Introducción:

Funcionamiento de un Web Site:

El funcionamiento de un Web-Site es un ejemplo típico de la arquitectura cliente-servidor, en donde múltiples clientes se conectan a un servidor (en algunos casos varios) en forma simultanea. En general el servidor depende de la instalación del site mientras que el cliente suele ser un browser, en general Netscape Navigator o Microsoft Explorer. Como en todo esquema cliente-servidor debe existir un protocolo que especifique de que forma se comunican e intercambian datos el cliente y el servidor, el protocolo utilizado en un web site es el protocolo HTTP que funciona "encapsulado" sobre el protocolo TCP/IP.

Introducción al Protocolo HTTP:

Básicamente el protocolo es iniciado por el cliente con un "request", es decir un pedido de un recurso determinado, que es casi siempre contestado por el server con el envío de una respuesta ("response") que incluye un código indicando si el pedido pudo ser resuelto por el server o no.

Un request genérico tiene la forma:

METODO URI PROTOCOLO CrLf
HEADERS*
CrLf
Datos

El MÉTODO en general puede ser GET o POST
URI es el identificador del recurso que se desea pedir, el formato es: http://host:port/path?query_string
PROTOCOLO debe ser HTTP / 1.1
CrLf es un Carriage Return seguido de un New Line (0x13,0x10)
Headers son de tipo: Header-Name: Value CrLf, y pueden indicar varias cosas.

Un ejemplo de pedido es:
GET http://www.yahoo.com HTTP/1.1

El server responde con una RESPUESTA de la forma:

PROTOCOLO STATUS VALOR CrLF
Headers*
Content-Type: TIPO CrLf
CrLf
Datos

Un ejemplo de respuesta de un server podría ser:

HTTP/1.1 200 OK
Date: Mon, 12 Jun 2000 14:04:28 GMT
Server: Apache/1.3.9 (Unix) ApacheJServ/1.1
Connection: close
Content-Type: text/html

Datos…..

Los datos que el server envía al browser dependen del “Content-Type” declarado, básicamente los tipos más usados son texto plano (text/plain), código html (text/html), o imágenes (image/gif u otros).

De esta forma el cliente y el server se comunican por medio de tantos ciclos REQUEST-RESPONSE como sean necesarios, es de destacar que por cada REQUEST se debe iniciar una conexión nueva entre el cliente y el servidor ya que el protocolo HTTP en su forma básica no admite que en una misma conexión se haga más de un pedido al server. En una página html simple con 3 imágenes por ejemplo es normal que se efectúen 4 conexiones al server: una para la página y luego una por cada imagen.

Ejemplo:

Supongamos que tenemos la siguiente página html en un servidor, supongamos que la dirección del servidor es www.prueba.com y que la página se llama index.html, el ciclo que se da entre el browser y el server es de La forma:

```
<HTML>
<HEAD>
<TITLE>Ejemplo</TITLE>
</HEAD>
<BODY>
Hola esta es una prueba
<IMG SRC=”prueba.gif”>
</BODY>
</HTML>
```

Comunicación Browser-Server

BROWSER	**SERVER**
GET http://www.prueba.com/index.html HTTP / 1.1	HTTP/1.1 200 OK Date: Tue, 13 Jun 2000 14:15:45 GMT Server: Apache/1.3.9 (Unix) PHP/4.0.0 Last-Modified: Tue, 13 Jun 2000 14:09:05 GMT ETag: "5804d-73-39464081" Accept-Ranges: bytes Content-Length: 115 Connection: close Content-Type: text/html <HTML> <HEAD> <TITLE>Ejemplo</TITLE> </HEAD> <BODY> Hola esta es una prueba <IMG SRC="prueba.gif"> </BODY> </HTML>
GET http://www.prueba.com/prueba.gif HTTP / 1.1	HTTP/1.1 200 OK Date: Tue, 13 Jun 2000 14:18:22 GMT Server: Apache/1.3.9 (Unix) PHP/4.0.0 Last-Modified: Tue, 13 Jun 2000 14:07:36 GMT ETag: "5804e-2b2-39464028" Accept-Ranges: bytes Content-Length: 690 Connection: close Content-Type: image/gif GIF89aGÕÿ11B99ZRJkcR-œk½Î ½„Æµ{µ„{ZÖÆŒï ÞœçÖ”-{sRskJ„{RkcBœ”k¥{sJµ (CORTADO)

Luego el browser es responsable de interpretar y mostrar en la pantalla el código html y la imagen que recibió del servidor.

Tecnologías disponibles para el desarrollo de aplicaciones:

Para desarrollar aplicaciones y dotar a las páginas web de funcionalidad se puede trabajar tanto en el lado del cliente como en el lado del servidor, las variantes son:

Programación en el cliente:

- El browser envía un request.
- El server envía un response que contiene código que el browser entiende.
- El browser interpreta el código enviado por el server y realiza una determinada acción.

Programación en el servidor:

- El browser envía un request.
- El server ejecuta una aplicación que realiza una determinada acción.
- El server envía el resultado de dicha aplicación al cliente.
- El browser muestra el resultado recibido del server.

Esquema mixto: (programación en el cliente y en el servidor)

- El browser envía un request.
- El server ejecuta una aplicación que realiza una determinada acción.
- El server envía el resultado de dicha aplicación al cliente conteniendo código a interpretar por el browser.
- El browser interpreta el código enviado por el server y realiza una determinada acción.

La programación del lado del cliente tiene como principal ventaja que la ejecución de la aplicación se delega al cliente, con lo cual se evita recargar al servidor de trabajo. El servidor sólo envía el código, y es tarea del browser interpretarlo. La gran desventaja de esta metodología es que el código que el server envía es "sensible" a que cosas puede o no hacer el browser. El usuario puede, por ejemplo, decidir deshabilitar una funcionalidad del browser que es necesaria para que se ejecute un determinado servicio o peor aún, browsers distintos pueden interpretar el mismo código de distintas formas. Típicamente Netscape y Microsoft, que producen los dos browser más usados del mercado, no se ponen de acuerdo sobre como se implementan diversas tecnologías en el cliente.

Programar del lado del servidor tiene como gran ventaja que cualquier cosa puede hacerse sin tener en cuenta el tipo de cliente, ya que la aplicación se ejecuta en el servidor que es un ambiente controlado. Una vez ejecutada la aplicación, el resultado que se envía al cliente puede estar en un formato "normalizado" que cualquier cliente puede mostrar. La desventaja reside en que el server se sobrecarga de trabajo ya que además de servir páginas es responsable de ejecutar aplicaciones. A menudo esto redunda en requisitos de hardware mayores a medida que el server ejecuta más y más servicios.

Programación en el cliente	**Programación en el servidor**
HTML	CGI (Cualquier Lenguaje)
CSS	ASP
DHTML	PHP
JavaScript	mod_perl
Java	
VBScript	

Debido a las incompatibilidades existentes y a la posibilidad de que el usuario controle que cosas se ejecutan y cuales no la programación del lado del cliente no es muy recomendable y debe limitarse a código altamente standard que pueda interpretarse de cualquier forma en cualquier browser, lo cual obliga a ejecutar la gran mayoría de las aplicaciones y servicios de un web site del lado del servidor.

Server Side Programming.

Para el desarrollo de aplicaciones del lado del servidor existen 3 grandes metodologías, utilizar el protocolo CGI, utilizar una API provista por el web-server o bien utilizar un "módulo" del web server.

El protocolo CGI:

El protocolo CGI (Common Gateway Interface) fue creado para establecer un protocolo standard de comunicación entre el web-server y cualquier lenguaje de programación de forma tal que desde el lenguaje "x" puedan recibirse datos que el usuario envía usando el método "POST" o "GET" y además el resultado de la aplicación sea derivado por el web-server al browser. Típicamente para recibir datos se usa alguna biblioteca o módulo del lenguaje elegido que implementa el protocolo CGI y para enviar datos simplemente se envían al standard-output desde el lenguaje elegido y el web-server se encarga de redireccionar esto al browser.

Para ejecutar una aplicación CGI el web-server en general procede de la siguiente manera:

- Se toma el "request del browser" y los datos que se envían al server por método "GET" o "POST" se pasan a variables de ambiente.
- El server redirecciona su salida standard al browser.
- El server crea un proceso (Fork) (que tiene la salida standard redireccionada)
- El server ejecuta en el proceso creado la aplicación deseada.
- Se ejecuta la aplicación

Cuando la aplicación termina de ejecutarse el proceso muere. Dentro de la aplicación se usa algún mecanismo para recuperar los datos enviados por el browser desde las variables de ambiente (todos los lenguajes manipulan variables de ambiente). El protocolo CGI justamente consiste en especificar de que forma los datos enviados por el browser se convierten en variables de ambiente, esto en general es transparente al usuario.

De esta forma pueden realizarse aplicaciones para un web-site en casi cualquier lenguaje, los lenguajes interpretados rápidamente ganaron terreno ya que tienen un ciclo de desarrollo en tiempo inferior a los lenguajes compilados y son más fáciles de debuggear dentro del ambiente CGI.

Los lenguajes no interpretados (C, C++) tienen como ventaja que requieren menos recursos del server al generarse el proceso CGI (no hace falta un interprete) y además suelen ser mucho más veloces en su ejecución (no se necesita interpretar nada), sin embargo el desarrollar y debuggear suelen ser tareas muy complejas y no siempre se justifica el esfuerzo si la aplicación es pequeña. En los comienzos de la web la gran mayoría de las aplicaciones se encontraban en la categoría chica / muy chica por lo que la eficiencia no era un factor importante y por eso los lenguajes compilados no se utilizaron demasiado.

La desventaja de las aplicaciones CGI consiste en que el server debe realizar un fork, y ejecutar la aplicación o bien el interprete de la aplicación, y este ciclo que se cumple cada vez que se ejecuta la aplicación CGI insume muchos recursos y en general es costoso en tiempo para el server. Durante muchos años este esquema no muy eficiente dominó ampliamente el mundo de las aplicaciones Web.

Uso de una API del servidor:

Otra técnica factible consiste en utilizar una API (application programming interface) provista por el web-server para desarrollar aplicaciones, es decir que el web-server provee un lenguaje en el cual se pueden desarrollar aplicaciones. Este esquema, como podemos apreciar, es mucho más eficiente que el anterior ya que el web-server es el encargado de ejecutar las aplicaciones en forma directa sin necesidad de crear un proceso. Las desventajas son sin embargo importantes: en primer lugar las aplicaciones creadas en este marco no son portables ya que sólo pueden ejecutarse en un web-server determinado, esto es una gran desventaja frente a las aplicaciones CGI que podían una vez desarrolladas ejecutarse en cualquier servidor. La segunda gran desventaja es que frecuentemente un error de programación de una aplicación podría ocasionar que el web-server deje de funcionar, genere un error, se cuelgue, pierda memoria u otros problemas. Esto ocasiona que este tipo de aplicación no sea confiable.

Uso de un "módulo del web-server"

La tecnología más reciente para la ejecución de aplicaciones consiste en anexar a un web-server "módulos" que permiten al web-server interpretar un determinado lenguaje. De esta forma se logra eficiencia ya que el server no necesita crear un nuevo proceso por cada aplicación que ejecuta. Las aplicaciones son portables ya que son desarrolladas en un lenguaje standard que no depende del web-server, las aplicaciones son confiables ya que si bien pueden producir un error en el lenguaje en que están diseñadas si el módulo es sólido dichos errores no pueden comprometer al web-server. Al combinar las ventajas más importantes del ambiente CGI y un desarrollo basado en APIs evitando los inconvenientes de los mismos este esquema suele ser el más adecuado para ejecución de aplicaciones.

Cuadro Resumen:

	CGI (interpretado)	**CGI (compilado)**	**API del server**	**Modulo del server**
Ejemplos	Perl, Python	C, C++	Netscape Enterprise	PHP, ASP, Mod_perl, mod_python, FastCGI
Tiempo de desarrollo	Corto	Largo	Medio	Corto
Debugging	Sencillo	Complejo	Complejo	Sencillo
Confiabilidad	Alta	Alta	Baja	Alta
Eficiencia	Baja	Media	Alta	Alta

Benchmarks:

Test 1: 1000 ejecuciones de un programa de 1 línea.

Tecnología	**Tiempo (segundos)**
CGI : C	20,6
CGI : Perl	23,8
CGI : Python	45,2
PHP	16,2
Mod_python	30,0
Mod_perl	16,6
FastCGI	16,4

Test 2: 100000 ejecuciones de un programa de 1 línea.

Tecnología	**Tiempo (segundos)**
CGI : C	2522
CGI : Perl	2886
CGI : Python	5486
PHP	2420
Mod_python	3519
Mod_perl	2117
FastCGI	2120

Test 3: 10000 ejecuciones de un programa de 7000 líneas.

Tecnología	**Tiempo (segundos)**
CGI : C	258
CGI : Perl	963
CGI : Python	978
PHP	304
Mod_python	347
Mod_perl	476
FastCGI	280

En los tests puede verse como las variantes CGI insumen mucho tiempo de "lanzamiento" de la aplicación. Esto se prueba ejecutando muchas veces aplicaciones muy chicas (solo 1 línea de código). De esta forma podemos ver como en este aspecto las tecnologías que no requieren que el server genere un nuevo proceso (php, fastcgi, mod_perl, mod_python) son mucho más eficientes que aquellas que sí lo necesitan.

En el tercer test donde se ejecuta una aplicación de gran tamaño se puede apreciar que la variante "compilada" en CGI es muy eficiente ya que no requiere tiempo alguno de interpretación y el tiempo necesario para generar el proceso es mínimo en comparación con el tamaño de la aplicación. Lo importante de este tercer test es que tanto php, como mod_perl o fastcgi pese a requerir de un interprete son también veloces en este tercer test.

Evolución de los Web-Sites

Desde la aparición de la web distintos "modelos" o prototipos de web-sites fueron siendo predominantes en distintos periodos de tiempo, de acuerdo a las características "comunes" de los sitios predominantes en cada momento podemos diferenciar 3 generaciones de web-sites distintas:

Web-Sites de primera generación:

En un web-site de primera generación las páginas se desarrollaban, se subían al servidor y el servidor se encargaba de enviar las páginas al browser, es un modelo basado en páginas estáticas, en donde predominaba el uso de texto, links a otros sites o a otras páginas del mismo site y listas con "bullets" para enumerar cosas. Frecuentemente se usaban líneas horizontales "rules" para separar contenidos y las páginas de gran extensión vertical con gran cantidad de texto, listas y links eran comunes. Si bien no eran visualmente atractivos estos sites estaban enfocados a funcionar en forma veloz y entregar al usuario gran cantidad de información interrelacionada.

Web-Sites de segunda generación:

La segunda generación de web-sites implicó una revolución en lo visual, a medida que los web-sites se volvían emprendimientos más comerciales que científicos el hecho de "capturar" usuarios se torno una premisa y por ello se le dio gran importancia al aspecto visual. Las páginas con "layouts" visualmente atractivos fueron más y más populares, el uso abundante de imágenes, imágenes animadas y elementos multimedia se volvió común, aparecieron páginas que controlaban los fonts, el estilo y la posición en la que los mismos se ubicaban. La gran mayoría de los sites usaban tablas HTML para controlar la posición exacta en que los elementos aparecerían en el browser y tags nuevos en html como "font" u otros para controla la forma en la cual los mismos se verían. El concepto fue dominar la presentación de la información. Otra característica importante de esta segunda generación de Web-sites fue la aparición explosiva de más y más aplicaciones a medida. Los servicios que ofrecía un site se volvían factores importantes en la atracción de usuarios, chats, foros de discusión, banners, contadores y muchas aplicaciones más empezaron a aparecer y las falencias del protocolo CGI, a medida que las aplicaciones eran mas grandes y la cantidad de usuarios crecía exponencialmente, comenzaron a hacerse notar.

Web-Sites de tercera generación:

La tercera generación de web-sites siguió basada en lo visual, el gran cambio vino en la forma cómo se generaba la información. Las páginas estáticas que dominaban el 100% de los sitios de primera y segunda generación fueron reemplazadas por páginas dinámicas que el web-server generaba en el momento, a partir de información que en general se guardan en una base de datos. Estos sitios "dinámicos" permiten actualizar la información e incluso cambiar completamente la forma en que se muestran dichos datos en velocidades asombrosas. Los sitios de tercera generación facilitaron las aplicaciones interactivas, la información en tiempo real y que día a día se ofrecieran nuevos servicios. Las aplicaciones empezaron a desarrollarse también usando otras tecnologías dejando de lado el protocolo CGI. Aplicaciones en ASP, mod_perl o PHP, mucho más poderosas y eficientes que sus pares CGI, son el standard de este tipo de sitios.