
<form action="welcome.php" method="post">  
  Name: <input type="text" name="fname" />  
  Age: <input type="text" name="age" />  
  <input type="submit" />  
</form>
```

welcome.php

Name: Age:

```
Welcome <?php echo $_POST["fname"]; ?>.<br />  
You are <?php echo $_POST["age"]; ?> years old!
```



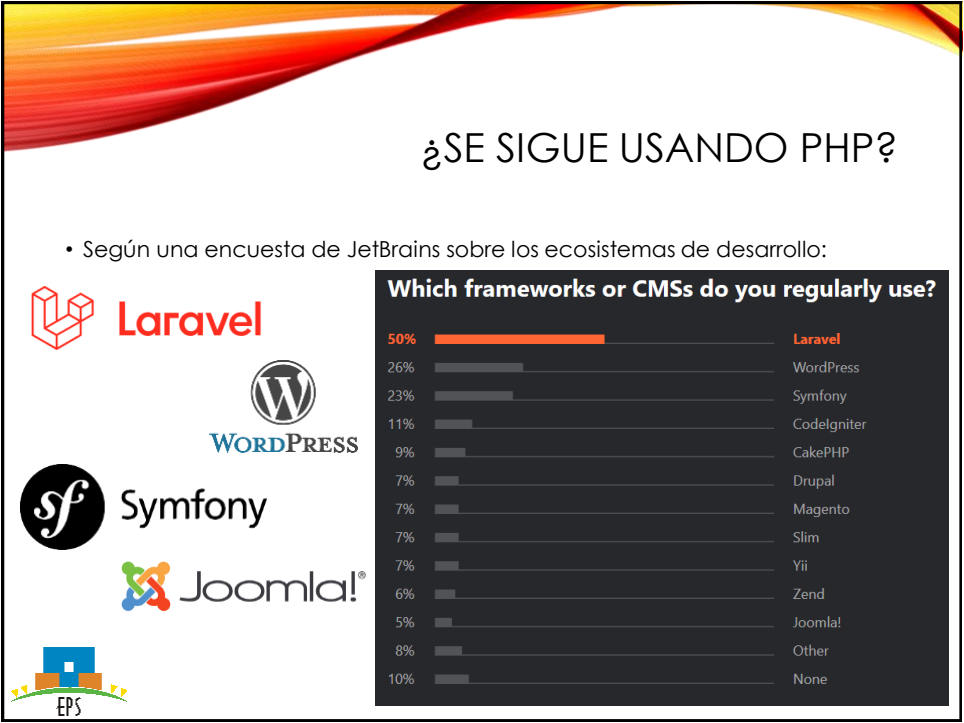
Welcome Usuario!
You are 29 years old.

¿SE SIGUE USANDO PHP?

- El titular del informe anual TIOBE de septiembre de 2019 es: "PHP is struggling to keep its top 10 position"
- A pesar de ello, es un lenguaje que aún está vivo en el mercado

Sep 2019	Sep 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	▲	C#	3.399%	+0.10%
6	5	▼	Visual Basic .NET	3.291%	-2.02%
7	8	▲	JavaScript	2.128%	-0.00%
8	9	▲	SQL	1.944%	-0.12%
9	7	▼	PHP	1.863%	-0.91%
10	10		Objective-C	1.840%	+0.33%
11	34	▲	Groovy	1.502%	+1.20%





PYTHON PARA LA WEB

- Lenguaje de programación de propósito general (interpretado)
- Orientado a objetos
- En el contexto web utilizado para generar contenido dinámico del lado del servidor
- Las aplicaciones siempre se podrían programar desde cero, pero lo habitual es utilizar bibliotecas o frameworks
- El vínculo URL – código no es directo, depende de la biblioteca/framework que usemos
 - Lo habitual es realizar mapeos de nivel superior
 - Aquí vamos a ver *Flask* y *Django*



FLASK

- Micro-framework (en contraste con frameworks más completos como Django)
 - Núcleo simple pero extensible
 - No toma decisiones por el programador (como por ejemplo, tipo de base de datos, e incluso si usa BdD o no!)
- Provee librerías (extensiones) para, por ejemplo:
 - Integración de BdD
 - Validación de formularios
 - Gestión de ficheros (uploads)
 - Autenticación
- ... pero puedo ignorar todo esto e implementarlo por mi mismo



Flask



FLASK: USO

```
from flask import Flask
app = Flask(__name__)
```

Nombre del módulo
‘__main__’

Interfaz con el servidor web

Por ejemplo: en fichero ‘hello.py’

pip install flask



FLASK: LANZAR SERVIDOR HTTP

- Flask incorpora un servidor web propio, sólo apto para desarrollo
- Dos métodos para lanzar el servidor web
 - Antes de versión 1 de Flask, en el propio script y ejecutándolo desde consola

```
if __name__ == '__main__':
    app.run(debug=True)
```

\$ python hello.py
 - A partir de la versión 1 de Flask, dos opciones desde consola, pero sin incluir nada en el script:

```
$ export FLASK_APP=hello.py
$ flask run
* Running on http://127.0.0.1:5000/
```

\$ export FLASK_APP=hello.py
\$ python -m flask run
* Running on http://127.0.0.1:5000/



19

ALGUNAS OPCIONES

- Para que sea visible desde el exterior

```
$ flask run --host=0.0.0.0
```

- Modo Debug

```
$ export FLASK_ENV=development  
$ flask run
```



MAPEO/ENRUTAMIENTO DE URLS

- Establece las relaciones entre URLs y código Python (funciones)

```
@app.route('/')  
def index():  
    return '<h1>Hello World!</h1>'
```



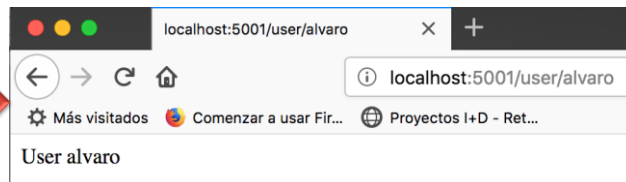
VISTAS

- Las funciones como `index()` se convierten en manipuladores (*handlers*) de la URL asociada
- En *Flask* (y otros frameworks) las funciones creadas para ser *handlers* se denominan vistas (*views*)
- Si basamos nuestra aplicación en una arquitectura MVC, los *handlers* suelen hacer las veces de controlador (cuidado con la posible confusión del nombre *view*)



MAPEO/ENRUTAMIENTO DINÁMICO

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username
```




MAPEO/ENRUTAMIENTO DINÁMICO

Converter types:

string	(default) accepts any text without a slash
int	accepts positive integers
float	accepts positive floating point values
path	like string but also accepts slashes
uuid	accepts UUID strings

```
@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # show the subpath after /path/
    return 'Subpath %s' % subpath
```




MAPEO/ENRUTAMIENTO

- El mapeo puede ser dependiente del método utilizado para la petición HTTP

```
from flask import request

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```



SE PUEDE COMPLICAR RÁPIDAMENTE

```
@app.route('/ex2')
def function_for_ex2():
    """
    It process the '/' url.
    :return: basic HTML for /ex2
    """
    return '<!DOCTYPE html>' \
        '<html lang="es">' \
        '<head>' \
        '<title> This is the page title </title>' \
        '</head>' \
        '<body> <div id ="container">' \
        '<h1>Example of HTML Content</h1>' \
        'Return to the homepage by clicking <a href="/"> here</a> </body>' \
        '</html>'
```



SOLUCIÓN: SEPARACIÓN DE RESPONSABILIDADES

```
@app.route('/ex2')
def function_for_ex2():
    return app.send_static_file("ejemplo.html")
```

Todo parece llevar a una arquitectura MVC

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title> This is the page title </title>
</head>
<body> <div id ="container">
  <h1>Example of HTML Content</h1>
  Return to the homepage by clicking <a href="/"> here</a>
</body>
</html>
```

static/ejemplo.html



¿PERO ENTONCES HTML ESTÁTICO O "LA MUERTE"?

- Entran los templates
- Por defecto, Flask utiliza el motor (intérprete) de templates Jinja2
 - <http://jinja.pocoo.org/>

```
from flask import render_template
```

```
@app.route('/hello/') @app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

```
/application.py
/templates
/hello.html
```

```
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello, World!</h1>
{% endif %}
```



COOKIES EN FLASK

```
from flask import make_response
```

```
@app.route("/setcookie/<user>")
def setcookie(user):
    msg = "user cookie set to: " + user
    response = make_response(render_template('mensaje.html', mensaje=msg))
    response.set_cookie('helloflask_user',user)
    return response
```

```
@app.route("/getcookie")
def getcookie():
    user_id = request.cookies.get('helloflask_user')
    if user_id:
        msg = "user is: " + user_id
    else:
        msg = "no user cookie"
    return render_template('mensaje.html', mensaje=msg)
```



SESIONES EN FLASK

- Aunque Flask no obliga a usar un sistema de sesiones en particular, viene con uno predefinido
- Construido sobre cookies firmadas criptográficamente
 - Ayuda a evitar Session hijacking y otros ataques

```
from flask import session

@app.route("/setsession/<data>")
def setsession(data):
    msg = "session data set to: " + data
    session['data'] = data
    session.modified = True
    return render_template('mensaje.html', mensaje=msg)

@app.route("/getsession")
def getsession():
    if 'data' in session:
        msg = "session data: " + session['data']
    else:
        msg = "no session data"
    return render_template('mensaje.html', mensaje=msg)
```



DJANGO

- Framework de alto nivel para el desarrollo de aplicaciones web basadas en Python
- De mayor calado que Flask, sin llegar a ser un macro-framework
- Gestiona e integra prácticamente todos los aspectos relevantes del desarrollo de una aplicación web desde una perspectiva de proyecto cerrado
- La poca flexibilidad que permite en el diseño de la arquitectura software se consigue vía customización
- Paquetes necesarios para usar Django
 - Framework (*django*)
 - Herramientas para la gestión de proyectos Django (*django-admin*)



DJANGO: LANZAR SERVIDOR HTTP

- Como todo, a través del script de administración

```
$ python manage.py runserver
```

- Dentro de la estructura de archivos que define el servidor con sus aplicaciones
- Lee los parámetros de configuración del fichero *settings.py*



SESIONES

- Gestionadas por un middleware que se debe activar en el fichero de configuración
 - Permite abstraer al desarrollador de todo el problema del envío, recepción y gestión de datos
- Por defecto, se basan en el almacenamiento de datos en el servidor y el envío de una cookie con un id de sesión
 - También es posible un mecanismo basado en el almacenamiento de datos en cookies (menos recomendable)
- Los datos de sesión por defecto se almacenan en la base de datos en unas tablas que se crean vía una migración
 - También es posible configurarlo para que se almacenen en disco o en memoria



SESIONES

- Cada vez que se recibe una nueva petición HTTP en el servidor, el middleware obtiene los datos de sesión del sistema de almacenamiento y los guarda en la *Request* en un objeto similar a un diccionario

```
from django.http import HttpRequest

request.session[nombre_variable]           // Lectura
request.session[nombre_variable] = valor   // Asignación
```

- Cuando se termina el procesamiento de una petición, el middleware persiste los datos almacenados en la *Request*
 - Si han cambiado
 - Se puede forzar la actualización:

```
request.session.modified = True
```

- Por defecto, los datos se serializan en JSON

