



Servidores Web

Sesión 1: Protocolo HTTP.
Introducción a las aplicaciones
web JavaEE



Puntos a tratar

- El protocolo HTTP
- Introducción a Tomcat
- Introducción a las aplicaciones web



Protocolos

- Internet se construye sobre TCP/IP
 - TCP: Protocolo de transporte
 - IP: Identificación de las máquinas de la red
 - Comunicación mediante *sockets*
 - Cada aplicación utiliza un determinado puerto por defecto
- Protocolos de aplicación sobre TCP/IP
 - HTTP: Acceso a la web (puerto 80)
 - FTP: Transferencia de ficheros (puerto 21)
 - Telnet: Terminales remotos (puerto 23)
 - SMTP: Envío de correo (puerto 25)
 - POP3: Recepción de correo (puerto 110)
 - etc



URLs

- Identifican los recursos en Internet

`protocolo://servidor[:puerto]/fichero`

- El protocolo puede ser:

`http`: Acceso a documentos

`https`: Acceso seguro a documentos

`ftp`: Transferencia de ficheros

`file`: Acceso a ficheros locales

`news`: Acceso a noticias

`mailto`: Envío de correo

`telnet`: Conexión a una máquina remota

- Ejemplos:

`http://www.ua.es/index.htm`

`ftp://ftp.dccia.ua.es/pub/winzip.exe`

`mailto:pepe@yahoo.com`



Protocolo HTTP

- Comunicación entre cliente y servidor
 - El cliente solicita un documento del servidor
 - El servidor sirve el documento al cliente
- Mecanismo petición/respuesta
 - Se solicita el documento enviando un mensaje de petición HTTP al servidor
 - El servidor devuelve el documento requerido al cliente dentro de un mensaje HTTP de respuesta
 - Si el documento no puede ser servido, devolverá un mensaje HTTP de respuesta indicando el error producido
 - Sin estado
 - Cada petición es independiente para el servidor



Protocolo HTTP: mensaje de petición

- Lo envía el cliente al servidor HTTP
 - Solicita un recurso
- Se compone de:
 - Comando HTTP

Compuesto por: `Comando URI Protocolo`
Por ejemplo: `GET /index.htm HTTP/1.1`
 - Cabeceras

Información sobre la petición
La sección de cabeceras finaliza con una línea en blanco
`(\r\n\r\n)`
 - Contenido adicional

Bloque de datos de cualquier tipo



Protocolo HTTP: comandos

- Comandos `GET` y `POST`
 - Se utilizan para solicitar un documento al servidor
 - `GET` proporciona los parámetros en la URI

```
GET /servlet/envia?msg=hola&usr=miguel HTTP/1.1
```

- `POST` proporciona los parámetros en el bloque de contenido
- Otros comandos:
 - `OPTIONS`: Consulta opciones del servidor
 - `HEAD`: Solicita información sobre el recurso (no su contenido)
 - `PUT`: Guarda un fichero en el servidor
 - `DELETE`: Borra un fichero del servidor
 - `TRACE`: Muestra el camino seguido por la petición



Protocolo HTTP: comando GET

- Se realiza esta petición cuando pulsamos sobre un enlace en una página web
 - Si queremos proporcionar parámetros tendremos que incluirlos en la misma URL

```
<a href="pag.jsp?id=123&nombre=pepe">Pulsa Aqui</a>
```

- También se realiza cuando utilizamos formularios con método GET

```
<form action="pag.jsp" method="GET">  
  <input type="text" name="id" value="123">  
  <input type="text" name="nombre" value="pepe">  
  <input type="submit" value="Enviar">  
</form>
```

- Los datos introducidos en el formulario se envían en la URI

```
GET /pag.jsp?id=123&nombre=pepe HTTP/1.1  
<cabeceras>
```




Protocolo HTTP: comando POST

- Se realiza cuando utilizamos un formulario con método POST

```
<form action="pag.jsp" method="POST">  
  <input type="text" name="id" value="123">  
  <input type="text" name="nombre" value="pepe">  
  <input type="submit" value="Enviar">  
</form>
```

- Los parámetros se envían en el bloque de contenido

```
POST /pag.jsp HTTP/1.1  
<cabeceras>
```

```
id=123&nombre=pepe
```

- Se utiliza también cuando conectamos desde Java con un objeto `URLConnection` activando la salida con `setDoOutput`
 - Lo que escribamos en el flujo de salida se escribirá en el contenido



Protocolo HTTP: cabeceras de petición

- Envían información sobre
 - El agente de usuario (navegador)
 - La petición realizada
- Algunas cabeceras estándar son:

<code>Accept-Language</code>	Idiomas aceptados
<code>Host</code>	Host y puerto indicado en la URL (requerido)
<code>If-Modified-Since</code>	Sólo se desea el documento si ha sido modificado tras esta fecha
<code>User-Agent</code>	Tipo de cliente que realiza la petición

- Por ejemplo, según el idioma especificado en la petición, algunos servidores podrán devolver el documento en dicho idioma



Protocolo HTTP: mensaje de respuesta

- El servidor nos responderá con un mensaje HTTP de respuesta
- Este mensaje se compone de:
 - Código de estado:

Indica si se ha procesado correctamente o si ha habido un error

Ejemplo: `HTTP/1.1 200 OK`
 - Cabeceras

Información sobre el recurso y sobre el servidor

Se definen de la misma forma que las de la petición
 - Contenido

En el bloque de contenido se incluye el recurso devuelto, si se ha devuelto alguno



Protocolo HTTP: códigos de estado

- Indica el resultado de la petición
- Encontramos varios grupos de códigos:
 - 1XX: Códigos de información
 - 2XX: Códigos de aceptación
 - 200 OK: Se ha servido correctamente
 - 204 No content: No hay contenido nuevo
 - 3XX: Redirecciones, el documento ha sido movido
 - 4XX: Errores en la petición
 - 400 Bad request: El mensaje de petición tiene sintaxis errónea
 - 401 Unauthorized: El usuario no tiene permiso
 - 5XX: Errores en el servidor
 - 500 Internal Server Error: Error interno del servidor



Protocolo HTTP: cabeceras de respuesta

- El servidor también puede enviar cabeceras en la respuesta con información sobre
 - El documento devuelto
 - Las características del servidor
- Algunas cabeceras estándar de la respuesta son:

<code>Content-Length</code>	Longitud del contenido (en bytes)
<code>Content-Type</code>	Tipo MIME del contenido
<code>Last-Modified</code>	Fecha de modificación del documento

- Podemos establecer estas cabeceras también desde la cabecera del código HTML de nuestro documento:

```
<META HTTP-EQUIV="Cabecera" CONTENT="Valor">
```



Cookies: definición

- El protocolo HTTP no tiene estado
- Podemos implementar estado sobre HTTP utilizando cookies
 - No es propio del protocolo HTTP
 - Está soportado por la mayoría de los navegadores
- Las cookies se componen de

`nombre=valor`

- Se almacenan en el cliente
- Se envían en cada petición al servidor
 - Identifican al cliente en cada petición



Cookies: envío y recepción

- El servidor envía una cookie al cliente con la cabecera `set-cookie`

```
Set-Cookie: CLAVE1=VALOR1;...;CLAVEN=VALORN [OPCIONES]
```

- Donde `OPCIONES` es

```
expires=FECHA;path=PATH;domain=DOMINIO;secure
```

- El cliente almacena la cookie de forma local
- En sucesivas peticiones al servidor se envía la cookie en la cabecera `cookie`

```
Cookie: CLAVE1=VALOR1;CLAVE2=VALOR2;...;CLAVEN=VALORN
```



Autenticación de usuarios

- Necesitamos autenticar usuarios para acceder a recursos restringidos
 - Se realiza mediante cabeceras de autenticación
- Tipos de autenticación
 - Básica
 - Codificación Base64
 - Los datos (login y password) viajan inseguros por la red
 - Digest
 - Se envía el password encriptado por la red (MD5)
 - No soportada por algunos servidores
- Certificados
 - Nivel de seguridad mayor



Certificados y SSL

- Clave pública
 - Par de claves asimétrico
 - Una de las claves sirve para descryptar los datos encriptados por la otra (RSA)
 - Estas claves se almacenan en forma de certificados (firmados)
- SSL (*Secure Sockets Layer*)
 - Capa entre HTTP y TCP
 - Gestiona la seguridad mediante clave pública
 - Establece una conexión mediante protocolo HTTPS (HTTP + SSL)

`https://www.ua.es`



El servidor web Tomcat

- Servidor web construido sobre la plataforma Java
 - Necesitamos tener instalado JDK para utilizarlo
- Soporta parte de la especificación de Java EE para desarrollar aplic. web (servlets y JSPs)
- Instalamos el servidor web descomprimiéndolo en el directorio escogido
 - En Windows contamos con un instalador
- Establecemos variables de entorno

`JAVA_HOME`: Directorio de JDK

`CATALINA_HOME`: Directorio donde hemos instalado Tomcat



Ejecución de Tomcat

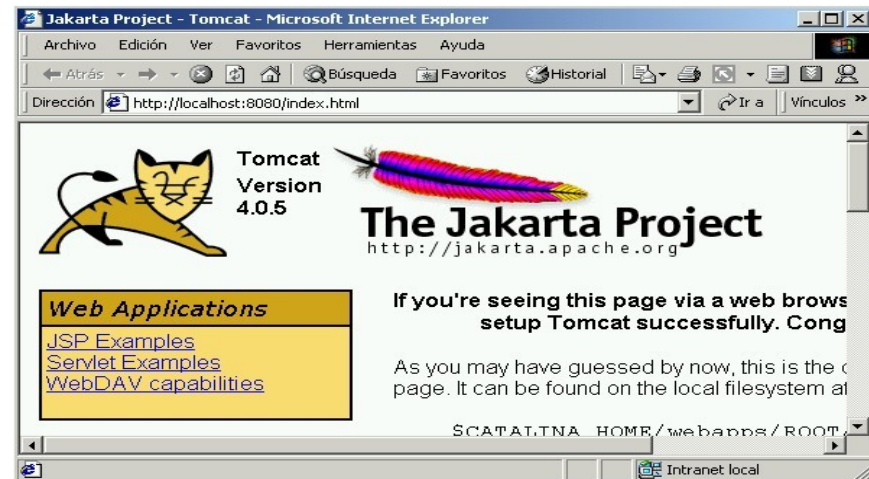
- En Linux contamos con los comandos:

`${CATALINA_HOME}/bin/startup.sh`: Activar el servidor

`${CATALINA_HOME}/bin/shutdown.sh`: Detener el servidor

- En Windows tenemos un *Monitor* en el menú *Inicio – Programas*, que permite iniciar y parar el servidor con el ratón
- Una vez en marcha podemos acceder a su página de bienvenida:

`http://localhost:8080/`





Ficheros WAR

- Podemos empaquetar las Aplicaciones Web en ficheros WAR (Archivos de Aplicación Web)
- Se utiliza la misma herramienta JAR para crearlos (sólo utilizamos una extensión distinta)
 - En realidad es un .zip con información adicional (MANIFEST)
 - Contendrá la estructura de directorios completa del contexto
- Es un estándar de los servidores de aplicaciones Java EE
- Se utiliza para distribuir aplicaciones web
 - Podremos copiar el fichero WAR directamente al servidor web para poner en marcha la aplicación



Ejemplo WAR

- Dada la siguiente estructura de carpetas:

```
C:/web/ejemplo/  
    index.html  
    WEB-INF/  
        web.xml  
        classes/  
            ClaseServlet.class
```

- Entrar en el directorio *C:/web/ejemplo* y teclear

```
jar cMvf ejemplo.war *
```



Despliegue de .war

- Manualmente
 - Copiar el archivo al directorio **webapps** de Tomcat
- “Tomcat manager” (<http://localhost:8080/manager/html>)
 - Requiere un usuario con rol “manager” (ya definido en Tomcat 6)
 - También se puede gestionar el ciclo de vida de las aplicaciones (replegar, parar, arrancar)



Gestor de Aplicaciones Web de Tomcat				
Mensaje:	OK			
Gestor				
Listar Aplicaciones	Ayuda HTML de Gestor	Ayuda de Gestor	Estado de Servidor	
Aplicaciones				
Trayectoria	Nombre a Mostrar	Ejecutiéndose	Sesiones	Comandos
/	Welcome to Tomcat	true	0	Arrancar Parar Recargar Replegar <div>Expire sessions with idle ≥ 30 minutes</div>