Desarrollo web en entorno servidor

UT 1.1 – Arquitectura y herramientas web

5 – La web y sus componentes – El protocolo HTTP

IES Clara del Rey – Madrid

World Wide Web

Servicio diseñado inicialmente para distribución de información.

Permite la distribución de múltiples tipos de información

- Texto
- Imágenes
- Video
- Audio

La información se encuentra distribuida por servidores en todo el mundo. A veces los servidores replican la información entre ellos.

Hay sistemas de indexación y búsqueda contenidos, como Google.

W3C – Estandarización

La primera versión de lo que conocemos como World Wide Web fue desarrollada por el CERN en 1989.

Actualmente su desarrollo se impulsa por el World Wide Web Consortium (W3C). Desarrolla estándares para su uso en la web:

- HTML
- XML
- CSS
- WCAG
- WAI-ARIA
- Otros.

Componentes de la web

Recursos: documentos, videos, texto, aplicaciones, audio, etc., que están disponibles para que los usuarios los consulten.

URI y URL: sistema de nombres y direcciones que identifican de forma inequívoca un recurso en la red.

Clientes web: navegadores, que permiten a los usuarios acceder a la web y recuperar los recursos usando su URL.

Servidores web: atienden las peticiones de los clientes y les envían los recursos solicitados.

Servidores de aplicaciones: pueden ejecutar lógica de negocio compleja y manejar transacciones. Los servidores web sirven contenido estático (como HTML) y manejan solicitudes HTTP básicas.

Componentes de la web

Proxies web: sistemas intermedios, ubicados entre los clientes y los servidores.

- Pueden actuar como filtro o cortafuegos, impidiendo el acceso a ciertos servidores.
- También sirven para acelerar la navegación web por medio del uso de la caché.

Protocolo HTTP: protocolo que utilizan los navegadores y los servidores web para comunicarse entre sí.

Tecnologías web: conjunto de técnicas y tecnologías usadas para la implementación de sitios web. HTML, CSS, XML, Ajax, JSON, SVG, ...

Modelo cliente–servidor

La web usa el modelo cliente / servidor.

El servidor pone a disposición de los clientes una serie de información, y se mantiene a la espera de peticiones entrantes

Los clientes se conectan a los servidores (inician la comunicación) y realizan la petición de la información que desean recuperar.

Clientes web (navegadores web): Google Chrome, Opera, Firefox, Microsoft Edge, Safari, Tor, Vivaldi, ...

Servidores web o de aplicaciones: Apache, IIS, NGINX, Lighthttpd, Node.js, Apache Tomcat, JBoss, ...

URI – URL – URN

Tres términos relacionados que a veces se confunden.

URI:

- Uniform Resource Identifier. Identificador uniforme de recurso.
- Cadena de caracteres que identifica un recurso en la red.
- Las URI se dividen en URL y URN

URL:

- Uniform Resource Locator. Localizador uniforme de recurso.
- Permite no sólo identificar, sino localizar un recurso en la red.
- Incluye información sobre la forma (protocolo) de acceso al recurso: http://, https://, ftp://, telnet://, etc.

URI – URL – URN

URN:

- Uniform Resource Name. Nombre uniforme de recurso.
- Es un nombre del recurso, que debe ser único en la red.
- Es como el “DNI” del recurso.
- No indican donde se encuentra el recurso, como sí hacen las URL.
- Es de la forma URN:NID:NSS, donde:
  - URN: Todas las URN comienzan así.
  - NID: Namespace ID. Identifica el tipo de URN que estamos usando.
  - NSS: Namespace specific string. El valor específico para el recurso.

URI – URL – URN

Ejemplos de URL:

- https://www.google.com/search?q=http
- ftp://ftp.rediris.es/welcome.msg
- http://www.miweb.co:2048/noticias/noticia.html

Ejemplos de URN:

- urn:isbn:0451450523 – Libro "The Last Unicorn".
  ISBN (International Standard Book Number)
- urn:ISSN:0167-6423 – Revista "Science of Computer Programming".
  ISSN (International Standard Serial Number)
- urn:lex:eu:council:directive:2010-03-09;2010-19-UE
  Directiva de la unión europea según Lex URN Namespace

Estructura básica de una URL

schema://[user[:password]@]host[:port][/path][?query][#fragment]

- Schema: protocolo / forma de acceso al recurso.
- User y password: Credenciales para acceder al recurso. Opcional.
- Host: servidor en el que se ubica el recurso. Nombre DNS o IP.
- Port: puerto TCP del servidor que admite conexiones. Sólo si el puerto no es el estándar del protocolo utilizado.
- Path: ubicación del recurso dentro del servidor
- Query: para recursos dinámicos (ejecutables). Parámetros necesarios para su ejecución. Habitual en aplicaciones web.
- Fragment: referencia interna dentro del recurso solicitado.

Ejemplos de URL web

https://www.portalento.es/AreaPrivada/Login.aspx

- Schema / protocolo: HTTPS.
- Host: www.portalento.es
- Path: /AreaPrivada/Login.aspx

https://www.portalento.es:443/AreaPrivada/Login.aspx

- Equivalente a la anterior. 443 es el puerto de HTTPS.

https://juan:erS3@www.contoso.com/productos?s=portatil&o=1

- Schema: https
- Usuario y contraseña: juan / erS3423
- Host: www.contoso.com
- Path: productos
- Query: ?s=portatil&o=1

HTTP y HTTPS

HTTP: HyperText Transfer Protocol

HTTPS: HyperText Transfer Protocol Secure

Puertos estándares:

- 80/TCP para HTTP
- 443/TCP para HTTPS

Es un protocolo sin estado. Esto quiere decir que el servidor no guarda, entre una petición y otra, estado del cliente.

El estado se suele implementar con mecanismos que están por encima de HTTP en la pila de protocolos. Por ejemplo, cookies y sesiones.

Funcionamiento básico

1.- El usuario introduce una URL en el navegador, o navega a una página usando un enlace de otra página, de un email, un pdf, etc.

2.- El navegador analiza la URL y establece una conexión TCP con el servidor, para lo que:

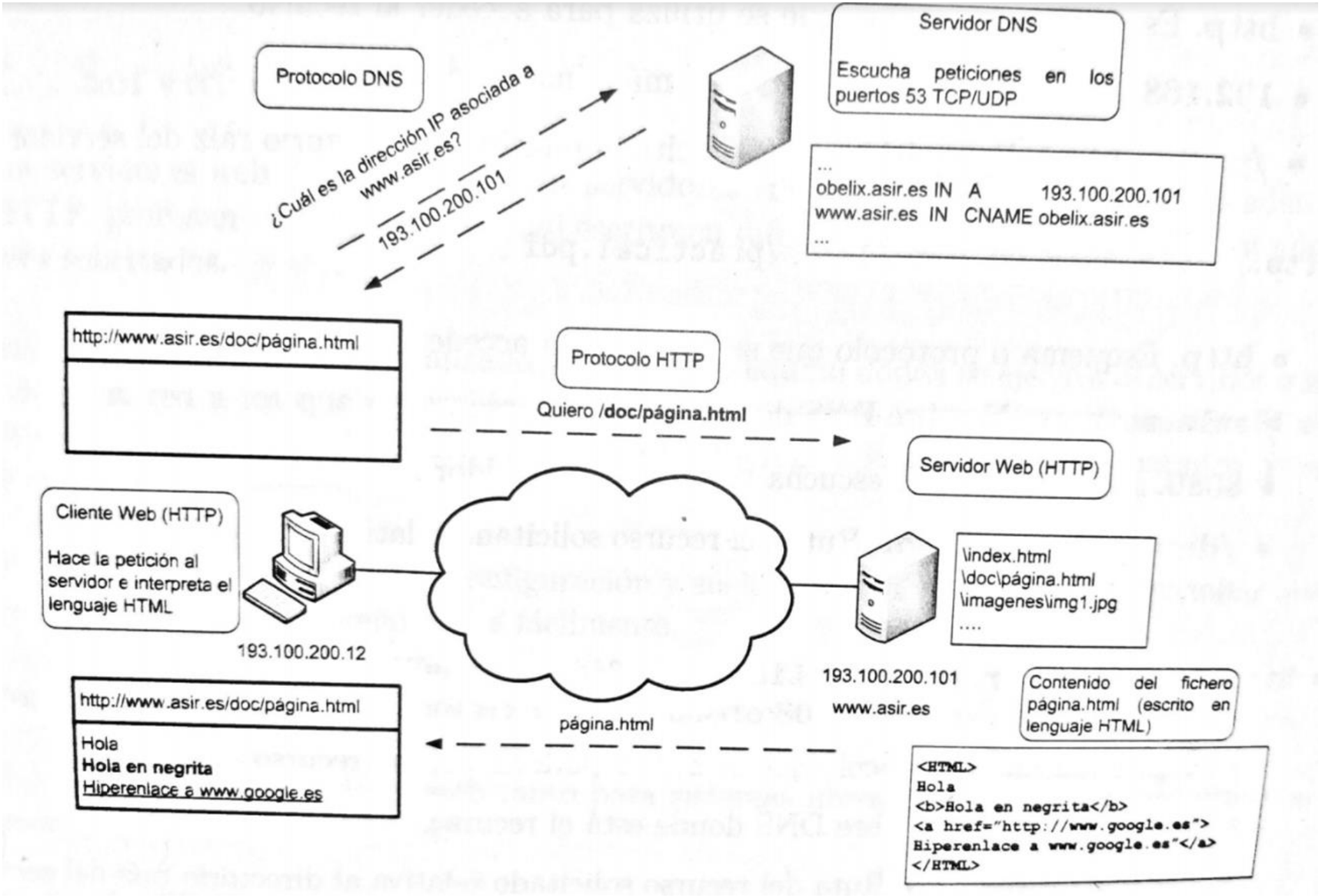
- Si es necesario, usa el servicio DNS para resolver (traducir) el nombre del servidor a una dirección IP.
- Establece una conexión TCP con el servidor en el puerto indicado en la URL. Si no se indica puerto, se usa el puerto por defecto del protocolo.

3.- El cliente envía un mensaje de petición HTTP.

4.- El servidor envía un mensaje de respuesta al mensaje del cliente.

5.- Se cierra la conexión.

Esquema de funcionamiento



Mensajes HTTP – Estructura

Peticiones y respuestas tienen una estructura similar, compuesta de:

- Una línea de inicio o inicial. Su contenido puede ser:
  - Peticiones: solicitud que se realiza al servidor
  - Respuesta: estado de éxito o fracaso de la petición respondida.
- Una segunda línea o grupo de líneas, con cabeceras HTTP.
- Una línea vacía que separa las cabeceras del cuerpo del mensaje.
- Un campo de cuerpo del mensaje.

No todas las líneas tienen por qué tener cuerpo del mensaje. Depende del tipo de petición y del contenido (cuerpo) que transporte

Mensajes HTTP – Ejemplos

Petición HTTP:

```
GET / HTTP/1.1\r\n
Host: apache.org\r\n
Connection: keep-alive\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
Accept: text/html,application/xhtml+xml,applicati
Accept-Encoding: gzip, deflate\r\n
Accept-Language: es-ES,es;q=0.9,en;q=0.8\r\n
\r\n
```

Mensajes HTTP – Ejemplos

Respuesta HTTP – Línea inicial y cabeceras:

```
HTTP/1.1 200 OK\r\n
Server: Apache\r\n
Last-Modified: Sun, 17 Oct 2021 15:49:02 GMT\r\n
ETag: "11fad-5ce8e5e0ca323-gzip"\r\n
Cache-Control: max-age=3600\r\n
Expires: Sun, 17 Oct 2021 16:49:05 GMT\r\n
Content-Encoding: gzip\r\n
Content-Type: text/html\r\n
Via: 1.1 varnish, 1.1 varnish\r\n
Content-Length: 16875\r\n
Accept-Ranges: bytes\r\n
Date: Sun, 17 Oct 2021 16:08:21 GMT\r\n
Age: 1156\r\n
Connection: keep-alive\r\n
X-Served-By: cache-hel6826-HEL, cache-mad22034-MAD\r\n
X-Cache: HIT, HIT\r\n
X-Cache-Hits: 4, 1\r\n
X-Timer: S1634486901.371888,VS0,VE1\r\n
Vary: Accept-Encoding\r\n
\r\n
```

Mensajes HTTP – Ejemplos

Respuesta HTTP – Cuerpo de la respuesta:

```
\r\n
[HTTP response 1/6]
[Time since request: 0.005016000 seconds]
[Request in frame: 322]
[Next request in frame: 348]
[Next response in frame: 375]
[Request URI: http://apache.org/]
Content-encoded entity body (gzip): 16875 bytes -> 73645 bytes
File Data: 73645 bytes
ne-based text data: text/html (1106 lines)
<!DOCTYPE html>\n
<html lang="en">\n
<head>\n
  <meta charset="utf-8">\n
  <meta http-equiv="X-UA-Compatible" content="IE=edge">\n
  <meta name="viewport" content="width=device-width, initial-scale=1">\n
  <meta name="description" content="Home page of The Apache Software Foundation">\n
  <link rel="apple-touch-icon" sizes="57x57" href="/favicons/apple-touch-icon-57x57.png">\n
  <link rel="apple-touch-icon" sizes="60x60" href="/favicons/apple-touch-icon-60x60.png">\n
  <link rel="apple-touch-icon" sizes="72x72" href="/favicons/apple-touch-icon-72x72.png">\n
  <link rel="apple-touch-icon" sizes="76x76" href="/favicons/apple-touch-icon-76x76.png">\n
```

Generado por herramienta de diagnóstico. No es parte de la respuesta.

Peticiones HTTP – Línea inicial

Tres elementos separados por espacios. Termina con CRLF.

- Método utilizado (GET, POST, PUT, DELETE, etc.).
  Indica al servidor tipo de operación que deseamos realizar.
- Path (ubicación) dentro del servidor del recurso que se solicita.
- Versión del protocolo HTTP utilizado. Opcional.

Ejemplo: GET /foundation/how-it-works.html HTTP/1.1

- Método utilizado: GET
- Path: /foundation/how-it-works.html
- Versión del protocolo: HTTP/1.1

En esta línea inicial no figura el servidor. Se usa una cabecera.

Peticiones HTTP – Cabeceras

Una línea por cabecera. Cada línea termina con CRLF. Cada línea tiene la forma "nombre: valor", donde:

- Nombre: nombre de la cabecera HTTP.
- Valor: valor que asignado a esa cabecera.

Ejemplos:

```
Host: apache.org
Cache-Control: no-cache
```

- Dos cabeceras, cada una con el valor asociado.
- La cabecera “Host” nos está indicando el servidor.

Detrás de la última cabecera se coloca una línea en blanco

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Peticiones HTTP – Cuerpo

Es opcional.

El cuerpo sólo se utiliza cuando la petición que estamos realizando necesita pasar parámetros o datos adicionales.

Por ejemplo:

- Cuando rellenamos un formulario con datos, en el cuerpo del mensaje se envían los datos del formulario.
- Cuando adjuntamos un fichero, en el cuerpo se envían los datos del fichero (nombre, tipo de contenido, y contenido, entre otros).

El tamaño y otras características del cuerpo se definen en las cabeceras.

Para el tamaño del cuerpo se usa la cabecera “content-length”.

Respuestas HTTP – Línea inicial

Tres elementos separados por espacios. Termina con CRLF.

- Versión del protocolo HTTP utilizado.
- Código de estado con el resultado de la petición.
- Texto explicativo del código de resultado.

Ejemplo:

HTTP/1.1 200 OK

- Versión del protocolo: HTTP/1.1
- Código de resultado: 200
- Descripción del código ("OK")

Respuestas HTTP – Cabeceras y cuerpo

Una línea por cabecera. Cada línea termina con CRLF.

Cada línea tiene la forma "nombre: valor", donde:

- Nombre: el nombre de la cabecera HTTP
- Valor: el valor que estamos asignando a esa cabecera

Ejemplo: Content-Type: text/html

Detrás de la última cabecera se coloca una línea en blanco, y a continuación el cuerpo.

El cuerpo es opcional. Algunas peticiones no solicitan contenido.

Lo más habitual es que devuelva el contenido (texto HTML, imagen, json, fichero, etc.)

Métodos HTTP – GET

El método más habitualmente utilizado, se usa para solicitar información al servidor. No envía información en el cuerpo del mensaje.

Utilizado al escribir una URL en el navegador, o al usar enlaces.

En los navegadores se puede repetir el GET con F5 (o CTRL+F5)

Puede enviar parámetros en la URL, en la parte “query”. Son de la forma nombre=valor, y se separan con “&”.

Ejemplo de URL con parámetros:

https://www.google.com/search?q=casa&lr=lang_es

- Parámetros: q y lr
- Valor de los parámetros: casa y lang_es

Métodos HTTP – POST

Envía información adicional en el cuerpo del mensaje.

Información contenida en el cuerpo no es visible en la URL, lo que es mejor, desde el punto de vista de la seguridad.

Es el método habitual para el envío de formularios.

No hay límite en el tamaño del cuerpo, por lo que se usa, por ejemplo, para enviar ficheros.

Las operaciones POST, en términos generales, no deben repetirse, por este motivo:

- No se recargan automáticamente con F5. Se pide confirmación al usuario para repetir la operación.
- No se cachean.

Métodos HTTP – Otros métodos

OPTIONS: Pregunta al servidor opciones de comunicación disponibles para un recurso. Puede hacerse antes de realizar un get o un post para verificar que es posible realizar la siguiente petición.

HEAD: Es igual que un GET, pero no devuelve el cuerpo, sólo las cabeceras. Por ejemplo, útil para saber si un recurso se ha actualizado.

PUT: Solicita cargar (subir) un recurso en el servidor.

DELETE: Solicita que se elimine un recurso del servidor.

PATCH: Solicita que se actualice parcialmente un recurso del servidor.

TRACE: Permite trazar la ruta a un recurso, para verificar la conectividad.

Veremos algunos de estos métodos con más detenimiento al desarrollar servicios web.

Cabeceras HTTP

Hay más de 100 cabeceras HTTP usadas de forma más o menos habitual.

Además de las estándares, cualquier aplicación web puede definir y usar cabeceras personalizadas.

HTTP no define límite para el número de cabeceras que puede incluir un mensaje. Los servidores web pueden establecer ciertos límites.

Se pueden clasificar en función de su utilidad. Entre otros, tenemos:

- Generales. Información aplicable a la petición o a la respuesta.
- De petición (a veces denominada de cliente)
- De respuesta (a veces denominada de servidor)
- De entidad. Información sobre el recurso solicitado / devuelto.

Cabeceras HTTP – Ejemplos

Authorization: transmite las credenciales de un cliente al servidor. El servidor las procesa y decide si conceder acceso o no al recurso.

Content-Type: tipo de contenido del cuerpo del mensaje. Por ejemplo, “application/json” indicaría datos en formato json para que el servidor los procese, o que el servidor está devolviendo datos en formato json.

Content-Length: tamaño del cuerpo del mensaje.

Accept: tipos de contenido admitidos por el cliente como respuesta a su petición.

Cookie: información de cookies que se intercambian cliente y servidor. Muy habitual para mantener sesiones o seguimiento.

Códigos de respuesta HTTP

El servidor responde con códigos indicando el resultado de procesar la petición.

Se han definido rangos para los distintos códigos HTTP, aunque no se usan, aún, todos los códigos en cada rango.

No se recomienda que los servidores usen códigos no estándar (de los no definidos / utilizados). Esto no garantizaría la interoperabilidad entre el servidor y los posibles clientes.

Algunos de los códigos se suelen acompañar de ciertas cabeceras para dar información al cliente de cómo debe proceder.

Listado completo de códigos en https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Códigos de respuesta HTTP

100 – 199: Informativos. Avisan de que la petición se está procesando, pero no se ha terminado aún.

200 – 299: Éxito. Indican que una operación se ha completado con éxito.

300 – 399: Redirección. Indican al cliente que debe repetir la petición, usando una dirección diferente.

400 – 499: Errores en la petición del cliente, que el servidor no ha podido procesar. Un error de cliente puede ser credenciales no adecuadas, método no admitido, tamaño del cuerpo del mensaje demasiado grande...

500 – 499: Errores de proceso en el servidor. Aunque la petición era correcta (no se devuelve error 4xx), se ha producido un error interno. Por ejemplo, el servidor no ha podido conectar con la base de datos.

Códigos de respuesta HTTP – Ejemplos

200 OK – El servidor ha procesado correctamente la petición.

204 No Content – Igual que el 200, pero la petición no genera datos para la respuesta (no hay cuerpo del mensaje).

308 Permanent Redirect – Redirección que indica que el recurso se ha movido permanentemente. Normalmente la cabecera HTTP “Location” incluye la nueva dirección del recurso.

401 Not Authorized – Aunque indica “No autorizado” en realidad significa “No autenticado”. Quiere decir que el usuario debe identificarse antes de intentar acceder a ese recurso.

403 Forbidden – Este es el código “No autorizado”. Se conoce la identidad del cliente, pero no tiene acceso (permisos insuficientes).

503 Service Unavailable – El servidor no está disponible.

Tipos MIME

MIME: Multipurpose Internet Mail Extensions

Especificaciones para hacer posible el intercambio de distinto tipo de información en Internet.

Inicialmente se definieron para su uso en protocolos de correo (SMTP, POP3, IMAP), pero se extendieron al resto de protocolos.

Definen tipos y subtipos de información, que sirven para indicar la clase de contenido enviado / recibido.

Clientes y servidores usan MIME para decidir cómo procesar información.

IMPORTANTE: los navegadores usan el tipo MIME para decidir como procesar la información. No la extensión de fichero.

También se definen reglas para codificar caracteres no ASCII.

Tipos MIME – Estructura

Un tipo MIME es de la forma “tipo/subtipo”, siendo:

- Tipo: la categoría “general” del contenido. Por ejemplo, texto (text), imágenes (image) o vídeo (video).
- Subtipo: la categoría “específica” del contenido. Por ejemplo, para el tipo “text” tendríamos, entre otros, “text/plain”, “text/html”, “text/vcard”, “text/xml”, o “text/javascript”.

Hay algunos tipos mime que tienen un parámetro adicional.

Por ejemplo, para descargar ficheros usando la cabecera content-disposition. Si el servidor quiere forzar la descarga de un fichero llamado “balance.txt”, usará una cabecera de respuesta similar a esta:

```
content-disposition: attachment;filename=balance.txt
```

Tipos MIME – Cabeceras MIME

MIME define cabeceras que los protocolos pueden usar para transferir información sobre el contenido enviado. Son las siguientes:

- MIME-Version. Versión de MIME utilizada.
- Content-Type. Tipo de contenido enviado.
- Content-Transfer-Encoding: Cómo codificar caracteres binarios (no ASCII).
- Content-ID: Identificador único del contenido.
- Content-Description: Descripción adicional del contenido. No es necesario en una gran parte de los casos.
- Content-Disposition: Cómo tratar el contenido. Por ejemplo, ficheros adjuntos.

MIME en HTTP

HTTP utiliza los tipos MIME, entre otras cosas, para:

- En una petición, para:
  - Cabecera accept: Indicar al servidor tipos de contenido que acepta el cliente como respuesta.
- En una petición o en una respuesta (se pueden usar en ambas):
  - Cabecera content-type: para indicar el tipo de contenido del cuerpo. Sólo se usa cuando se necesita (si no hay cuerpo no hace falta).
  - Cabecera content-disposition. Para especificar la estructura de la petición. En concreto, cuando envía más de un recurso en el cuerpo del mensaje, se utiliza el valor “multipart” en esta cabecera.