MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE WEB ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

El desarrollo web

01

ÍNDICE

OBJETIVOS

Identificar qué es el servidor en una solución web

Conocer las tareas del cliente en una solución web

Comprender el protocolo de comunicación en la web

Saber el funcionamiento de la arquitectura C/S

/ 1. Introducción y contextualización práctica

El desarrollo web se refiere a todo el trabajo necesario para crear un sitio web que pueda estar disponible tanto en Internet como en una red privada. Dentro del desarrollo web podemos encontrarnos un abanico muy grande de soluciones, desde una simple página web estática, hasta una aplicación con fines comerciales o, incluso, una red social.

Dependiendo del tipo de desarrollo web, pueden verse implicados diferentes profesionales. Normalmente, cuando creamos una aplicación web podemos ver, al menos, dos partes muy bien diferenciadas: el cliente y el servidor.

El servidor requiere del uso de lenguajes de programación complejos y una arquitectura interna que pueden influir directamente en el funcionamiento global del sistema, por ejemplo, con el acceso a los datos. El cliente, por su parte, es el encargado de facilitar la interacción con el usuario, es decir, es la interfaz gráfica con la que podemos interactuar. Es la puerta de entrada a todo el sistema. En este tema abordaremos los fundamentos básicos para el desarrollo web en el lado del cliente.

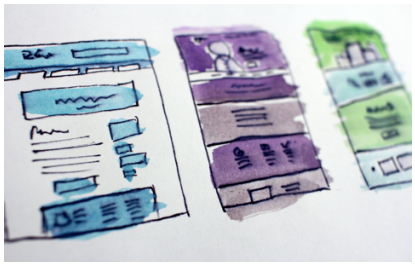

Fig. 1. Fases del desarrollo web

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad:

Audio Intro. Peticiones HTTP en cliente/servidor

https://bit.ly/3cLtWJv

/ 2. Fundamentos del desarrollo web

Para entender cómo funciona un sitio web, es necesario, en primer lugar, saber qué es. Podríamos decir que un sitio web es simplemente un conjunto de archivos que se almacena en una máquina, normalmente denominada servidor. Como el servidor está conectado a Internet, es posible que otras aplicaciones, conocidas como navegadores, puedan conectarse y visualizar el contenido. Cada una de las máquinas que se conecta al servidor recibe el nombre de cliente.

En este escenario, cada vez que el cliente se conecta al servidor a través de un navegador, podrá cargar los datos y visualizar el contenido.

Como desarrollador web, es importante determinar en qué parte del sistema vamos a trabajar: en el servidor (back-end), en el cliente (front-end) o en el cliente y servidor (fullstack).

Si nos centramos en el desarrollo en front-end, es necesario conocer, al menos, HTML, CSS y JavaScript.

2.1. Arquitectura cliente/servidor

La arquitectura cliente/servidor es un modelo que permite crear aplicaciones distribuidas que, como su propio nombre indica, son capaces de distribuir diferentes tareas. Un posible uso de este tipo de arquitectura es lograr un balanceo correcto de las tareas que se deben realizar. En esta imagen se pueden diferenciar claramente dos elementos: cliente y servidor.

Fig. 2. Conexión del cliente con el servidor

Normalmente, un cliente es capaz de comunicarse con el servidor a través de una red de comunicaciones. Sin embargo, hay situaciones en las que tanto el cliente como el servidor se ejecutan en la misma máquina. En este caso, ambos se comunicarían a través de un bus de altas prestaciones.

2.2. Arquitectura cliente/servidor: cliente-servidor

Como norma general, el cliente nunca comparte sus recursos con el servidor.

Audio 1. Arquitectura cliente/servidor
https://bit.ly/2Yqin5m

En el desarrollo de software, existe una separación lógica entre la capa de presentación (front-end) y la capa de acceso a datos (back-end). En relación con la arquitectura cliente/servidor, el front-end caería en el lado del cliente mientras que el back-end se centraría en el lado del servidor.

2.3. Tipos de arquitecturas

Como hemos visto anteriormente, el modelo cliente/servidor se fundamenta en la realización de peticiones por parte del cliente y la generación de respuestas por parte del servidor. Dependiendo de

las tareas que realice el servidor, se pueden diferenciar algunas modificaciones:

- **Arquitectura en dos capas.** Es el modelo por defecto en el que el servidor, al recibir una petición, es capaz de responder con sus propios recursos.

- **Arquitectura en tres capas.** En este caso, generalmente, el servidor requiere del acceso a un componente externo, como puede ser un servidor de datos.

- **rquitectura en N capas.** Este escenario supondría la generalización del modelo en el que se podrían introducir tantas capas de datos, de procesamiento, etc. como el problema requiera.

2.4. Consideraciones de la arquitectura cliente/servidor

Esta arquitectura presenta ciertas ventajas para la implantación de soluciones web, como, por ejemplo:

- No es dependiente de una plataforma, es decir, los usuarios pueden utilizar el sistema operativo que deseen.
- Existe un alto nivel de interacción en las interfaces gráficas que, además, son ligeras.
- Permite que el sistema se pueda escalar fácilmente sin la necesidad de realizar ninguna modificación en el cliente.
- Ofrece una estructura modular para cada uno de los componentes.
- Sin embargo, también podemos encontrar algunas limitaciones del modelo:
- El sistema puede fallar si centralizamos todo en un servidor, pues, en caso de que este caiga, ningún usuario se podrá conectar.
- En términos económicos, en ocasiones, puede resultar una solución costosa, pues los servidores suelen requerir un alto nivel de hardware.

Fig. 3. Sistema cloud basado en cliente/servidor

/ 3. Caso práctico 1: "Back-end"

Planteamiento. Una vez implantado un sistema mediante la arquitectura cliente/servidor, se permite que los usuarios puedan realizar peticiones al back-end. En un momento determinado, se observa que el número de clientes aumenta considerablemente. Al mismo tiempo, se observa que la mayor parte del tiempo el servidor está resolviendo el acceso a datos.

Nudo. ¿Qué crees que puede estar pasando? ¿Cómo lo solucionarías?

Desenlace. Según hemos visto en la descripción del modelo cliente/servidor, se está siguiendo un modelo por defecto en el que el servidor también realiza el procesamiento de datos. Al crecer el número de usuarios, los recursos hardware requeridos aumentan. Si a esto le sumamos que el servidor debe resolver los datos, tendremos un cuello de botella. Para solucionarlo, es mejor migrar a una arquitectura en 3 capas, por lo menos, en la que los datos se gestionen en otra máquina y el servidor pueda seguir atendido a los clientes.

Fig. 4. Toma de decisiones en el back-end

/ 4. Protocolo HTTP

En la computación, un protocolo de comunicación permite establecer las reglas necesarias para que unos dispositivos se comuniquen con otros.

En entornos web, el protocolo utilizado para la transferencia de contenido web es HTTP (Hypertext Transfer Protocol). Este protocolo funciona mediante un sistema de peticiones/respuesta dentro de una arquitectura cliente/servidor. Un funcionamiento básico implica una solicitud del navegador al servidor, petición que viaja utilizando el protocolo HTTP. Ante esta solicitud, el servidor puede crear una respuesta y enviar el contenido web que visualizaremos mediante el mismo protocolo.

En otros casos, el servidor puede recibir una petición para realizar alguna tarea, por ejemplo, almacenar algún tipo de contenido. Independientemente del mensaje, el protocolo utilizado sigue siendo HTTP. Al recibirse la respuesta, este protocolo guarda un campo que indica cómo se ha procesado la petición. Los posibles valores de una petición pueden ser los siguientes:

- **1XX:** respuestas informativas
- **2XX**: respuestas de ejecución correcta.
- **3XX**: respuestas que indican redirección.
- **4XX**: respuestas que indican errores en el cliente.
- **5XX**: respuestas que indican errores en el servidor.

Por otro lado, hay que tener en cuenta que el protocolo HTTP no es capaz de almacenar información del estado de una comunicación, por lo que el cliente deberá guardar cierta información para ello.

Para poder navegar por la web, es necesario que diferenciemos los siguientes conceptos:

- **URL** (Uniform Resource Locator). Es conocida normalmente como la dirección web de una página.
- **URI** (Uniform Resource Identifier). Permite identificar un recurso en la red

Fig. 5. Ejemplo de introducir una URL en el navegador

4.1. Peticiones HTTP

El protocolo HTTP se basa en mensajes. En la versión HTTP 1.1 los mensajes soportados son:

- **GET**. Permite pedir recursos a través de los parámetros pasados por la URL.
- **POST**. Permite pedir recursos a través de los parámetros pasados en el cuerpo del documento HTML.
- **HEAD**. Ejecuta una petición de datos para un recurso determinado.
- **PUT**. Permite enviar datos a un servidor.
- **DELETE**. Indica al servidor que debe borrar el recurso indicado en la petición.
- **TRACE**. Se centra en enviar peticiones para funciones de depuración.

- **OPTIONS**. Realiza una petición para conocer las funciones que tiene el servidor.
- **CONNECT.** Se guarda para los servidores intermedios.

/ 5. Seguridad en la comunicación web

A medida que la comunicación web evoluciona, podemos realizar mayor cantidad de tareas a través de la red. Un ejemplo de ello podría ser el comercio electrónico. Para poder ejecutar nuestras tareas a través de la red de forma segura, es necesario que la solución web creada incorpore mecanismos de seguridad. Por este motivo, debemos conocer las principales medidas de seguridad de estos entornos.

Fig. 6. Busca siempre comunicaciones seguras

Hablamos de un sistema de comunicación seguro cuando la transmisión que se lleva a cabo entre dos entidades no puede ser interceptada por un tercer individuo. Además, durante este proceso de comunicación debemos asegurarnos de que el destinatario es realmente quien dice ser, y que el mensaje que recibe es el que realmente hemos enviado.

5.1. HTTPS

En la transmisión web, el protocolo más utilizado es HTTPS (Hypertext Transfer Protocol Secure). Como se puede intuir, este protocolo está basado en HTTP.

El protocolo HTTPS permite la comunicación segura en un entorno web gracias a la utilización de cifrado de mensajes a través de SSL (Secure Socket Layer) / TLS (Transport Layer Security)

En la definición de protocolos de comunicación podemos encontrar diversos niveles que se encargan de diferentes funcionalidades. En este modelo por niveles, cada uno de ellos puede utilizar las funcionalidades del nivel inmediatamente inferior. Para una comunicación web, los niveles de mayor interés son estos:

- **Nivel de red**. Se trata de un nivel que únicamente ofrece la conectividad entre dos sistemas.
- **Nivel de transporte.** Se encarga de la transmisión de datos entre el emisor y el receptor, sin errores.
- **Nivel de aplicación**. Ofrece la funcionalidad directamente al usuario que puede utilizar, si es necesario, ese servicio.

Los protocolos HTTP y HTTPS pertenecen a la capa de la aplicación, pues los usuarios (o aplicaciones) pueden hacer uso de ellos si lo estiman oportuno. Ambos protocolos se apoyan directamente en la capa de transporte, en concreto en el protocolo TCP, y éste a su vez en el nivel de red, particularmente sobre el protocolo IP. La diferencia entre HTTPS y HTTP es que HTTPS incluye un protocolo adicional en la capa de transporte (SSL/TLS). Precisamente, esta se añade por encima de TCP (véase la Tabla 1).

Vídeo 1. Esnifando la red
https://bit.ly/2MLS8kz

Aplicación	HTTPS/HTTP
Transporte	SSL/TLS TCP
Red	IP

Tabla 1. Pila de protocolos que intervienen en HTTP/HTTPS

5.2. Limitaciones HTTPS

Como hemos visto, HTTPS funciona sobre la encriptación SSL y TLS. Nótese que la seguridad de un método de encriptación dependerá, en gran medida, de la tecnología disponible en cada momento. Por ejemplo, hace años el cifrado DES se consideraba seguro, pues el tiempo para poder descifrar un mensaje con la tecnología que existía en dicho momento se consideraba muy elevado (mayor que el tiempo de vida del ser humano). Sin embargo, actualmente DES se considera inseguro por diversos motivos, entre ellos, el tiempo requerido para descifrar un mensaje.

En cuanto a la encriptación SSL y TLS, dispone de dos formas de funcionamiento: simple o mutua. En cuanto a la simple, la autenticación se realiza únicamente por parte del servidor; con la mutua, el cliente deberá instalar un certificado para que el servidor lo verifique. Por ejemplo, en la actualidad se ha observado que SSL puede sufrir un ataque de MID (Man in the Middle), lo cual intercambia el protocolo a HTTP. No se trata de ataques sencillos, pero nos encontramos con situaciones en las que pueden ocurrir. De esta forma, HTTPS puede mostrar ciertas vulnerabilidades si se tiene la capacidad de analizar grandes volúmenes de datos y se dispone de la tecnología adecuada. Estos tipos de ataques se basan en el intento de averiguar alguna propiedad sobre el mensaje cifrado para poder obtener el mensaje enviado.

/ 6. Otros protocolos utilizados en la red

En el mundo web, los usuarios pueden utilizar los protocolos HTTP o HTTPS (según la solución que requiera) para comunicarse con el servidor a través del navegador. Sin embargo, desde el punto de vista del mundo de la programación, se deben considerar otros protocolos para poder completar y publicar nuestra nueva solución en el servidor.

En el desarrollo en cliente algunos de los protocolos más utilizados son: FTP y SSH

Fig. 7. Escenario de funcionamiento del protocolo FTP

6.1. FTP. File Transfer Protocol

Es un protocolo para poder realizar transferencia de archivos entre un cliente y un servidor a través de la red. Se encuentra disponible en todos los sistemas operativos de propósito general. En el desarrollo web lo podemos utilizar para la transferencia de archivos del cliente al servidor o viceversa. Por ejemplo, se puede utilizar para publicar la última versión de la web creada.

6.2. SSH. Secure Shell

Se trata de un protocolo para trabajar con seguridad en un entorno inseguro que sigue una arquitectura cliente/servidor. Normalmente requiere la utilización de la línea de comandos para interactuar con el sistema remoto. Para conectar con este sistema se nos pedirá un nombre de usuario y contraseña. Para poder funcionar, SSH requiere de un cliente SSH y un servidor SSH. Se encuentra disponible para todos los sistemas operativos de propósito general. Gracias a este protocolo, se pueden realizar tareas sobre una máquina remota con facilidad, como por ejemplo modificar un archivo, renombrar carpetas, etc.

Vídeo2. Protocolos FTP y SSH
https://bit.ly/3dRDnIW

/ 7. Caso práctico 2: "Errores"

Planteamiento. Una vez implantado un sistema mediante la arquitectura cliente/servidor, se permite que los usuarios puedan realizar peticiones al back-end mediante peticiones HTTP y utilizando un front-end desarrollado por nosotros. En este proceso, el servidor nos devuelve en primer lugar un error 500, luego un 404 y por último un error 403.

Nudo. ¿De qué nos está informando el servidor? ¿Qué significa cada uno de los mensajes? ¿Cuál de ellos arreglarías primero?

Desenlace. El error 500 nos informaría de un error interno del servidor. A su vez, tanto el error 404 como el 403 nos indicaría un error en el cliente. El error 404 nos indicaría que el cliente está realizando una petición a un recurso inexistente, mientras que el error 403 nos indicaría que el usuario no se ha identificado correctamente en el sistema. De entre estos errores, el único que podríamos solventar a nivel de servidor sería el 500, ya que los otros dos dependen de lo que esté realizando el cliente, por ejemplo, puede que no haya introducido correctamente su información de inicio de sesión.

Fig. 8. Error 404 HTTP

/ 8. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema hemos presentado los principales aspectos que debemos considerar en el desarrollo web. Concretamente, nos hemos centrado en la arquitectura cliente/servidor, con sus diferentes variantes. Además, hemos estudiado los distintos protocolos que pueden intervenir en la comunicación entre el cliente y servidor desde el punto de vista del usuario: HTTP y HTTPS. Como hemos visto, en una comunicación en la red, es muy importante garantizar la seguridad de la comunicación. Por este motivo, hemos presentado también algunas medidas de seguridad.

Desde el punto de vista del desarrollador hemos destacado otros protocolos que se pueden utilizar, como FTP y SSH.

Resolución del caso práctico de la unidad

Cuando un servidor recibe múltiples peticiones que no puede resolver, las va dejando en cola. Cuando el número de peticiones en cola crece, el tiempo de respuesta del servidor lo hace en la misma medida. Desde el punto de vista del usuario, se puede observar que la página aparece como «cargando» o «conectando». Aunque puede afectar a la experiencia de usuario, lo que puede estar pasando desde el punto de vista de la seguridad es un ataque DoS (Denial of Service). Este ataque supone una denegación de servicio, a través del cual provoca que el servidor no pueda atender a peticiones reales, mientras que se estén generando peticiones que solo vayan destinadas a «tirar» el servidor.

/ 9. Bibliografía

Ballard, P. y Moncur, M. (2009). Ajax, JavaScript y PHP. Madrid: Anaya Multimedia.

Sawyer McFarland, D. (2012). JavaScript y jQuery. Madrid: Anaya Multimedia.

Welling, L. y Thomson, L. (2005). Desarrollo web con PHP y MySQL. Madrid: Anaya Multimedia.