Introducción al desarrollo web

Este documento describe los fundamentos del desarrollo de aplicaciones web. Comenzaremos viendo la arquitectura cliente/servidor, sus componentes y la relación entre ellos en el contexto de las aplicaciones web. Después se detallará el funcionamiento de la generación dinámica de contenido web para finalizar haciendo un repaso a los lenguajes de programación y frameworks empleados en el desarrollo de aplicaciones web.

Tabla de contenido

Introducción al desarrollo web

1 Introducción

El desarrollo web es una disciplina dentro del desarrollo de aplicaciones que se encarga del diseño y construcción de aplicaciones software que se ejecutan sobre un sistema distribuido con estructura cliente-servidor, también conocidas como aplicaciones web.

La especial naturaleza de este tipo de aplicaciones complica su diseño y desarrollo, por lo que desde su aparición han surgido tecnologías enfocadas a facilitar y flexibilizar la construcción de las mismas.

Si bien en las aplicaciones de escritorio hay que tener en cuenta aspectos como el diseño funcional, interfaz de usuario, plataforma de ejecución, etc. en las aplicaciones web intervienen una serie de elementos que complican su diseño y desarrollo al obligar a ser conscientes sobre aspectos implicados como sistemas multiplataforma, protocolo de comunicaciones, ...

En los siguientes epígrafes veremos que las aplicaciones web se ajustan a un modelo de arquitectura basada en las peticiones de clientes atendidas por un servidor. Veremos su estructura, con una descripción de cada componente y la relación entre ellos, además de sus características.

Estas aplicaciones web se basan en la generación dinámica de contenido web de diferente tipo que se realiza en un servidor web y se devuelve a un cliente, generalmente un navegador web, como respuestas a una petición previa. Veremos pues los principios en los que se basa el desarrollo web centrándonos en el lado del servidor, con los lenguajes que dan soporte a esta programación, sus características, cómo se integran con los lenguajes de

marcas, en los propios servidores web y las herramientas necesarias para su programación.

2 Arquitectura cliente-servidor

La arquitectura cliente-servidor es un modelo de sistema informático distribuido compuesto por tres elementos principales: el cliente, el servidor y la infraestructura de red.

- ✔ El servidor ➜ Cuando hablamos de una forma genérica, si mencionamos a un servidor, nos referimos a un ordenador, normalmente con prestaciones hardware elevadas, que ejecuta servicios para atender las demandas de diferentes clientes. Sin embargo, bajo el punto de vista de la arquitectura cliente/servidor, un servidor es una aplicación informática que ofrece un conjunto de servicios o recursos disponibles a los clientes que lo solicitan. Es muy frecuente que, para referirse a un proceso servidor, se utilice el término back-end.

- ✔ Cliente ➜ Igual que antes, al hablar de forma genérica sobre un cliente, nos referimos a un ordenador, normalmente con prestaciones hardware limitadas, que requiere y utiliza los servicios de un equipo servidor. Sin embargo, bajo el punto de vista de la arquitectura cliente/servidor, un cliente es una aplicación que solicita los servicios de un servidor, normalmente a petición de un usuario. En entornos cliente/servidor, suele utilizarse el término front-end para referirse a un proceso cliente. Normalmente, un proceso cliente se encarga de interactuar con el usuario, por lo que estará construido con alguna herramienta que permita implementar interfaces gráficas (GUI). Además, se encargará de formular las solicitudes al servidor y recibir su respuesta, por lo que deberá encargarse de una parte de la lógica de la aplicación y de realizar algunas validaciones de forma local.

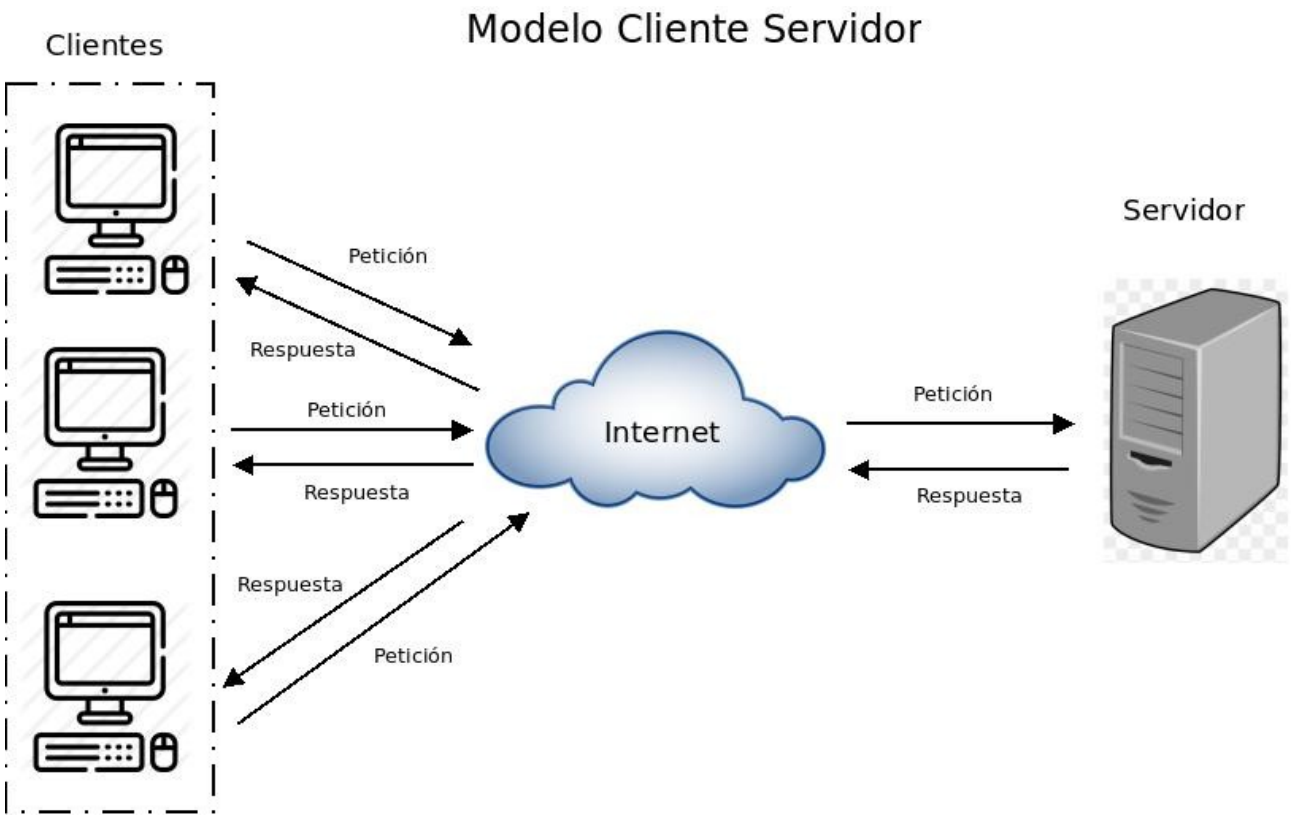
- ✔ Infraestructura de red (intranet o Internet) ➜ Encargada de conectar a los servidores con los clientes. Además del hardware de red, también se necesita un software de red que regule la comunicación. Es la parte del software del sistema que se encarga del transporte de los mensajes entre el cliente y el servidor, por lo que se ejecuta en ambos lados de la estructura. También conocida como middleware, permite independizar a los clientes y a los servidores, sobre todo, gracias a los sistemas abiertos, que eliminan la necesidad de supeditarse a tecnologías propietarias. Podemos estructurar el middleware en tres niveles:

  - ➢ El protocolo de transporte, que será común para otras aplicaciones del sistema.
  - ➢ El sistema operativo de red.
  - ➢ El protocolo del servicio, que será específico del tipo de sistema cliente/servidor que estemos considerando.

Un sistema distribuido es un conjunto de elementos software que se ejecutan en varios nodos diferentes conectados en red para lograr un objetivo compartido común. Una aplicación web que sigue la arquitectura cliente/servidor se considera un sistema distribuido ya que parte de la aplicación se encuentra en los clientes y parte en el servidor (aunque podrían estar en la misma máquina) y se comunican únicamente por medio de la intranet o

de Internet con un conjunto de protocolos de red comunes.

El objetivo es dar servicio a los usuarios finales que pueden estar dispersos en un área geográfica más o menos extensa y acceder a un conjunto común de recursos compartidos. Además, el acceso debe ser transparente, es decir, el cliente puede desconocer la ubicación física del recurso que pretende utilizar y, preferiblemente, multiplataforma, es decir, independiente del sistema operativo, del software de aplicación e incluso del hardware.



El funcionamiento básico de una arquitectura cliente servidor es el siguiente:

1. Una aplicación cliente realiza peticiones a otra aplicación, la cual se ejecuta en un servidor.
2. La aplicación del servidor recibe la petición, la interpreta y construye una respuesta adecuada a la misma.
3. La respuesta elaborada por la aplicación del servidor se envía a la aplicación cliente que hizo la solicitud.

Esta idea también se puede aplicar a aplicaciones que se ejecutan sobre un único ordenador, aunque resulta mucho más conveniente múltiples clientes realizando peticiones a múltiples servidores a través de la red.

En esta arquitectura, el cliente y el servidor son aplicaciones diferentes, de tal forma que cada una puede ser desarrollada de forma independiente, dando como resultado dos aplicaciones separadas, las cuales pueden ser construidas con tecnologías diferentes, pero siempre empleando el mismo protocolo de comunicación para establecer conexiones entre

ambos.

La separación del cliente y el servidor radica en la centralización de la información y la división de tareas y responsabilidades entre ambos. Por una parte, el servidor será el componente que tendrá acceso a los datos y los proporciona a los clientes que los soliciten. De esta forma, se protege la información y la lógica detrás del procesamiento de los datos. Además, el servidor puede atender simultáneamente a varios clientes, por lo que suele ser un equipo con muchos recursos. Por otro lado, el cliente suele ser instalado en ordenadores con recursos limitados, ya que solo procesa la información recibida del servidor y la interfaz de usuario, simplemente actúa como un visor de los datos y delega las operaciones pesadas al servidor.

Quizás uno de los puntos más característicos de la arquitectura cliente/servidor es la centralización de los datos, pues el servidor recibe, procesa y almacena todos los datos provenientes de todos los clientes. Si bien los clientes por lo general solo se conectan a un solo servidor, existen variantes donde hay clientes que se conectan a múltiples servidores.

2.1 Características

Además de la transparencia y la independencia del hardware y del software, una implantación cliente/servidor debe tener las siguientes características:

- ✔ En la arquitectura cliente/servidor el remitente de una solicitud es conocido como cliente. Es quien inicia solicitudes o peticiones, tiene por tanto un papel activo en la comunicación. Posteriormente, espera y recibe las respuestas del servidor a las solicitudes emitidas. Por lo general, puede conectarse a varios servidores a la vez y normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.
- ✔ Al receptor de la solicitud enviada por el cliente se le conoce como servidor. Desempeña un papel pasivo en la comunicación, es decir, al iniciarse espera a que lleguen las solicitudes de los clientes. Tras la recepción de una solicitud, la procesa y luego envía la respuesta al cliente. Por lo general, acepta las conexiones de un gran número de clientes, aunque en ciertos casos el número máximo de peticiones por unidad de tiempo puede estar limitado para evitar el riesgo de saturación.
- ✔ El cliente y el servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- ✔ Las funciones de cliente y servidor pueden estar en plataformas separadas, o en la misma plataforma.
- ✔ Debe utilizar protocolos asimétricos, donde el servidor se limita a escuchar, en espera de que un cliente inicie una solicitud. Es decir, servidor y cliente tienen funciones diferentes.
- ✔ El servidor ofrecerá recursos, tanto lógicos (procesamiento de solicitud, información, etc.) como físicos (espacio de almacenamiento, bases de datos, impresoras, etc.) a una cantidad variable y diversa de clientes.

- ✔ El servidor ofrecerá también una serie de servicios, que serán usados por los clientes. Estos servicios estarán encapsulados, para ocultar a los clientes los detalles de su implementación, por ejemplo, aceptar el requerimiento de un cliente sobre una base de datos o formatear los datos obtenidos antes de transmitirlos al cliente.
- ✔ Se facilitará la integridad y el mantenimiento tanto de los datos como de los programas debido a que se encuentran centralizados en el servidor o servidores, los cuales son responsables de ello.
- ✔ Los sistemas estarán débilmente acoplados, ya que interactúan mediante el envío de mensajes.
- ✔ Se facilitará la escalabilidad, de manera que sea fácil añadir nuevos clientes a la infraestructura (escalabilidad horizontal) o aumentar la potencia del servidor o servidores, incrementando su número o su capacidad de cálculo (escalabilidad vertical). Entendemos por escalabilidad la capacidad de adaptación y respuesta de un sistema con respecto al rendimiento del mismo a medida que aumentan de forma significativa el número de usuarios del mismo.
- ✔ Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los clientes o de los servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.
- ✔ El acceso a los recursos oculta la complejidad de los diferentes tipos de formatos de datos y de los protocolos.

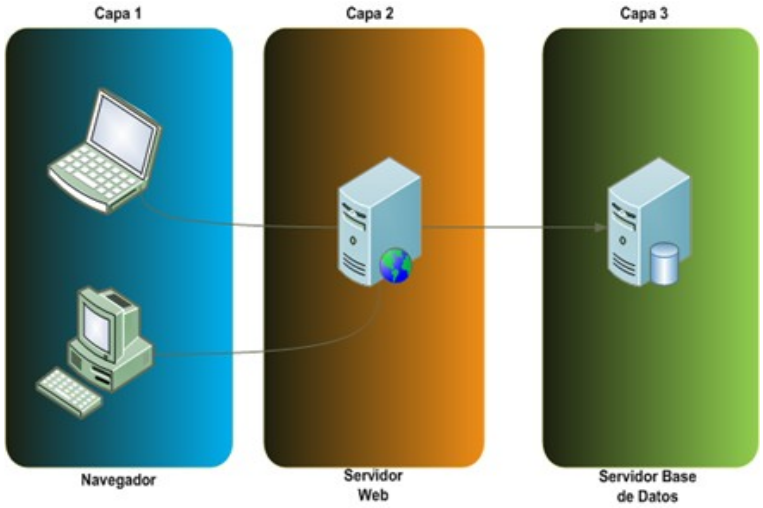
2.2 Estructura del modelo C/S

La mayoría de los servicios en Internet se basan en la arquitectura cliente-servidor. Por ejemplo, en el servicio WWW, los clientes (navegador web) solicitan que se les sirva una página web para visualizarla, aunque también es posible solicitar información si hablamos del caso de los servicios web. En ambos casos es el mismo escenario, donde un servidor web se encuentra permanentemente en ejecución a la espera de que los diferentes clientes realicen una solicitud. Cuando se recibe una, el servidor web devuelve una página web al cliente que hizo la solicitud para visualizarla en el navegador.

Normalmente a la solicitud que hacen los clientes al servidor se le llama petición (*request*) y a lo que el servidor devuelve a dicho cliente le llamamos respuesta (*request*).

Desde el punto de vista del desarrollo una aproximación más detallada para este modelo de ejecución es lo que se conoce como modelo en 3 capas. Es un modelo donde se muestra más en detalle como se distribuye los componentes software que forman una aplicación web. Sigue estando presente la arquitectura cliente-servidor (todo se basa en ella) pero aparecen más detalles como el software utilizado en cada uno de los dos actores y como interactúan las diferentes tecnologías o aplicaciones.

Para comprender mejor que es el modelo de desarrollo de 3 capas podemos echar un vistazo al siguiente esquema donde se muestra cada una de esas capas y de que se encarga cada una de ellas:



Las capas que componen este modelo son:

- ✔ Capa de presentación ➜ Es la capa más cercana y que interactúa con el usuario. Consiste en una GUI (*Graphical User Interface*, Interfaz Gráfica de Usuario). En el caso de una aplicación web sería el código HTML/CSS/Javascript que se carga directamente en el navegador web. Esta capa se ejecuta directamente en el equipo del cliente.

- ✔ Capa de negocio ➜ Es la capa intermedia donde se lleva a cabo toda la lógica de la aplicación. Siempre se ejecutará en el lado servidor. Esta capa, tras realizar todos los cálculos y/o operaciones sobre los datos, genera el código HTML que será presentado al usuario en la capa anterior.

- ✔ Capa de datos ➜ Es la capa que almacena los datos. Básicamente, en condiciones normales, hace referencia al propio SGBD que es el encargado de almacenar los datos. Dependiendo de la arquitectura de la aplicación, esta capa y la de negocio se pueden encontrar físicamente en el mismo host, aunque también es posible que se tengan que separar por cuestiones de rendimiento. La capa de datos sirve toda la información necesaria a la capa de negocio para llevar a cabo sus operaciones.

Por ejemplo, en una tienda online, la capa de datos almacena toda la información en una base de datos (usuarios, pedidos, artículos, ofertas, ...), la capa de negocio debe acceder a esa información y, tras procesar un pedido, por ejemplo, debe presentar el resultado final al usuario en el navegador, que es la capa de presentación.

Y si este modelo de tres capas se define desde el punto de vista del software y las tecnologías utilizados tendríamos lo siguiente:

- ✔ Aplicación cliente ➜ Es la aplicación que interactúa con el usuario y puede ser cualquier con capacidad para realizar peticiones a un servidor. Generalmente es un navegador web: Mozilla Firefox, Google Chrome, Safari, Opera, Microsoft Edge, ...
- ✔ Aplicación servidor ➜ Un servidor web (Apache, Nginx, IIS, ...) acompañado de su correspondiente módulo que permite ejecutar aplicaciones escritas en lenguaje de programación web (PHP, ASP, Python, ...) y que desarrolla la parte que ocupa la capa de negocio.
- ✔ SGBD ➜ Finalmente en la capa de datos podemos colocar cualquier SGBD, como pueden ser MySQL, Oracle, MariaDB, PostgreSQL, ...

En el ámbito del desarrollo web hay 3 perfiles de desarrollo web bien diferenciados en función del lado en que se ubican las tecnologías y para qué se utilizan éstas:

- ✔ Front-end ➜ Es la parte del desarrollo que se encarga del diseño y maquetación de la aplicación web utilizando tecnologías como HTML, CSS y Javascript (y sus frameworks). En este caso debe preocuparse también de la correcta presentación en cualquier tipo de dispositivo e incluso del posicionamiento en buscadores.
- ✔ Back-end ➜ Es la parte del desarrollo que se encarga de la programación de la lógica de negocio del lado servidor utilizando tecnologías como PHP, JSP y Python. También se encarga de la administración del servidor de aplicaciones y la base de datos.
- ✔ Full stack ➜ En un perfil que engloba los dos anteriores. En este caso el desarrollador quizás no es un experto de ninguna tecnología concreta pero tiene amplios conocimientos de todo el conjunto y es capaz de colaborar en cualquiera de las partes.

2.3 Tipos de arquitecturas cliente/servidor

Ahora que conocemos los fundamentos de la tecnología cliente/servidor, debemos analizar la interacción entre sus componentes para poder establecer cómo implementarla de la forma más adecuada.

En este sentido, deberemos realizar un análisis previo de los requerimientos en cuanto a los eventos que pueden producirse y las restricciones a las que se verá sometida la instalación, el tipo y volumen de información que va a procesarse, el tipo de bases de datos a utilizar y su tamaño (si son necesarias), la estimación sobre el tráfico de la red y el tiempo de respuesta, la ubicación física tanto de los datos que se van a manejar como de los procesos que la van a procesar, etc.

Para dar respuesta a estas situaciones, estableceremos dos tipos de clasificación diferentes: El primero atenderá al tamaño del lado servidor comparado con el tamaño del lado cliente. El segundo hará referencia al tipo de servicio que se ofrece.

2.3.1 Según el tamaño del lado cliente y del lado servidor

Una de las características del modelo cliente/servidor es que permite balancear la potencia de cálculo aplicada hacia el lado servidor o hacia el lado cliente, según convenga.

Por ejemplo, si el número de clientes fuese elevado, y la mayoría del proceso se realizará en el lado servidor, no necesitaríamos clientes muy potentes, pero probablemente necesitaríamos ampliar la potencia de cálculo del lado servidor y, como situación complementaria, tendríamos que valorar el aumento de tráfico en la red.

Por otro lado, con clientes más potentes, buena parte del cálculo puede realizarse en el lado cliente, accediendo al servidor de forma esporádica. Esto derivaría en un servidor con menos necesidades de recursos, un menor tráfico en la red y un mayor coste de los equipos en el lado cliente.

Por lo tanto, como puede deducirse, disponemos de dos alternativas:

- ✔ Cliente pesado, servidor ligero (*Thick Client, Thin Server*) ➜ Aquí, tanto el nivel de presentación como el nivel de negocio (aplicación) se ejecutan en el lado cliente. Incluso podrían procesarse contenidos multimedia con un alto consumo de recursos. El servidor se utiliza para tareas como el hospedaje del SGBD o incluso para otras tareas menores, como administrar las tareas de impresión. En este tipo de esquemas, incluso podría interrumpirse de forma momentánea el servicio de red sin perjudicar de forma significativa a los clientes.
- ✔ Servidor pesado, cliente ligero (*Thick Server, Thin Client*) ➜ El lado cliente se emplea sólo para el nivel de presentación (muchas veces utilizando simplemente un navegador web) y el lado servidor se encarga de ejecutar la aplicación. En este tipo de esquemas podríamos disponer, incluso, de clientes sin disco duro. Otra de las ventajas de esta opción es que ofrecen una mayor seguridad frente a intentos de acceso indebido.

Normalmente, la segunda alternativa presenta una mayor flexibilidad y un menor coste, siempre que se hayan resuelto los posibles problemas que mencionábamos antes.

2.3.2 Según el tipo de servicio que se ofrece

Obviamente, son muchos los servicios que se pueden ofrecer en una arquitectura cliente/servidor y sería muy extenso realizar una clasificación detallada de los tipos de servidor atendiendo a todos los servicios que pueden ofrecer. Por ese motivo, aquí nos limitaremos a incluir sólo los más importantes:

- ✔ Servidores de archivos ➜ Se suelen utilizar para crear almacenes de documentos en un lugar centralizado de la red (Copias de seguridad, imágenes, plantillas de documentos, etc.)

- ✔ Servidor de aplicaciones ➔ Consiste en servidor que ejecuta un conjunto de aplicaciones que proporcionan servicios de a los clientes. Un servidor de aplicaciones generalmente gestiona la mayor parte de las funciones de lógica de aplicación y de acceso a los datos de las aplicaciones. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.
- ✔ Servidores de bases de datos ➔ Normalmente están asociados a la utilización de aplicaciones cliente/servidor donde un proceso cliente requiere datos al servidor de bases de datos.
- ✔ Servidores web ➔ Su función consiste en devolver contenido web cuando un cliente lo solicita empleando en la comunicación el protocolo HTTP.

Lógicamente, el acceso a los datos será compartido por diferentes clientes de forma simultánea (aplicándose los mecanismos de protección necesarios sobre los datos ante la concurrencia de diferentes clientes y la existencia de distintos niveles de privilegio). En este sentido podemos utilizar dos enfoques:

1. Realizar la gestión de los usuarios, permisos y roles en el propio SGBD. Cuando se ejecute la aplicación web, el acceso a los datos se hará con una de las cuentas de usuario del SGBD. Será el propio SGBD el que controle el acceso a la información, liberando al servidor de aplicaciones de esta tarea.
2. Realizar la gestión de los usuarios, permisos y roles por la propia aplicación web y empleando una única cuenta de usuario con acceso total a los datos en el SGBD. En este caso, el SGBD se libera de esta tarea quedando únicamente para servir los datos que se le soliciten.

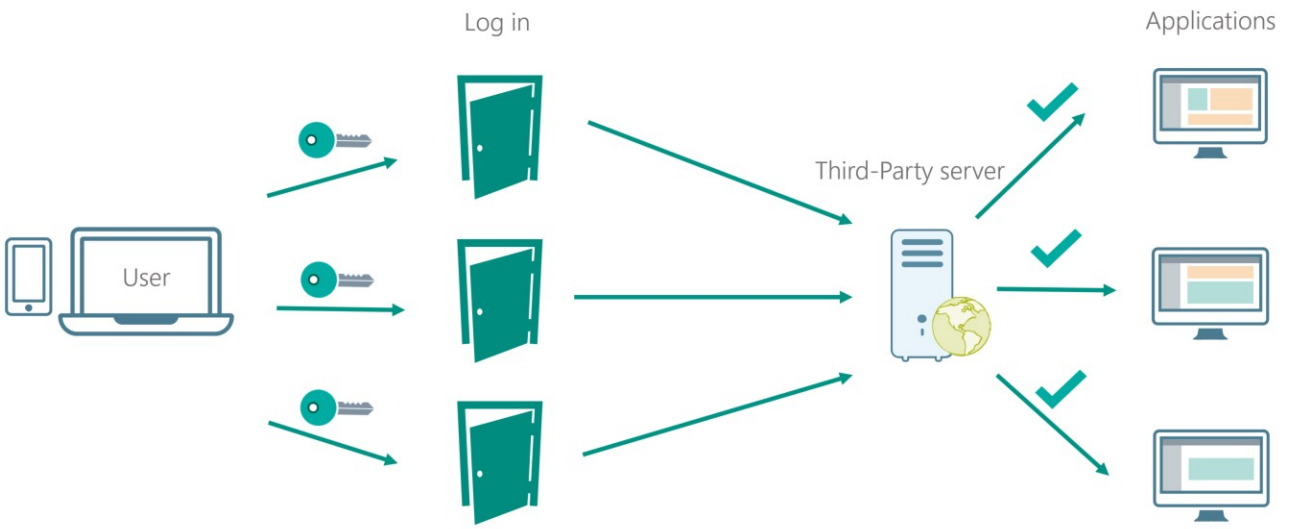
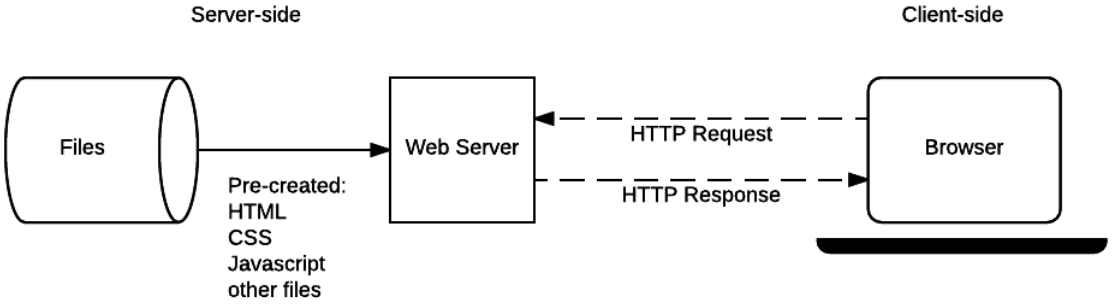


Figura 1.- Acceso autenticado

3 Generación dinámica de contenido web

Los navegadores web se comunican con los servidores web usando el protocolo HTTP. Al escribir una URL (dirección web) en la barra de dirección del navegador, se hace click en un enlace en una página web se está enviando una petición HTTP desde el navegador web al servidor web de destino. Los servidores web esperan las peticiones de los clientes, los procesan cuando llegan y responden al explorador web con un mensaje de respuesta HTTP.

El imagen inferior muestra una arquitectura de servidor web básica correspondiente a un sitio web estático. Cuando un usuario quiere navegar a una página, el navegador envía una petición HTTP especificando su URL. El servidor recibe la petición, la analiza y recupera de su sistema de ficheros el documento solicitado y devuelve una respuesta HTTP que contiene el documento y un código estado (normalmente 200 OK si la petición hecha pudo ser completada) o un código de error (404 Not Found, 403 Forbidden, ...) si por alguna razón la petición no puede ser completada.

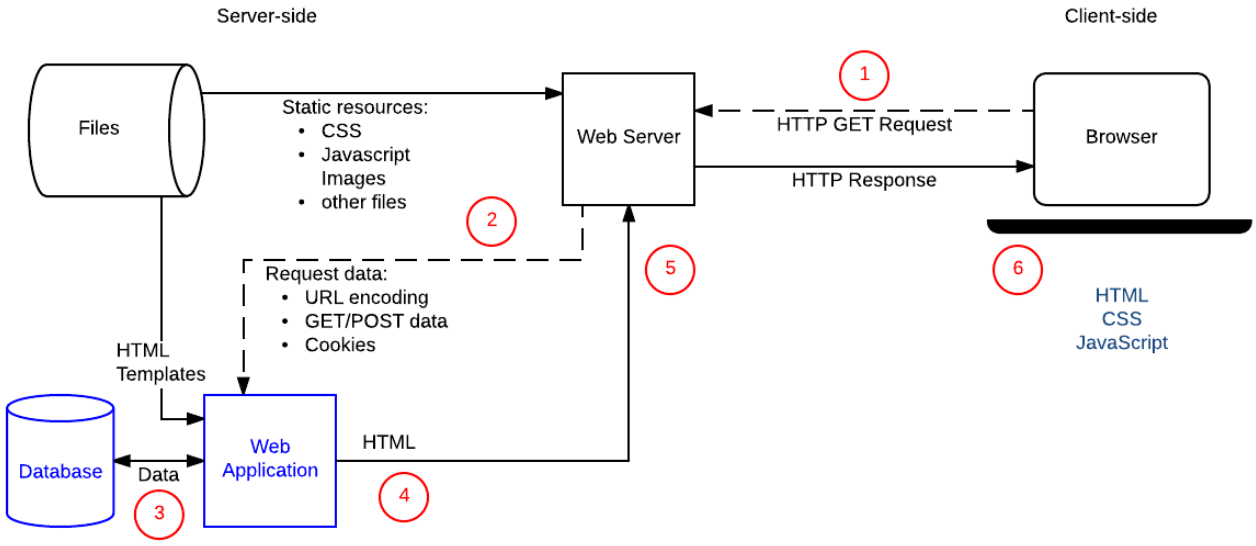


Un sitio web dinámico es aquél en que algún contenido de la respuesta está generado dinámicamente sólo cuando se solicita. En un sitio web dinámico las páginas HTML que se devuelven al cliente se han generado en el propio servidor web como respuesta a una solicitud. Pueden ser desde archivos HTML en los que se insertan algunos datos como resultado de algún proceso (plantilla HTML) hasta generación completa de una respuesta HTML a la solicitud del cliente.

Un sitio dinámico puede devolver datos diferentes para una misma URL basados en la información proporcionada por el usuario o sus preferencias almacenadas y puede realizar otras operaciones como parte de la devolución de respuesta.

El código de las aplicaciones en un sitio web dinámico se ejecuta en el servidor. La creación de este código se conoce como *programación de lado servidor* o *back-end scripting*.

La imagen inferior muestra una arquitectura simple para un sitio web dinámico y enumera los pasos que se ejecutan un ciclo completo petición/respuesta. Como en el diagrama previo, los navegadores web envían peticiones HTTP al servidor. El servidor procesa a continuación las peticiones y devuelve las respuestas HTTP apropiadas. Las peticiones de recursos estáticos son gestionadas de la misma manera que para los sitios estáticos (los recursos estáticos son cualquier archivo cuyo contenido que no cambia, generalmente: CSS, JavaScript, imágenes, PDFs creados previamente, etc.).



El proceso comienza con una petición HTTP del navegador web (1). Esta petición es reenviada (2) a una aplicación web la cual se encarga de generar el contenido de la respuesta. Para ello es probable que necesite leer información desde una base de datos (3). Estos datos, junto con otros generados por la propia aplicación web pueden combinarse (4) con archivos HTML y otros tipos de archivo con contenido estático (formato CSS, scripts javascript, imágenes, ...). La respuesta generada dinámicamente se entrega al servidor web (5) el cual la envía al cliente que hizo la petición inicial (6).

3.1 Ventajas

Es evidente que este segundo enfoque en la generación de contenido web para los navegadores tiene una serie de ventajas que han contribuido a su éxito. Aunque el coste inicial puede ser un poco más alto que el de una web estática, esto se compensa rápidamente con todas las ventajas que tiene y con el hecho de que su mantenimiento es menos costoso. Entre estas están:

- ✔ Facilidad en el cambio de diseño ➜ En una web dinámica se puede añadir y quitar elementos, tanto por parte de los desarrolladores y diseñadores web como por parte de los visitantes, según las necesidades de cada momento, haciendo que el diseño sea algo totalmente único en cada petición. Estos cambios se pueden aplicar de forma rápida y sencilla. De esta forma, en cualquier momento se puede realizar un cambio a la imagen de la web corporativa.

- ✔ Navegación más personalizada ➜ Uno de los mayores beneficios de tener un sitio web dinámico es ofrecer a los usuarios una experiencia de navegación personalizada cuando visitan un sitio web. Las sugerencias de contenido y productos, las funciones para los artículos vistos recientemente e incluso la oferta de contenido basada en la ubicación y los datos demográficos pueden contribuir a que los visitantes permanezcan en la página.

- ✔ Relación más sólida con el usuario ➜ En los sitios web dinámicos los usuarios no son elementos pasivos, pueden actuar y participar de muchas formas. En este tipo de web

se consigue crear una relación más estrecha entre los usuarios y los propietarios del sitio. Esto permite a las empresas conseguir nuevos clientes, establecer un vínculo más estrecho con ellos y que aumente la fidelización.

- ✔ Mejor posicionamiento en buscadores ➔ Una página de tipo dinámico tiene más posibilidades de posicionarse mejor que una estática.
- ✔ Dispositivos móviles ➔ Hoy en día la mayor parte del tráfico que reciben las páginas web llega a través de dispositivos móviles y no de ordenadores. Esto es algo que hay que tener muy en cuenta al diseñar páginas, porque la visualización no es igual en una pantalla de ordenador que en la pequeña pantalla de un móvil. Cuando se genera contenido dinámico puede hacerse para ajustarse al tamaño del dispositivo.

4 Lenguajes de programación en entorno servidor

Como cualquier otra aplicación, una aplicación web se escribe mediante un lenguaje de programación. Hemos visto anteriormente, que una aplicación web es un sistema distribuido en el que sus componentes pueden ubicarse en diferentes máquinas. Según el modelo de 3 capas, en el lado cliente estarán los componentes de la aplicación responsables de interactuar con el usuario, mientras que en el lado servidor están los componentes de la capa de negocio y de datos.

Por tanto, cuando hablamos de un lenguaje de programación en el lado del servidor nos referimos a un lenguaje de programación utilizado para desarrollar la parte de la aplicación que se ejecuta en el servidor y que generalmente accede a las bases de datos y aplica la lógica de negocio.

Desde la aparición del servicio WWW, se han utilizado diferentes lenguajes para generar desde el servidor contenido dinámico que se enviaba a los clientes. En el siguiente epígrafe veremos algunos de ellos.

4.1 Tipos

Las aplicaciones web han evolucionado y con ellas los lenguajes empleados para desarrollarlas, buscando siempre mayor flexibilidad y portabilidad de las mismas. En las primeras aplicaciones web de lado del servidor se empleó la tecnología CGI (*Common Gateway Interface*). Consiste en la primera tecnología que se empleo para que un cliente web solicitara datos a una aplicación ejecutada en un servidor web. CGI no es un lenguaje de programación en sí, sino que especifica un estándar para transferir datos entre el cliente y el programa ejecutado en el servidor. Es un mecanismo de comunicación entre el servidor web y una aplicación externa cuyo resultado final de la ejecución son objetos MIME. Las aplicaciones que se ejecutan en el servidor reciben el nombre de CGIs.

Un programa CGI puede ser escrito en cualquier lenguaje de programación que produzca un archivo ejecutable. Entre los lenguajes más habituales se encuentran: C, C++, Perl, Java, Visual Basic, Cobol... No obstante, debido a que el CGI recibe los parámetros en forma de texto será útil un lenguaje empleado que permita realizar manipulaciones de las cadenas de caracteres de una forma sencilla, como por ejemplo Perl.

Si bien la tecnología CGI resolvía el problema de la presentación exclusiva de información estática, al mismo tiempo presentaba dos limitaciones importantes: el problema de seguridad que podía suponer el hecho de que mediante una petición se pudiesen ejecutar programas indeseados en el servidor y la carga del servidor (si una página que lanzaba un programa era llamada desde 100 clientes simultáneamente, el servidor ejecutaba 100 procesos, uno por cada cliente que solicitaba dicha página).

Esta tecnología aun está en uso hoy en día. Sin embargo, han surgido nuevos lenguajes de programación que facilitan la tarea de desarrollo al programador a la vez que permiten más flexibilidad y portabilidad a las aplicaciones web de lado servidor. Fragmentos de código de estos lenguajes pueden ser incrustados en código HTML. Sus sentencias pueden ser interpretadas (como por ejemplo las páginas ASP o PHP) o precompiladas (como en las páginas JSP o ASP.net).

Una de las diferencias mas notables entre los lenguajes empleados para las aplicaciones web es la manera en que se ejecutan en el servidor web. Distinguimos tres grandes grupos:

- ✔ Lenguajes de guiones (scripting) ➜ Son aquellos en los que los programas se ejecutan directamente a partir de su código fuente original. Se almacenan normalmente en un fichero de texto plano y cuando el servidor web necesita ejecutarlo le pasa la petición a un intérprete, que procesa las líneas del programa y genera como resultado una página web. Pertenecen a este grupo Perl, Python, PHP y ASP (el precursor de ASP.Net).
- ✔ Lenguajes compilados ➜ Son aquellos en los que el código fuente se traduce a código binario, dependiente del procesador, antes de ser ejecutado. El servidor web almacena los programas en su modo binario, que ejecuta directamente cuando se les invoca. El método principal para ejecutar programas binarios desde un servidor web es CGI. Utilizando CGI podemos hacer que el servidor web ejecute código programado en cualquier lenguaje de propósito general como puede ser C.
- ✔ Lenguajes compilados a código intermedio ➜ Son lenguajes en los que el código fuente se traduce a un código intermedio, independiente del procesador, antes de ser ejecutado. Es la forma en la que se ejecutan por ejemplo las aplicaciones programadas en Java, y lo que hace que puedan ejecutarse en varias plataformas distintas. En la programación web, operan de esta forma los lenguajes de las arquitecturas Java EE (servlets y páginas JSP) y ASP.Net. Para acelerar la ejecución, el compilador puede traducir todo o parte del código intermedio a código nativo cuando se invoca a un programa. El código nativo obtenido suele almacenarse para ser utilizado de nuevo cuando sea necesario.

Cada una de estas formas de ejecución del código por el servidor web tiene sus ventajas e inconvenientes:

- ✔ Los lenguajes de guiones tienen la ventaja de que no es necesario traducir el código fuente para ser ejecutados, lo que aumenta su portabilidad. Si se necesita realizar alguna modificación a un programa, se puede hacer en el momento. Por el contrario

el proceso de interpretación ofrece un peor rendimiento que las otras alternativas.

- ✔ Los lenguajes compilados a código nativo son los de mayor velocidad de ejecución, pero tienen problemas en lo relativo a su integración con el servidor web. Son programas de propósito general que no están pensados para ejecutarse en el entorno de un servidor web. Por ejemplo, no se reutilizan los procesos para atender a varias peticiones: por cada petición que se haga al servidor web, se debe ejecutar un nuevo proceso. Además los programas no son portables entre distintas plataformas.
- ✔ Los lenguajes compilados a código intermedio ofrecen un equilibrio entre las dos opciones anteriores. Su rendimiento es muy bueno y pueden portarse entre distintas plataformas en las que exista una implementación de la arquitectura (como un contenedor de servlets o un servidor de aplicaciones Java EE).

4.2 Características de los lenguajes de scripting

En el desarrollo de una aplicación web dinámica resulta muy habitual utilizar un lenguaje interpretado, script, en el servidor. También es posible desarrollar dichas aplicaciones con lenguajes compilados como Java o incluso C/C++, pero aquí nos vamos a centrar en el uso de intérpretes. Entre las características de estos intérpretes encontramos:

- ✔ Multiplataforma ➜ Los lenguajes de lado servidor interpretados más utilizados cuentan con una licencia libre y están disponibles para diferentes plataformas. Por tanto pueden utilizarse en servidores Windows o Linux.
- ✔ Código integrado en HTML ➜ Permiten intercalar contenido estático HTML y código del propio lenguaje. En lugar de escribir todo un programa que genere la salida deseada, lo que se escribe es un documento (una página web) en la que mediante el uso de unas marcas se incluye el código que deberá ser interpretado.
- ✔ Se trata de lenguajes de alto nivel que facilitan el desarrollo rápido de la aplicación.
- ✔ Implementación completa de HTTP ➜ Estos lenguajes incluyen un conjunto de elementos que tienen acceso completo a la comunicación mediante el protocolo HTTP entre el cliente y el servidor web.

4.3 Integración con los lenguajes de marcas

Cuando la web comenzó a evolucionar desde las páginas web estáticas a las dinámicas, una de las primeras tecnologías que se utilizaron fue la ejecución de código utilizando CGI. Los scripts CGI son programas estándar, que se ejecutan por el sistema operativo, pero que generan como salida el código HTML completo de una página web. Por tanto, los guiones CGI deben contener, mezcladas dentro de su código, sentencias encargadas de generar la página web.

Hoy en día, sin embargo, los lenguajes de scripting o compilados a código intermedio de lado de servidor permiten incluir en su código fuente sentencias del lenguaje con elementos HTML, lo que facilita el desarrollo.

A la hora de integrar el código HTML con el código de la aplicación en el lenguaje de

lado servidor existen dos formas:

- ✔ Integrar las etiquetas HTML en el código de los programas ➜ Los programas incluyen dentro de su código sentencias de salida que son las que incluyen el código HTML de la página web que se obtendrá cuando se ejecuten. De esta forma se programan, por ejemplo, los guiones CGI y los servlets de JavaEE.
- ✔ Integrar el código del programa en medio de las etiquetas HTML ➜ De esta forma, el contenido que no varía de la página se puede introducir directamente en HTML, y el lenguaje de programación se utilizará para todo aquello que se debe generar de forma dinámica. Esta metodología de programación es más segura en la web frente a ataques tipo XSS y es la que se emplea en los lenguajes ASP, PHP y con las páginas JSP de Java EE.

Estas dos formas suelen implementarse en los lenguajes de scripting y compilados a código fuente. El siguiente ejemplo corresponde a la primera forma y genera una página web utilizando CGI a partir de un script bash de Linux:

```
# Create HTML
echo "Creating HTML file…"
> snapshots_preview.html
echo "<html><body><div style=\"width:100vw;\">" > snapshots_preview.html

# Obtaining PNGs files
echo "Iterating subdirectories"

for subdirectory in $(echo $subdirectories)
do
  # 1
  echo "<h1>Title$subdirectory</h1>" >> snapshots_preview.html
  echo "<div style=\"display: flex; flex-wrap: wrap; flex-direction: row; width: 100vw;\

  completePath=$(echo "$SNAPSHOTS_DIR/$TEST_TARGET.$subdirectory/*.png")
  for snapshotFilePath in $completePath
  do
    filename=$(echo $snapshotFilePath | cut -d'/' -f 5)

    # 2
    snapshotHTMLCode=$(echo "<div style=\"flex-direction: column; border:1px solid black
  done

  # 3
  echo "</div>" >> snapshots_preview.html
done

# 4
echo "</div></body></html>" >> snapshots_preview.html
```

Esta es una de las principales formas de realizar páginas web dinámicas: integrar las etiquetas HTML en el código de los programas. Es decir, los programas como el guion anterior incluyen dentro de su código sentencias de salida (`echo` en el caso anterior) que son las que incluyen el código HTML de la página web que se obtendrá cuando se ejecuten. De esta forma se programan, por ejemplo, los guiones CGI y los servlets.

Para la segunda forma tenemos el siguiente ejemplo, donde podemos incluir dentro de una página HTML un pequeño código en lenguaje PHP que muestre el nombre del servidor:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>
```

```
<?php
  echo "¡Hola, soy un script de PHP!";
  echo "El nombre del servidor es: " . $_SERVER['SERVER_NAME'];
?>

  </body>
</html>
```

4.4 Integración con los servidores Web

Como ya sabemos, la comunicación entre un cliente web o navegador y un servidor web se lleva a cabo gracias al protocolo HTTP. En el caso de las aplicaciones web, HTTP es el vínculo de unión entre el usuario y la aplicación en sí. Cualquier introducción de información que realice el usuario se transmite mediante una petición HTTP, y el resultado que obtiene le llega por medio de una respuesta HTTP.

En el lado del servidor, estas peticiones son procesadas por el servidor web. Es por tanto el servidor web el encargado de decidir cómo procesar las peticiones que recibe. Cada una de las arquitecturas que acabamos de ver tiene una forma de integrarse con el servidor web para ejecutar el código de la aplicación.

En el caso de las aplicaciones que siguen la especificación CGI, la gran mayoría de los servidores web lo implementan. Define qué debe hacer el servidor web para delegar en un programa externo la generación de una página web. Esos programas externos, los guiones CGI, independientemente del lenguaje en el que estén programados (aunque se suelen programar en lenguajes de guiones como Perl). El principal problema de CGI es que cada vez que se ejecuta un guión CGI, el sistema operativo debe crear un nuevo proceso. Esto implica un mayor consumo de recursos y menor velocidad de ejecución.

En el caso de las aplicaciones escritas con lenguajes de scripting interpretados, el servidor web suele incluir un módulo que se encarga de traducir y ejecutar el fragmento de código escrito en su correspondiente lenguaje. Por tanto, es necesario instalar el módulo correspondiente al lenguaje de programación elegido en el servidor web. Por ejemplo, si el lenguaje es PHP y el servidor web es Apache, habrá que instalar el módulo PHP para Apache. Cada servidor web dispone de la posibilidad de instalar módulos adicionales para interpretar y ejecutar los scripts del lenguaje de programación necesario.

La arquitectura Java EE es más compleja. Para poder ejecutar aplicaciones Java EE en un servidor básicamente tenemos dos opciones: servidores de aplicaciones, que implementan todas las tecnologías disponibles en Java EE, y contenedores de servlets, que soportan solo parte de la especificación. Dependiendo de la magnitud de nuestra aplicación y de las tecnologías que utilice, tendremos que instalar una solución u otra.

Existen varias implementaciones de servidores de aplicaciones Java EE certificados. Las dos soluciones comerciales más usadas son IBM Websphere y BEA Weblogic. También existen soluciones de código abierto como JBoss, Geronimo (de la fundación Apache) o Glassfish.

La arquitectura ASP.Net utiliza el servidor IIS de Microsoft, que ya integra soporte en forma de módulos para manejar peticiones de páginas dinámicas ASP y ASP.Net. La utilidad

de administración del servidor web incluye funciones para administrar las aplicaciones web instaladas en el mismo.

4.5 Herramientas de programación

Los componentes principales con los que debemos contar para ejecutar aplicaciones web en un servidor son los siguientes:

- ✔ Un servidor web para recibir las peticiones de los clientes web (normalmente navegadores) y enviarles la página que solicitan (una vez generada puesto que hablamos de páginas web dinámicas). El servidor web debe conocer el procedimiento a seguir para generar la página web: qué módulo se encargará de la ejecución del código y cómo se debe comunicar con él.
- ✔ El módulo encargado de ejecutar el código o programa y generar la página web de la respuesta. Este módulo debe integrarse de alguna forma con el servidor web, y dependerá del lenguaje de programación y tecnología que utilicemos para programar la aplicación web.
- ✔ Un SGBD que puede residir en el mismo servidor web o en otro diferente. No es estrictamente necesario pero en la práctica se utiliza en todas las aplicaciones web que utilizan grandes cantidades de datos para almacenarlos. Si está instalado, habrá que incorporar al servidor web un módulo del lenguaje del programación con las funciones de acceso a la base de datos.
- ✔ Un navegador web para realizar peticiones HTTP al servidor web. No es estrictamente necesario ya que existen herramientas que permiten hacer esta labor y, posteriormente, el desarrollador debe comprobar la salida. Sin embargo, si el desarrollo de la aplicación es fullstack, necesitaremos verificar que la capa de presentación se ejecuta adecuadamente.

Estos mismos componentes los vamos a necesitar para la fase de desarrollo de la aplicación, y además de un editor de código o un IDE adaptado al lenguaje de programación empleado.

A la hora de incluir estos componentes en el ordenador que vamos a utilizar para desarrollar la aplicación tenemos dos opciones:

- ✔ Instalar cada componente por separado ➔ Consiste en elegir primero e instalar después cada uno de los componentes necesarios: un servidor web, el módulo que interpreta y ejecuta el código de la aplicación y un SGBD. Esta opción nos otorga flexibilidad para elegir estos componentes por separado que mejor se ajusten a las necesidades de la aplicación web. Por ejemplo podemos instalar como servidores web Apache, Nginx o IIS. El módulo para ejecutar la aplicación puede ser el de PHP, Perl, ... Finalmente, el SGBD puede ser Oracle, MySQL, MariaDB, PostgreSQL, etc.
- ✔ Instalar una plataforma que integra todos los componentes ➔ Existen soluciones gratuitas en el mercado que incluyen un servidor web con su correspondiente módulo de ejecución web y un SGBD, todo ello integrado y con su configuración

centralizada para facilitar la creación de una plataforma de desarrollo de aplicaciones web. Por ejemplo tenemos XAMPP que incluye el servidor web Apache, el SGBD MariaDB y los módulos para PHP y Perl. Lo tenemos disponible tanto para plataformas Windows, como Linux y OS X. Otra es WampServer que incluye el servidor web Apache, el módulo para PHP y el SGBD MySQL, solo para plataformas Windows.

4.6 Frameworks

En los últimos años, y sobre todo en el campo de las aplicaciones web, se ha extendido el uso de los frameworks. Un framework es un entorno de trabajo que combina un conjunto de herramientas y técnicas que facilitan el desarrollo de una aplicación. Está diseñado para ayudar a los desarrolladores a crear sitios web de manera más eficiente y consistente, al proporcionar una serie de componentes y funciones comunes que se utilizan con frecuencia en el desarrollo web. Un framework extiende la definición de una biblioteca, en la que se incorporan otros elementos necesarios en el desarrollo de aplicaciones grandes y complejas.

No es un lenguaje de programación en concreto, sino la combinación de código reutilizable escrito en un lenguaje determinado (bibliotecas) para su utilización en el desarrollo de múltiples aplicaciones unido a un patrón de diseño, lo que facilita la escritura de código y reducen el esfuerzo de programación.

La mayor ventaja que ofrece un entorno de este tipo es que disminuye los intentos de prueba y error para comprobar el funcionamiento de un código. Asimismo, permite que estos no se repitan y, al contar con plantillas, hace más fácil la organización de los módulos web. Además, la mayoría de los que se encuentran en el mercado brindan una seguridad a prueba de ataques o de robo de datos.

Existen múltiples frameworks, cada uno es específico de un lenguaje de programación. También impone el uso de un patrón de diseño al que se tiene que ajustar la estructura de la aplicación a desarrollar, lo que permite estructurar mejor el código.

Algunos de los elementos y características comunes que incluye un framework son:

- ✔ Enrutamiento ➜ Suelen proporcionar un sistema de enrutamiento que facilita la gestión de las URL y las rutas de las páginas dentro de una aplicación.
- ✔ Plantillas ➜ Permiten crear un sitio web de manera más sencilla mediante el uso de plantillas predefinidas o motores de plantillas que ayudan a generar el código HTML de manera dinámica.
- ✔ Controladores ➜ A menudo utilizan el patrón de diseño MVC (modelo-vista-controlador) o una variante de él.
- ✔ Manejo de solicitudes y respuestas ➜ Ofrecen funciones para gestionar solicitudes HTTP entrantes y generar respuestas HTTP, lo que simplifica la interacción con el cliente.
- ✔ Bases de datos ➜ Muchos entornos integran herramientas y abstracciones para interactuar con bases de datos, lo que facilita el almacenamiento y recuperación de

los mismos.

- ✔ Seguridad ➜ Suelen incluir mecanismos de seguridad incorporados para proteger contra amenazas tales como los ataques de inyección SQL o ataques de scripting entre sitios (XSS).
- ✔ Sesiones y autenticación ➜ Proporcionan funciones para gestionar las sesiones de usuario y su autenticación, lo que facilita la creación de sistemas de inicio de sesión y control de acceso.
- ✔ API y servicios web ➜ Facilitan la creación y consumo de estas interfaces y servicios, lo que permite la integración con otras aplicaciones y servicios.

Figura 2.- Framework

De entre los frameworks más destacados están:

- ✔ React ➜ También conocido como React.js, es una biblioteca desarrollada por Facebook y una herramienta poderosa para crear interfaces de usuario interactivas y componentes reutilizables en aplicaciones web.
- ✔ Angular ➜ Desarrollada por Google; es una poderosa herramienta para la construcción de aplicaciones web de una sola página (SPA) y aplicaciones web en general.
- ✔ Django ➜ Se basa en el lenguaje de programación Python y proporciona un conjunto completo de herramientas y componentes listos para usar que aceleran el desarrollo web.
- ✔ Flask ➜ Es un microframework de desarrollo web en Python que me parece extremadamente sencillo y flexible de utilizar. A pesar de su simplicidad, Flask es muy

poderoso y versátil, esto lo convierte en una excelente opción para desarrolladores que desean crear aplicaciones web de manera rápida y eficiente.

- ✔ Ruby on Rails (Rails) ➜ A menudo simplemente llamado Rails, es un entorno de desarrollo web de código abierto escrito en el lenguaje de programación Ruby y se enfoca en simplificar el desarrollo web al proporcionar convenciones y abstracciones predefinidas para tareas comunes.
- ✔ Laravel ➜ Es un framework de código abierto para PHP y se ha convertido en uno de los frameworks PHP más populares y utilizados. Destaca por su enfoque en la elegancia del código, la simplicidad y la productividad en el desarrollo de aplicaciones.
- ✔ Symfony ➜ Framework de desarrollo web de código abierto en PHP y se ha convertido en uno de los más populares de su clase y respetados en la comunidad de desarrollo web. Está diseñado para facilitar la construcción de aplicaciones web robustas y escalables utilizando PHP.
- ✔ Spring ➜ Framework integral y altamente eficiente para el desarrollo de aplicaciones empresariales en Java. Es una herramienta muy utilizada y respetada en la comunidad de desarrollo Java debido a su enfoque en la simplicidad, la modularidad y la escalabilidad.
- ✔ Express.js ➜ Framework minimalista y flexible para Node.js altamente eficiente y popular en el desarrollo en JavaScript. Se emplea, por lo regular, para crear aplicaciones web y API utilizando Node.js, un entorno del lado del servidor.
- ✔ ASP.NET Core ➜ Framework de código abierto desarrollado por Microsoft versátil y potente. Está diseñado para permitir a los especialistas crear aplicaciones web modernas y escalables utilizando el lenguaje de programación C# y el entorno de tiempo de ejecución .NET Core (ahora conocido como .NET 5 y posteriores). Es una evolución de ASP.NET, pero con enfoque en la modularidad, el rendimiento y la compatibilidad multiplataforma.
- ✔ Gin ➜ Framework web ligero y de alto rendimiento para el lenguaje de programación Go (también conocido como Golang). Fue creado con el objetivo de proporcionar una manera eficiente y sencilla de construir aplicaciones web y API en Go. Gin se ha vuelto muy popular en la comunidad de desarrolladores de Go debido a su velocidad y facilidad de uso.

5 Bibliografía

GARCIA, J. (3 de julio de 2023) *Lenguajes de programación y herramientas*. https://jairogarciarincon.com/clase/arquitecturas-y-herramientas-de-programacion-en-lado-servidor/lenguajes-de-programacion-y-herramientas.

FRISOLO, C. (7 de septiembre de 2024) *Los 12 mejores frameworks para desarrollo web en 2024*. https://blog.hubspot.es/website/framework-desarrollo-web

MDN Contributors (1 de agosto 2023) *Introducción al lado servidor.* https://developer.mozilla.org/es/docs/Learn/Server-side/First_steps/Introduction