Capítulo 1: ¿Qué es el HTTP?

HTTP- Hypertext Transfer Protocol, se usa para gestionar los requerimientos (cliente) y las respuestas (servidor) por internet.

Para ello se requiere de una transferencia de datos entre ubicaciones específicas de la red.

Esta transferencia de datos se produce gracias a la intervención del TCP (Transmission Control Protocol). Este se encarga de administrar los canales entre el navegador (browser) y el servidor.

Para que el cambio de información sea posible se han de establecer un cierto orden, normas. Ejemplo, los dispositivos por los que se quiera enviar o transmitir datos han de tener ambos HTTP.

Petición GET/HTTP GET - son una especie de método por el que el cliente puede llamar/solicitar.

Proceso - Petición GET:

1. El cliente solicita al browser acceder a una página/recursos.
2. Al digitalizar la dirección se extrae el "http" ya que se reconoce el protocolo de red, dando lugar al dominio URL.
3. Seguido a esto se le solicita al Servidor DNS conocer la dirección IP (Internet Protocolo).
4. Dado el protocolo, se hace un protocolo "HTTP/1.0".
5. Después se realiza una revisión del protocolo "HTTP/1.1".
6. Una vez responde a la navegación, en la primera línea del encabezado ponemos "HTTP/1.1 200 OK" como confirmación de que el servidor comprende el protocolo con el que el cliente desea comunicarse.

---

7. En el caso de que el servidor no lograra localizar el camino solicitado por el cliente, es decir, en el momento en que el servidor identifica el protocolo HTTP, pero no logra encontrar el contenido, se nos muestra el código Status, "404 NOT FOUND". Esto sucede debido a que el contenido de la URL fue movido/redirigido, o simplemente, esta fue digitada de forma incorrecta.

Capítulo 2: ¿Qué es el HTTPS?

HTTP es un requerimiento que puede ser leído (documentos, textos) tanto por humanos como por computadoras, aunque no es seguro que en estos documentos haya información sensible, confidencial o personal (tarjeta de crédito, por ejemplo).

Para dicho problema surge HTTPS o HTTP Secure, este es igual al HTTP, pero con un nivel adicional de seguridad, pues permite/tiene la capacidad de cifrar datos.

Para acceder a estos nuevos niveles de seguridad, HTTPS utiliza llaves públicas y privadas. Para generar estas llaves, el poseedor debe de acreditar y tener confirmada su identidad.

El funcionamiento es el siguiente:

- Los navegadores (clientes/usuarios) una vez acceden a la página se les proporciona una llave pública, ésta cifra la información que el usuario envía al servidor (ejemplo, registro en una tienda online: correo, contraseña, tarjeta de crédito, dirección…), mientras que el servidor es quién utiliza la llave privada para descifrar la información cifrada por la llave pública.

Este funcionamiento basado en 2 llaves distintas se le conoce como cifrado asimétrico. Y cabe destacar su debilidad, el bajo rendimiento. Como alternativa a este problema surge el cifrado simétrico, este utiliza una única llave para cifrar y descifrar los datos.

En comparación con el cifrado asimétrico, el cifrado simétrico es mucho más rápido, pero, a su vez, menos seguro.

Por tanto, para alcanzar el mayor punto/equilibrio entre rendimiento y seguridad, el HTTPS decide utilizar ambos métodos.

El funcionamiento es el siguiente:

- El inicio es igual al sistema/cifrado asimétrico, con la diferencia de que una vez a recibido el servidor la información cifrada con la llave pública del usuario. HTTPS realiza una conversión ← (???) entre el cifrado asimétrico y simétrico. De esta forma el cliente genera una llave simétrica, que es enviada al servidor utilizando el cifrado asimétrico. Una vez el servidor descifra la llave tanto el cliente como el servidor comienzan a usar y comunicarse con la misma llave de forma simétrica.

  En otras palabras, el usuario envía de forma asimétrica, es decir, privada, una llave pública de mi información al servidor, de esta forma, una vez la descifra, ambos funcionamos de forma simétrica.

Cabe resaltar que para que una empresa utilice el formato HTTPS debe solicitar un certificado de una Autoridad Certificadora, garantizando la identidad de la URL y validando dicho certificado.
Dentro de este contexto (HTTPS), un certificado digital es aquel que prueba que nosotros como usuarios podemos acceder a la información sobre el dominio de un servidor, así como también la llave pública y la expiración/caducidad de la validez del certificado.

Capítulo 3: Métodos (Verbos) HTTP

Un verbo HTTP es como se le conoce a la serie de métodos/acciones aplicados a las peticiones de los clientes, así como las respuestas del servidor. Por tanto, una solicitud HTTP o verbo HTTP son las acciones iniciadas por el cliente y finalizadas por el servidor como respuesta.

Hay 9 métodos de solicitud HTTP:

- Los principales/más utilizados: GET, PUT, POST y DELETE
- Los secundarios: HEAD, CONNECT, OPTIONS, TRACE y PATCH.

- GET: Este es el método de solicitud más común. Consiste en solicitar al servidor un recurso específico.

- PUT: Este método es usado como una herramienta para alterar el estado vigente en el servidor. Se usa para añadir algo al servidor. Por ejemplo, cuando modificamos o añadimos nuevos datos en nuestro registro.

- POST: Es un método utilizado con el fin de alterar los datos del servidor. Se emplea en la sustitución de los datos del recurso del destino por los datos del requerimiento. Aunque en ocasiones también se usa para crear recursos.

  Tanto PUT como Post pueden parecer lo mismo, pues se encargan de crear, añadir y actualizar datos en el servidor. Pero, la diferencia radica en que PUT es idempotente, mientras que POST es no idempotente. Por ejemplo, si accidentalmente llamamos al POST varias veces para crear un pedido de compra, varios registros idénticos serán creados, al contrario del PUT.

  (Idempotente en el campo de la informática es utilizado para describir una operación que produce los mismos resultados si se ejecuta una o varias veces, es decir, conduce al mismo resultado, no importa cuantas veces se ejecute.)

- DELETE: Este método se encarga de excluir, borrar o remover un recurso (dato????) en un servidor.

- HEAD: Este método es el responsable de solicitar una respuesta por parte del servidor. La petición realizada por HEAD hacia el servidor requiere de una respuesta idéntica a la de la solicitud GET, destacando en que es más rápida, debido a que se transfieren pequeñas cantidades de datos.

- CONNECT: Método que establece un puente/conexión con el servidor. Esta comunicación es bidireccional entre el cliente y el recurso solicitado.

- OPTIONS: Este método se usa para describir las opciones de comunicación con el recurso de destino.

- TRACE: Este es el método por el cual un test denominado loopback se ejecuta (envío de un mensaje a lo largo del camino hacia el recurso destinado).

- PATCH: Este es un método que aplica modificaciones parciales en un recurso (sin la necesidad de alterar los datos completamente).

Códigos de Devolución del HTTP/HTTPS

Un verbo HTTP es como se le conoce a la serie de métodos/acciones aplicados a las peticiones de los clientes, así como las respuestas del servidor. Por tanto, una solicitud HTTP o verbo HTTP son las acciones iniciadas por el cliente y finalizadas por el servidor como respuesta.

Capítulo 4: Códigos de Devolución del HTTP/HTTPS

El protocolo HTTP funciona mediante códigos estándar, estos códigos (de 3 dígitos) sirven para informar al browser sobre el estado o situación de la respuesta. Los códigos están divididos en 5 diferentes clases:

1. Información (Informational Response). Este código indica que la solicitud del cliente fue recibida y comprendida por el servidor.
2. Éxito (Success). La solicitud del cliente fue recibida, aceptada y procesada con éxito por el servidor.
3. Redireccionamiento (Redirection). Informa que el recurso solicitado por el cliente no existe donde el cliente envió la solicitud original.
4. Error del Cliente (Client Error). Estos códigos son enviados cuando una solicitud realizada por el cliente no es correcta y el servidor no logra comprender.
5. Error del Servidor (Server Error). Enviados cuando la solicitud realizada por el cliente es válida, pero el servidor encontró algún error al procesar la solicitud.

Capítulo 5: Parámetros de Requerimientos del HTTP/HTTPS

Otra función que tienen los parámetros HTTP es la transferencia/envío de parámetros vía URL. Ejemplo, al buscar HTTP en Google, la URL que obtendremos de esta petición será algo similar a: https://www.google.com/search?q=http. Aquí vemos como Google altera la propia URL añadiendo una serie de parámetros e informaciones. En este caso “/search” indica la dirección de Google en la web; mientras que, la “q”, seguido del “=” indican el nombre del parámetro y “HTTP” el valor de este.

Cabe destacar que existe cierta diferencia entre la transmisión de parámetros en los métodos más utilizados, el GET y el POST.

El GET tiene una transmisión vía URL; mientras que, el POST los parámetros siguen en el cuerpo del requerimiento, existiendo cierta diferencia en términos de seguridad entre ambos.

Capítulo 6: Algunos Conceptos Importantes Más

- Dominio:

  Un dominio es la dirección electrónica de un servidor, el nombre de este por el que será encontrado dentro de internet.

  El Registro de Dominio es el servicio encargado de administrar estos nombres (dominios), esto es, encargarse del registro y la renovación de las direcciones electrónicas.

  Por lo general, un dominio se divide en 4 partes:

  - "https" es el protocolo de comunicación.
  - "www" World Wide Web es un término opcional. Su uso no es necesario. Se utiliza para hacer ver al usuario que el sistema funciona a través de internet y es de alcance mundial.
  - ".com" Identifica la naturaleza del site. Se entiende como un complemento al nombre inicial del dominio.
  - ".br" Identifica el país de origen del servidor

- Subdominio:

  Un subdominio representa aquellas páginas/secciones específicas dentro del host, por ejemplo, de Drive (drive.google.com).

- Recursos:

  Un recurso (Resource) es el fin/objetivo de una petición HTTP/HTTPS. Por ejemplo, en la URL https://developer.mozilla.org/pt-BR/docs/Learn/, el objetivo que buscamos es el recurso "pt-BR/docs/Learn/"

- Dirección IP:

  La dirección IP (en inglés Internet Protocol) identifica de forma única un servidor, una red o un dispositivo en internet, ejemplo, una impresora. Una dirección IP es un número de 32 bits.

- DNS:

  Un servidor DNS (Domain Name System) es un servicio que traduce un nombre del site (URL) en una dirección IP. El DNS es un elemento importante de la infraestructura de la web, pues sirve como una lista telefónica de internet, banco de datos de dominios.

- PUERTAS:

  Según los procedimientos ya dados, una vez se ha establecido una conexión en la red entre la petición del cliente y el servidor (dirección IP), se accede a dicho servidor. Sin embargo, no basta solo con el protocolo HTTP y la dirección IP; sino que requerimos saber mediante qué conexión realizar el acceso. En este caso, por qué puerta debemos conectar con el servidor.

Ya hemos mencionado en varias ocasiones las URL o Uniform Resource Locator, estas son las direcciones web.

Sin embargo, también existen las URN o Uniform Resource Name, una URN se encarga de definir un recurso de la web/internet, de esta manera identifica de forma única y persistente identifica de forma única y persistente un recurso, independiente de su localización, permitiendo la separación entre la identificación (nombre único) y la localización (URL) de un determinado recurso.
A su vez, también existe la URI o Uniform Resource Identifier, en español, Identificador Uniforme de Recursos, es una “string”, secuencia de carácteres, que refiere a un recurso.

Por tanto, una URI es similar a una URL, ambas definen direcciones web. Pero, la diferencia es que no todas las URI son URL, pues estas solo pueden definir un único recurso, y no la dirección completa como si es el caso de la URL.

Capítulo 7: Modelo Requerimiento/Petición-Respuesta (Request-Response)

Ahora que entendemos como es el funcionamiento HTTP y HTTPS, tenemos que tener claro que la comunicación entre un cliente y un servidor es siempre iniciada por un pedido (requerimiento/petición) por parte del cliente, posteriormente entrará en acción el servidor, quién aportará a la comunicación una respuesta, ya sea con una solicitud atendida o con un código de devolución de error (ejemplo, error 404).

Todo pedido realizado por un cliente está formado por un nuevo requerimiento. Esto es que al cliquear en algún link dentro de la página, un nuevo requerimiento es realizado y una nueva respuesta es emitida por el servidor.

Este requerimiento se compone principalmente de 3 elementos:

- Método: es el comando general enviado al servidor.
- Encabezado: Además del “encabezado” método, existen encabezados HTTP que contienen informaciones más específicas sobre la solicitud (tanto obligatorios como opcionales).

    1. Host: Es el URL hacia el cual la solicitud es hecha.
    2. User-Agent: Informa al servidor qué navegador y sistema operativo se está utilizando, así como el dispositivo.
    3. Cookie: Usado para almacenar informaciones, como las de login, permitiendo que dichos datos no necesiten ser repetidos en los requerimientos siguientes.
    4. Referer: Informa al servidor la dirección de la página web anterior.

- Cuerpo: Es utilizado para métodos en donde nuevas informaciones son enviadas al servidor, por ejemplo POST. El cuerpo es el elemento en el cual dichos datos son transmitidos.. Aunque, no todos los métodos requieren de un cuerpo, por ejemplo, GET, DELETE, HEAD.

- Sesiones y Cookies:

    El “login” o registro de sesión, así como las cookies son acciones/condiciones que precisan ser aceptadas por el cliente para acceder al servidor y sus funciones. Por lo general, los servidores emplean un sistema en el que una vez ya se ha registrado el cliente, una vez tenga que volver a entrar, este pasó podrá ser obviado.

    Esto es así, ya que al realizar el login el site crea un número aleatorio, difícil de adivinar y como identificación única (una ID, mismamente). Gracias a este nuevo número identificatorio, se sustituyen el login y la contraseña por dicho número.

    Y donde se almacenan estos números, en las cookies. Las cookies son datos que el servidor envía al cliente.

Capítulo 8: Web Services

El término Web Services es utilizado para referirse a las aplicaciones que proporcionan servicios (datos) a otras. Podemos definir las WS como un subconjunto de APIs. Aunque este último no posee tantas limitaciones como si las tienen las Web Services.

Una API (Application Programming Interface) es una interfaz de programación de aplicaciones que posee la documentación necesaria para su uso, permitiendo una comunicación eficiente entre sistemas.

Las Web Services hacen uso principalmente del protocolo HTTP. Esto se puede intuir de manera natural, pues se sabe que las WS no solo sirven para devolver datos, sino que, también se pueden usar para alterar datos dentro del servidor.

En la mayoría de los casos GET se usará para recuperar informaciones sin la posibilidad a modificaciones en las mismas, sin efecto colateral del lado del servidor. POST se utilizará para agregar informaciones. PUT para actualizar/modificar y, eventualmente agregar informaciones. Y, DELETE para excluir informaciones.

Por tanto, a partir de una petición, una WS devuelve, normalmente, un conjunto de datos (por ejemplo: texto, imágenes, audio, vídeo, etc.). Una vez se devuelven estos datos al cliente, se suele utilizar un formato estándar a fines de que este entienda y pueda hacer uso de los mismos. Esta devolución estandarizada de datos es muy común mediante los siguientes lenguajes de marcas (LND):

1. JSON (JavaScript Object Notation)
2. XML (eXtensible Markup Language)
3. HTML (HyperText Markup Language)

Al realizar una petición, informamos de que requerimos la respuesta del servidor a través del encabezado HTTP, por ejemplo.

Cabe destacar que una WS no solo sirve para devolver datos, sino que, también se puede usar para alterar datos dentro del servidor.

Por último, podemos deducir que estas informaciones, anteriormente mencionadas, son sinónimos de los recursos. Esto quiere decir, que en realidad los métodos son recursos del lado del servidor. Siendo estos recursos definidos por vía URI.

De esta manera vemos cómo de repente una Web Service se compone de 3 componentes básicos: Recursos (URI), Operaciones (Métodos HTTP) y Datos (XML, JSON…).

- XML vs JSON:

  Respecto a JSON, los datos son mapeados en pares de valores-llave, tornándose a un uso bastante intuitivo y legible. Aquí la llave representa el nombre y formato de los datos, mientras que, el valor se representa por sí mismo (datos). Como desventaja se encuentra su ventaja, la simplicidad y facilidad en el uso, ya que no es muy complicado agregar metadatos en archivos JSON, volviendo complejo obtener mayores informaciones sobre los datos.

  En cambio, XML, se puede entender básicamente como una extensión actualizada y mejorada en cierta medida de HTML, generalmente se utiliza en la transmisión de datos, dado que permite crear elementos personalizados.

  El uso de XML dada una mayor complejidad respecto a JSON permite definir fácilmente la estructura de grandes volúmenes de datos/metadatos. Sin embargo, esto tiene su desventaja, pues es

menos legible por los humanos, pero altamente legible para computadoras. Otro punto es que cuanta mayor cantidad de volúmenes acepte, mayor será el esfuerzo de desarrollo y su “peso”.

Capítulo 9: El Estándar REST

Antes pudimos ver cuáles eran los 3 componentes que conforman una Web Service. A esto se le conoce como la base de un estándar de arquitectura para interface (comunicación) entre aplicaciones que denominamos REST (Representational State Transfer) o API REST.

A su vez, aquellos recursos son representados por URIs y ubicados en el estándar RESET. En estos sistemas RESET las URIs, sólo pueden contener sustantivos, y no sustantivos-verbos.

Por último los datos, estos no son más que la definición del formato de datos que optamos por usar en determinada Web Service.

Capítulo 10: HTTP/2

En los capítulos anteriores hemos estado tratando la versión HTTP 1.1, propia de los años 90 's, sin embargo, la informática que hoy en día conocemos no es la misma que la de hace 20-30 años.

Antes, WEB 1 se caracterizaba por su estatismo/estaticidad. No existía la interacción directa entre cliente y servidor, sólo se podía utilizar para lectura (usuario pasivo).

No obstante, con WEB 2, las cosas cambian, pues pasamos a tener interactividad, los servidores pasan de tener un carácter pasivo y estático a uno activo y dinámico (usuario activo). A pesar de que los sites siguen hospedados en servidores, comienzan a surgir aplicaciones móviles como resultado de la llegada de los smartphones y aplicativos móviles.

Sin embargo, actualmente nos encontramos en WEB 3, teniendo como principal característica la descentralización, es decir, el cliente pasa a actuar también como servidor.

Tras este último cambio, como respuesta a la negativa de quedar obsoleto, el protocolo HTTP evoluciona a su segunda versión, a HTTP/2.

La llegada del HTTP/2 trae consigo bastantes mejoras/ventajas respecto a su antigua versión: optimización del rendimiento, mayor y mejor seguridad y simplificación en el desarrollo de las aplicaciones.

Uno de los problemas que hubo en el curso de la actualización del protocolo tuvo que ver con las aplicaciones web, pues a mayor cantidad de datos a transmitir, mayor requerimiento de trabajo y mayor posibilidad de problemas durante el rendimiento.

Para ello HTTP 1.1 habilitó el uso de GZIP, un software libre que reemplaza al programa UNIX. GZIP se encarga de compactar los datos de las respuestas. Dicha acción, primero deberá de estar habilitada de forma explícita y no es automática para su correcto funcionamiento.

En cambio, tras la actualización, HTTP/2 emplea GZIP de forma habilitada por estándar y obligatoria en su uso de manera predeterminada.

Una diferencia de esto se aprecia en la compresión de datos. En HTTP 1.1 solo se comprime el cuerpo de la respuesta, el resto sigue transitando de forma textual, sin compresión. Por su parte, HTTP/2, por estándar, los encabezados son binarios, y utilizan el algoritmo HPACK para compactar, reduciendo el volumen de datos transitados en los encabezados.

Respecto a la seguridad, HTTP/2, el uso de TLS pasa a ser estándar y obligatorio, campo que no era considerado como “necesario” en HTTP 1.1.

Por último, otro cambio que hizo notarse fue el estado de los encabezados. En el HTTP 1.1 estos no guardaban la información de las peticiones anteriores (stateless): esto implicaba inminentemente que en toda petición el encabezado debería ser reenviado. Esto no ocurre en el HTTP/2, pues los encabezados son statefull.

- SERVER-PUSH:

  El término Server-Push se utiliza para referirse al momento en que al solicitar una página (utilizando HTTP/2), el servidor, además de devolver el contenido inicial, “entiende” qué recursos adicionales serán necesarios y los envía de inmediato. Proceso que no ocurre de esta forma vía HTTP 1.1. El objetivo de hacer esta conversión anticipada de datos por parte de la página sin que todavía lo haya requerido el cliente, tiene su truco, pues, este envío de información ya sabe el servidor que se le pedirá a continuación, de esta manera, se optimiza el tráfico de datos.

En resumen, el HTTP 1.1 se ocupa de los requerimientos en serie; en cambio el HTTP/2, de los paralelos.

- KEEP-ALIVE:

  En informática, el concepto Keep-Alive hace referencia al canal/túnel (efectuado mediante TCP, protocolo de transporte usado en el HTTP 1.1) por el que la comunicación cliente-servidor se efectúa (petición-respuesta), estos canales resaltan sobre el resto, debido a que se dejan abiertos durante un mayor periodo, con el fin de no estar abriendo y cerrando un nuevo canal una vez finalizada la respuesta. Cabe mencionar que, este recurso de túnel abierto a pesar de ser un sistema bastante rápido en términos informáticos, su desventaja es que se demora mucho en ser asignado.

  Los browsers actuales tienen la capacidad de mantener entre 4 y 8 conexiones simultáneas por dominio.

  De igual forma, el mecanismo o proceso Keep-Alive también es implementado en el HTTP/2, pero con una diferencia: el cliente puede realizar una serie de requerimientos antes de recibir la respuesta del primer requerimiento. La verdadera diferencia entre el HTTP 1.1 y el HTTP/2 respecto a este a Keep-Alive es que en el HTTP 1.1 la comunicación se realiza de forma sincrónica, mientras que en el HTTP/2 de forma asincrónica.

  En HTTP2 este concepto también se le conoce como multiplexing.

Capítulo 11: Roadmap del Desarrollador Web

Antes de empezar con este capítulo se insiste y recomienda de manera asertiva que la persona que quiera aprender acerca de esto controle, aprenda, hable con fluidez o domine el idioma inglés; o, al menos que tenga un mínimo de control en la lectura y escritura. Esto es debido a qué el inglés es la lengua materna en la ciencia computacional, en la comunidad TI (Tecnología de la Información).

- Cultive Buenos Hábitos:

  En informática, el concepto Keep-Alive hace referencia al canal/túnel (efectuado mediante TCP, protocolo de transporte usado en el HTTP 1.1) por el que la comunicación cliente-servidor se efectúa (petición-respuesta), estos canales resaltan sobre el resto, debido a que se dejan abiertos durante un mayor periodo, con el fin de no estar abriendo y cerrando un nuevo canal una vez finalizada la respuesta. Cabe mencionar que, este recurso de túnel abierto a pesar de ser un sistema bastante rápido en términos informáticos, su desventaja es que se demora mucho en ser asignado.

En este capítulo final el autor nos especifica una serie de conceptos y herramientas que nos ayudarán en la toma del dominio de hábitos a fines de dedicarnos o aprender a ser un Desarrollador Web. Este libro no busca tener una intención distinta a la de informar y educar sobre los aspectos más básicos de este campo.

El autor nos sugiere a empezar a tener unos mejores hábitos:

1. ¡Menos Netflix, más vídeos técnicos!
2. ¡Menos consumo, más estudio!
3. ¡Menos gastos, más inversiones!
4. ¡Menos hablar, más hacer!
5. ¡Menos reclamos, más gratitud!
6. ¡Menos zona de confort, más desafíos!

Nota: No es necesario levantarse a las 5 de la mañana, bañarse con agua fría, ni ningún hábito de aspecto radical.

- Camino Full-Stack:

  Para llegar a ese estado de validación como profesional de TI, convertirse en un desarrollador web full-stack.

  Para hacer esta idea lo menos tediosa y abrumadora, se dividirá en 3 grupos:

1. Front-end: Parte cliente, lo que el usuario ve e interactúa. Interfaz Gráfica.
2. Back-end: Lado servidor, lo que mantiene la aplicación: Lógica de la Aplicación, Procesos y Banco de Datos.
3. Full-stack: Front-end + Back-end.

  Por tanto, para una captación full-stack se sugiere lo siguiente:

ConcePtos Básicos:

1. ¿Qué es Desarrollo Web?
2. ¿Cómo funciona la Web? (Aquí se ajusta este libro)

FRONT-END BÁSICO: (interfaz básica)

- HTML

- CSS
- JavaScript

FRONT-END AVANZADO: (interfaz avanzada)

- Bootstrap (y/o Tailwind)
- React (y/o Angular y/o Vue y/o Next.js)
- Git y Github
- SCSS

BACK-END PROGRAMACIÓN[2]: (Desarrollo servidor)

- PHP + Laravel y/o
- Python + Django y/o
- Java + Spring/Play/Hibernate

BACK-END BANCO DE DATOS: (Desarrollo en banco de datos)

- Relacionales
    - mySQL y/o
    - PostgreSQL y/o
    - SQLite
- No relacionales
    - MongoDB

Se recomienda encarecidamente y repetidamente que ante cualquier duda o sugerencia sobre herramientas o cualquier otro dilema acerca del tema, se busque información en Google,o, contactar con el autor del libro.