Arquitectura cliente-servidor. Protocolos

M1

Introducción

¿Alguna vez te has preguntado qué hay detrás de términos como «está en la nube», «bájalo de internet» o «la página no me responde»? Estas son frases que hemos escuchado miles de veces y la mayoría de las personas elaboran internamente definiciones más o menos certeras sobre dónde queda esa nube, qué es, dónde se guarda lo que está en internet y cómo es que cuando entramos en una página web podemos interactuar con ella.

En esta lectura nos introduciremos en el mundo web, partiendo de cómo se organiza la **arquitectura cliente-servidor** y cómo se realiza, en un alto nivel, *la comunicación entre estas dos partes utilizando protocolos específicos.*

1. Internet

¿Cómo surgió lo que hoy conocemos como internet? La red de redes le debe el nacimiento a desarrollos tecnológicos con fines bélicos, a fines de la década del 60, que buscaban el trabajo conjunto y redundante —o en paralelo— de computadoras ante posibles bajas. El concepto de nodos distribuidos territorialmente, con estructura rizomática, en comunicación entre sí y con redundancia de tareas o datos, en cierta medida hoy sigue representando el funcionamiento de internet. Pero faltaban muchos avances significativos, que sintetizamos en los párrafos siguientes.

Durante la década de 1970, se desarrolla la posibilidad de sumar más nodos y se incorpora el protocolo **Network Control Protocol (NCP)** de comunicación entre dichos nodos. Más tarde, se desarrolló el **protocolo TCP-IP, que permitió comunicar diferentes computadoras sin importar en qué red se encuentren**, en qué medio viaje la información o qué marca de computadora se utilice, homogeneizando el intercambio. Con el auge de las computadoras personales, en la década de 1980, se amplían los nodos en forma exponencial. *La posibilidad de contar con un equipo conectado dejaba de ser exclusividad de las fuerzas de seguridad, las grandes empresas, equipos científicos, entre otros pocos, permitiendo a quienes trabajaban en una oficina o hasta en los mismos hogares disponer de una computadora personal conectada, con la posibilidad de enviar y recibir información en redes privadas*.

Es en la década de 1990, cobran popularidad las redes abiertas, nombradas como la **World Wide Web**, que se transforma en un espacio público al que **cualquier persona con una computadora y conexión a internet puede conectarse para consumir información**. Crecen también los proveedores del servicio de internet *(ISP, por las siglas en inglés de Internet Service Providers).* Proliferan los browsers (**navegadores**) y aparecen los primeros buscadores.

Años más tarde, empiezan a aparecer páginas webs que permiten la participación de visitantes: **blogs, foros, y un sinnúmero de formatos.** En los últimos 20 años, hemos visto crecer esta participación para convertirnos cada uno en generadores de contenidos, a veces en igual medida que lo que consumimos.

Internet, como tal, tiene sus raíces hace unos cincuenta años, como hemos visto. Sin embargo, en esta materia nos interesa involucrarnos en solo la mitad de su historia, ya que nos enfocaremos en el funcionamiento y posibilidades de desarrollo dentro de la web. Podemos decir que internet es el medio de comunicación que utilizamos, y la web es solamente una de las muchas aplicaciones que tiene.

La web: arquitectura cliente-servidor

La arquitectura base de funcionamiento no ha cambiado en gran medida. A diferencia de las redes privadas anteriores, **la web toma la arquitectura cliente-servidor**. Esto nos permite pensar — *si bien vamos a explicarlo en singular, nombrando en general «el» cliente*— que podemos llegar a tener infinidad de clientes para un mismo único servidor.

Cañedo Andalia (2004) describe los cimientos de la arquitectura de la siguiente manera:

> La web opera sobre una arquitectura cliente-servidor. El software server almacena y el software cliente busca y recupera. El cliente también ofrece una interfaz para el usuario final. Cada uno de estos software pueden perfeccionarse de forma independiente. Un servidor web es un programa receptor de solicitudes de documentos hipertextos residentes en su computadora y el encargado de enviarlas a la máquina que los solicita; el cliente web es el encargado de enviar las solicitudes hechas por el usuario a la computadora que opera como servidor y visualizar las recibidas en pantalla. (https://lc.cx/w5dZTE)

La World Wide Web, en 1990, ya establecía cuatro elementos, a saber:

- Los **documentos de hipertexto**, que son usualmente representados en formato HTML (HyperText Markup Language), tema que abordaremos en otra lectura y a lo largo de los

diferentes módulos.

- Un **protocolo sencillo** para el intercambio de dichos documentos, que va a ser el protocolo de transferencia de hipertexto (HTTP, por las siglas en inglés Hypertext Transfer Protocol).
- Un **cliente** (muchos) que muestre esos documentos de la forma más homogénea posible, a pesar de las diferencias en las pantallas, resoluciones, etcétera, constituido por el navegador web o browser, nombrado por Cañedo Andalia (2004) como «cliente web».
- Un **servidor** para dar acceso a los documentos, «escuchando» las solicitudes (requests) y retornando las respuestas (responses).

Respecto del **protocolo HTTP**, ***haremos hincapié solamente en algunos aspectos que nos resultan relevantes***, trabajando con tecnologías de alto nivel como las que utilizaremos en esta materia. Algunos detalles los retomaremos en los sucesivos módulos, pero la mayoría solo nos servirán de consulta, para comprender las bitácoras de errores (*logs*).

¿Por qué es necesario un protocolo? **Los protocolos de comunicación son imprescindibles cuando queremos establecer un intercambio de datos entre dos equipos, de forma automatizada, ya que tanto el emisor como el receptor deben conocer el formato de los paquetes, la estructura de los mensajes, las formas de encriptación o codificación, etcétera**. En particular, HTTP nos permite las comunicaciones entre el cliente y el servidor en las aplicaciones webs.
Así como hay algunos lineamientos de HTTP que son imprescindibles de ser cumplidos para desarrollar una aplicación web, ya que de otro modo los browsers no podrán comunicarse con nuestro servidor, hay otros que podrían ser pasados por alto sin imposibilitarnos el funcionamiento. Sin embargo, recomendaremos, en esta y otras oportunidades, ceñirnos a los protocolos y estándares por los siguientes motivos:

- **Mantenimiento de la aplicación:** otros equipos o personas podrían tener que modificar, corregir o analizar el desarrollo actual; si este no cumple a rajatabla las recomendaciones estándar, tendría que ser acompañado por aclaraciones en comentarios y documentación adicional, lo cual reduce la expresividad de la solución.
- **Calidad de la implementación:** si es un estándar ya ha sido testeado ampliamente, podremos contar con cierta fiabilidad.
- **Configuración:** al tratarse del funcionamiento normalizado, si lo seguimos no tendremos que realizar configuraciones adicionales, en algunos casos engorrosas. Esta preferencia por

seguir convenciones a realizar configuraciones la llaman convention over configuration. Intentaremos seguir la recomendación siempre.

Para adentrarnos en este protocolo, utilizaremos a modo de ejemplo el siguiente caso práctico ficticio.

La Siglo es una editorial que comercializa sus libros de forma 100 % online. Hay un panel de administración, al que tienen acceso solamente determinados usuarios, donde pueden cargar nuevos libros, cada uno con su nombre, descripción, autores, fotos y precio. También, pueden ver el listado de todos los libros cargados y el detalle de un libro existente, así como modificarlo o darlo de baja.

Además, existe la tienda en sí misma, donde personas externas pueden consultar los libros disponibles, hacer los pedidos, abonarlos y demás. Pero, por el momento, nos enfocaremos en el panel de administración.

Los lineamientos del protocolo HTTP que nos interesa mencionar son los siguientes:

- **Identificador de recurso uniforme** (URI, por las siglas en inglés, uniform resource identifier). A cada recurso se lo identifica a través de la URI. También, nos referiremos a las URI como rutas de la aplicación. En nuestro caso, un recurso podría ser un libro, y algunos URI las siguientes: /libros, /libro/48 (para identificar el libro con identificador 48).
- **Métodos o verbos.** Cada solicitud (request) HTTP del cliente al servidor va identificada con un método o verbo que define la acción que se desea ejecutar sobre un recurso determinado. Dos solicitudes a la misma URI, pero con diferente método HTTP, identifican diferentes acciones sobre el mismo recurso.

- **GET.** *Lo utilizaremos para obtener información, consultar o leer.* Es esperable que no modifique el recurso. **Por ejemplo:** con solo conocer la solicitud al servidor es GET a la URI/libros, sabemos que se está intentando obtener para consultar un conjunto de libros, no se intenta modificar ni cargar nuevos en esa solicitud. Una solicitud GET a la URI /libros/nuevo podría ser para obtener un formulario y cargar los datos de un libro nuevo. Obsérvese que en esa solicitud tampoco se espera efecto de lado, ya que el libro no será creado hasta que el formulario sea completado en el cliente y enviado en una nueva

solicitud. Una solicitud GET a la URI /libros/123, nos mostraría los detalles del libro con identificador 123.

- **POST.** *Lo utilizaremos para enviar datos de una nueva instancia del recurso, por lo que es esperable que tenga un efecto de lado en el sistema*. **Por ejemplo**, una solicitud POST a la URI /libros, con los datos del nuevo libro que en el cliente se cargaron en el formulario como parámetros, se espera que cargue un nuevo libro.
- **PUT.** *Lo utilizaremos para modificar todos los campos de un recurso*. Es esperable que se altere el recurso. **Por ejemplo**, una solicitud PUT a la URI /libros/48, con parámetros como nombre, descripción y precio con valores nuevos, actualiza el libro con identificador 48 a dichos valores nuevos.
- **PATCH.** *Lo utilizaremos para modificar parcialmente un recurso, es decir, solo algunos campos.* Se espera que altere el recurso en cuestión. Cabe destacar que muchas veces este método es omitido, y simplemente se usa con menor cantidad de parámetros, entendiendo que solo hay que actualizar los campos enviados. **Por ejemplo**, una solicitud PATCH a la URI /libros/48, con parámetro solamente de precio con un valor nuevo, actualiza el libro con identificador 48 al nuevo precio. No cambia los demás campos.
- **DELETE.** *Lo utilizaremos para eliminar el recurso identificado. Es indistinto con respecto al protocolo si la eliminación será física o lógica, en términos semánticos representa una baja en ambos casos*. **Por ejemplo**, una solicitud DELETE a la URI /libros/34 daría de baja el libro con identificador 34. Si efectivamente es eliminado de la base de datos o si simplemente será marcado con una fecha y hora de baja o un campo que identifique la baja, no es parte del protocolo HTTP, sino que dependerá de la solución.

Existen otros métodos HTTP que se utilizan, pero los cinco mencionados son los que nos resultarán de utilidad a lo largo de la materia y la mayoría de los proyectos de desarrollo web.

- **Parámetros.** En el paquete de datos de la solicitud, se identifican algunos parámetros generales de dicho mensaje. Por ejemplo, la versión de HTTP que se está usando, conjuntos de caracteres o codificación (US-ASCII, ISO-8859-1, etcétera), tipo de contenido (imágenes, HTML, etcétera) y configuración de idioma.
- **Encabezados.** Tanto en la solicitud como en la respuesta, se comunican al servidor o al cliente datos sobre el mensaje, sobre el cliente o sobre el servidor. Controles de caché, tipo de codificación del mensaje, tipos de contenido y codificación esperados en la respuesta,

información sobre cookies, autenticación —temas que abordaremos más adelante—, instrucciones para reintentos en respuestas, son solo algunos ejemplos.

- **Códigos de estado o respuesta (status/response code).** Las respuestas HTTP del servidor contienen un número de 3 dígitos, entero, cuyo primer dígito indica el tipo de respuesta.

Más adelante, veremos **cómo abordar los diferentes códigos de respuesta para acompañarlos de un mensaje orientativo en el cuerpo** (por ejemplo, en el archivo html) que sea claro para el usuario.
Aquí las cinco clases existentes y algunos ejemplos de los más utilizados:

- **1xx.** Los códigos de respuesta que comienzan con 1 indican que la solicitud llegó al servidor y el proceso continúa. No es usual que lo usemos adrede en desarrollo web.
- **2xx.** Indican una respuesta exitosa. La acción solicitada fue recibida, entendida, aceptada, ejecutada. Por ejemplo, 200 indica éxito a secas, 201 indica éxito y nuevo recurso creado.
- **3xx.** Indican alguna forma de redirección. 302 indica una redirección temporal; por ejemplo, cuando tenemos la página en mantenimiento, cuando el cliente quiere ingresar, lo redireccionamos a la vista amable que indica que pronto estará disponible nuevamente. 301, por su lado, indica una redirección permanente.
- **4xx.** Indican un error en la solicitud del cliente. Por ejemplo, 404, el más conocido, indica que se solicitó un recurso no encontrado; 401, que necesitamos iniciar una sesión (con usuario y contraseña, por ejemplo); 403, que no podemos ingresar por permisos.
- **5xx.** Indican un error en el servidor. El más usual es 500, que representa un error interno del servidor.

HTTPS

Seguramente, habrás observado que, al navegar muchas páginas webs, en el navegador, junto con la URL, hay un candado. Esto sucede cuando «la navegación es segura». Pero, ¿cómo sabe el navegador eso? Porque se está utilizando un protocolo levemente diferente al que vimos: HTTPS. La S que agregamos al final hace referencia a que es seguro (secure, en inglés).

La diferencia entre ambos protocolos no está en ninguno de los ítems que ya vimos, **sino en la forma en que viajan los datos a través de la red, en la capa de transporte.** Si los datos son

transferidos mediante *HTTP*, **estos viajan de forma plana y son accesibles para cualquiera que intercepte la comunicación.** En cambio, *HTTPS* incorpora el protocolo seguridad en la capa de transporte (TLS, por las siglas en inglés de transport layer security) y, mediante un cifrado SSL (secure sockets layer) a los paquetes, **los datos viajan de un modo seguro de un lugar a otro.**

Para habilitar el protocolo HTTPS en un sitio web propio, debemos obtener un certificado de seguridad, que es emitido por una autoridad de certificación (CA). Las CA son las *encargadas* de verificar que realmente una dirección web pertenece a quien dice pertenecer.

Martorell y Gutiérrez (2006) detallan el procedimiento para una comunicación segura con SSL:

> Para establecer una comunicación segura utilizando SSL, se tienen que seguir una serie de pasos. Primero, se debe hacer una solicitud de seguridad. Después de haberla hecho, se deben establecer los parámetros que se utilizarán para SSL. Esta parte se conoce como SSL handshake. Una vez que se haya establecido una comunicación segura, se deben hacer verificaciones periódicas para garantizar que la comunicación siga siendo segura a medida que se transmitan datos. Luego que la transacción ha sido completada, se termina SSL. (https://lc.cx/6H22_7)

De acuerdo con Google, las ventajas que nos brinda utilizar HTTPS son las siguientes:

1. **Cifrado.** Se cifran los datos intercambiados para mantenerlos a salvo de miradas indiscretas. Eso significa que, cuando un usuario está navegando por un sitio web, nadie puede «*escuchar*» sus conversaciones, hacer un seguimiento de sus actividades por las diferentes páginas ni robarle información.
2. **Integridad de los datos.** Los datos no pueden modificarse ni dañarse durante las transferencias, ni de forma intencionada ni de otros modos, sin que esto se detecte.
3. **Autenticación.** Demuestra que los usuarios se comunican con el sitio web previsto. Proporciona protección frente a los ataques *man-in-the-middle* y contribuye a la confianza de los usuarios, lo que se traduce en otros beneficios empresariales. (citado en IntegraciónTIC, s.f., https://lc.cx/L6ajer)

Google y otras empresas referentes en el mundo de los sitios web recomiendan fervientemente el uso de HTTPS en las páginas, y se estima que mejora el posicionamiento de las páginas al tenerlo activado. En las próximas lecturas, veremos algunas recomendaciones que debemos tener presente si nos interesa tener un sitio

web público al que queramos atraer mayor audiencia, ya que el posicionamiento en los resultados de búsqueda lo es todo en ese caso. Las técnicas y recomendaciones para lograrlo se denominan SEO, por las siglas en inglés de search engine optimization (optimización de buscadores).

Referencias

Cañedo Andalia, R. (2004). *Aproximaciones para una historia de Internet* (pp. 1-24). ACIMED, 12(1). http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352004000100005

IntegraciónTIC, (s.f.). *Proteger sitios web con el protocolo HTTPS*. https://www.integraciontic.com/google-marca-sitio-web-no-seguro-si-no-posees-https/

Martorell, S. O. y Gutiérrez, L. C. (2006). *Protocolo de seguridad SSL. Ingeniería Industrial* 27(3). https://www.redalyc.org/articulo.oa?id=360433561012