Arquitectura de aplicaciones web

FORMADORES { IT }

info@formadoresfreelance.es · www.formadoresit.es

Ayuntamiento de
FUENLABRADA

Índice

Aplicaciones de Escritorio y Cliente – Servidor

Actualmente existen infinidad de aplicaciones para multitud de necesidades distintas: aplicaciones de gestión, ofimática, aplicaciones móviles, páginas web, aplicaciones web, etc.

Existen multitud de clasificaciones diferentes, pero en este módulo vamos a estudiar dos grandes grupos muy genéricos que abarcan la gran mayoría de aplicaciones que encontramos en el mercado. Las aplicaciones de "escritorio" y las que siguen el esquema "cliente-servidor"

Veremos que, en función de las características de cada uno y del escenario en el que nos encontremos, deberemos utilizar una u otra.

A continuación, vamos a conocer sus características. También veremos algunas ventajas e inconvenientes de cada tipo de aplicación.

Aplicación de escritorio

Una aplicación de escritorio es toda aquella que se instala y ejecuta en un ordenador. Esta aplicación accede directamente a los recursos del sistema como la CPU, memoria, disco, redes, etc. Puede contar con una base de datos (por ejemplo, un fichero Access) o acceder a una base de datos externa mediante la red (por ejemplo, Oracle o MySQL).

Toda la lógica de la aplicación, transacciones o almacenamiento la lleva a cabo la propia aplicación de escritorio.

Ilustración 1: Aplicaciones Convencionales

Por estos motivos, nos encontramos con ciertos inconvenientes cuando queremos implantar este tipo de sistemas en entornos colaborativos o empresariales, donde necesitamos que muchas personas accedan a las mismas aplicaciones y datos:

- Duplicidad de datos si estos se encuentran recluidos en cada instancia (instalación) de la aplicación. Por ejemplo, guardar información en Hojas de excel que se van a compartir.
- El mantenimiento puede llegar a ser costoso, ya que suele ser necesario que todos los usuarios tengan instalada la misma versión del producto (sobre todo si acceden a datos comunes, se debe mantener la coherencia de la lógica de negocio).
- Falta de portabilidad o incompatibilidades entre diferentes sistemas operativos (o incluso en distintas versiones del mismo).
- Administración de la seguridad puesto que se encuentra dispersa por cada instalación de la aplicación.

Aplicaciones Cliente – Servidor

Estas aplicaciones constan de dos elementos principales:

- Servidor: en el servidor se encuentra la aplicación propiamente dicha con la lógica y reglas de negocio. Es el único punto de entrada a la aplicación desde los clientes (puede haber una o más instancias del servidor incluso varios programas por cada servidor, pero estos hechos serían transparentes para los clientes).
- Cliente: es la parte que se ejecuta en el ordenador del usuario. Puede ser un cliente pesado (una aplicación muy sencilla que se ejecuta en el equipo del usuario), un cliente web como un navegador (Firefox, Chrome, IExplorer, etc).

En este modelo, los clientes simplemente muestran el interfaz de usuario y recogen sus peticiones que son enviadas al servidor para su procesamiento. El servidor realiza las tareas necesarias (cálculos, accesos a fichero, almacenamientos en base de datos, etc.) en función de los requerimientos del cliente y posteriormente le devuelve los resultados para ser mostrados

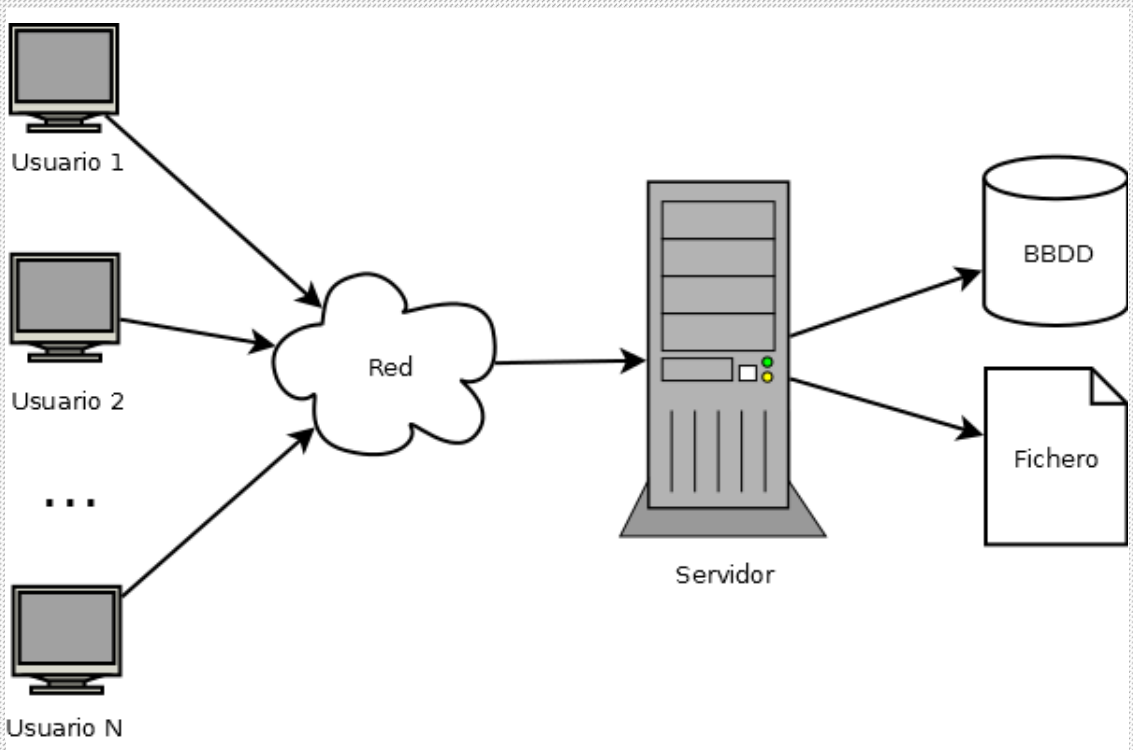


Ilustración 2: Aplicaciones Cliente-Servidor

Si la aplicación es accesible mediante un navegador web, estamos hablando de una "aplicación web". En este caso, el servidor genera el interfaz gráfico de la aplicación como una página web, que es mostrada por el navegador del cliente.

En entornos empresariales donde muchos usuarios quieren acceder a datos comunes, este modelo cuenta con muchas ventajas con respecto al modelo tradicional de aplicación de escritorio:

- Solo los ordenadores destinados a alojar los servidores deben contar con unos requisitos de potencia elevados ya que los clientes que son ejecutados en los equipos de los usuarios (la gran mayoría) son muy ligeros y, por tanto, necesitan muchos menos recursos para su correcto funcionamiento.
- La administración del sistema se simplifica en gran medida. Sobre todo, tratándose de una "aplicación web", una actualización del sistema solo sería necesaria en el servidor. También simplifica la tarea de la realización de copias de seguridad.
- Centralización de seguridad puesto que se encontraría en el servidor que es el punto común al que acceden todos los clientes.

- Los datos serían accedidos desde el servidor, por lo que todos los clientes podrían acceder, modificar y compartir los mismos datos.

La principal desventaja es la dependencia de la red:

- Puede que en algún momento no tengamos conexión. En este caso, la aplicación quedaría inaccesible.
- Limitaciones de ancho de banda que impidan el correcto funcionamiento de la aplicación o, simplemente, empobrezca la experiencia del usuario.

La arquitectura en tres niveles

Este tipo de arquitectura es uno de los más extendidos, tanto en aplicaciones de Internet, como empresariales.

Los niveles cliente, intermedia y datos:

- Cliente: es el encargado de mostrar el interfaz de usuario y realizar las peticiones que requiera el usuario al servidor.
- Nivel intermedio: es el servidor donde se encuentra nuestra aplicación y se encarga de procesar las peticiones del cliente y devolverle resultados a la vez que se comunica con un servidor de base de datos.
- Datos: se implementa mediante un servidor de base de datos y su tarea es la de almacenar y recuperar la información que aplicación solicita.

Características

Gran parte del éxito de este modelo radica en su flexibilidad y a la especialización de cada nivel.

El nivel intermedio es capaz de alojar múltiples aplicaciones cliente-diferentes y por lo tanto, atender a varios tipos diferentes de clientes.

A su vez, el nivel de datos puede almacenar información de una o varias aplicaciones diferentes alojadas en el nivel intermedio. Dichas aplicaciones podrían estar en distintos servidores e, incluso, estar desarrolladas con distintas tecnologías.

Además, permite aumentar la capacidad de cada nivel por separado. Por ejemplo, podríamos añadir más máquinas en el nivel intermedio para aumentar la potencia solo de dicho nivel si fuera necesario. A este concepto se le denomina "escalabilidad". Esto es aplicable a cualquiera de los niveles.

La seguridad de las aplicaciones y los controles de acceso, logs, etc. se encuentran centralizada en el nivel intermedio.

La gestión de copias de seguridad (backup) se podría realizar únicamente en el nivel de datos.

Interacción entre los distintos niveles

Petición desde el cliente:

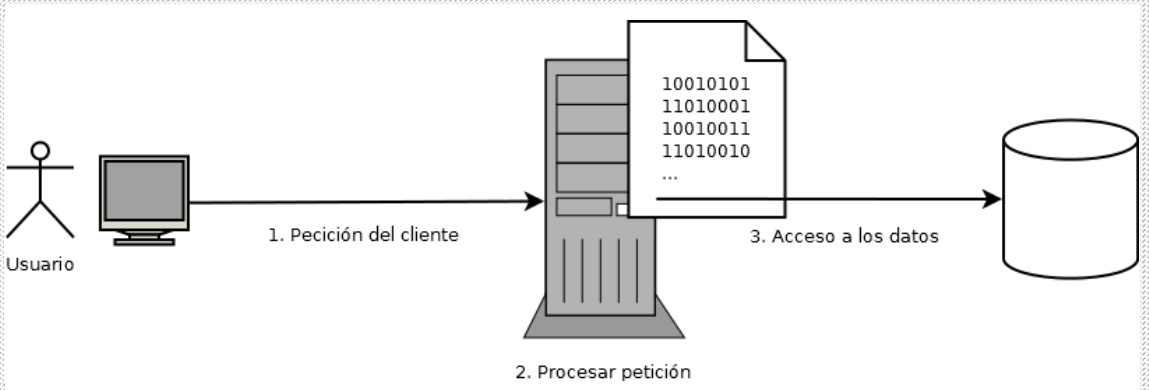


Ilustración 3: Interacción del usuario con la aplicación

Devolución de los datos al cliente:

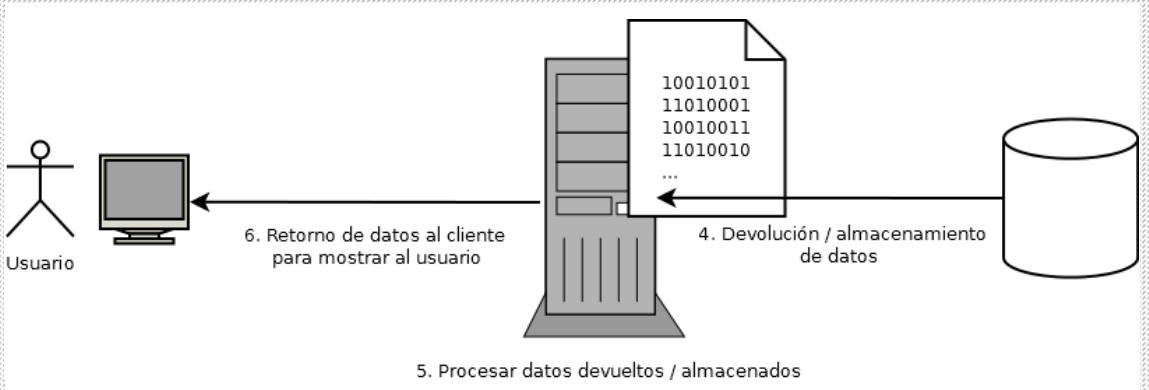


Ilustración 4: Devolución del resultado al usuario

Habitualmente, la comunicación entre el cliente y servidor se realiza mediante el protocolo HTTP. Es un protocolo basado en intercambio de documentos de texto plano y es el que se utiliza entre los navegadores web y los servidores de páginas para intercambiar contenido. Veremos el protocolo HTTP más adelante.

La comunicación entre el servidor y las bases de datos se realiza habitualmente mediante un protocolo propietario no estandarizado y diferente para cada

sistema gestor de base de datos (de ahora en adelante SGBD). Existen algunas excepciones, por ejemplo, en los incipientes SGDB "noSQL" como MongoDB que propone un intercambio de datos basados en el standard JSON (documentos de texto plano sobre HTTP).

Tecnologías en el nivel de cliente

A continuación, citamos algunas de las tecnologías más utilizadas en este nivel

- Navegadores web: ya hemos comentado con anterioridad las aplicaciones web, un subtipo concreto de aplicaciones de tipo cliente-servidor. Estas aplicaciones utilizan en su nivel de cliente un navegador web como puede ser Internet Explorer, Firefox, Google Chrome, Opera, etc. para acceder al nivel intermedio. Cada vez se están popularizando más debido a la ausencia de mantenimiento en el nivel de cliente puesto que, en principio, no sería necesario instalar o mantener ningún software adicional. Esto supone una drástica reducción del mantenimiento con respecto a las aplicaciones de escritorio convencionales ya que el número de equipos en este nivel puede llegar a ser muy elevado.
- HTML, CSS, javascript, JSON y XML: son los estandares de documentos, estilos y lenguajes de programación con los que el nivel intermedio suministra la información al nivel de cliente en las aplicaciones web (usan un navegador como cliente). Veremos en detalle todas estas tecnologías en módulos posteriores.
    - HTML (HyperText Markup Languaje): se trata de un lenguaje de marcado basado en etiquetas. Con este lenguaje, se puede describir el contenido de una pagina web que, posteriormente, será interpretado por un navegador para mostrarlo al usuario. Es el formato habitual en el que el nivel intermedio devuelve la información a un navegador web. Por ejemplo:

Ilustración 5: Visualización de código HTML

- CSS (Cascading Style Sheets): es un lenguaje creado para describir los estilos con los que un navegador va a presentar al usuario el contenido de un documento escrito en HTML. Dichos estilos se deben incluir en el documento HTML para su correcta visualización. Vemos a continuación un ejemplo:

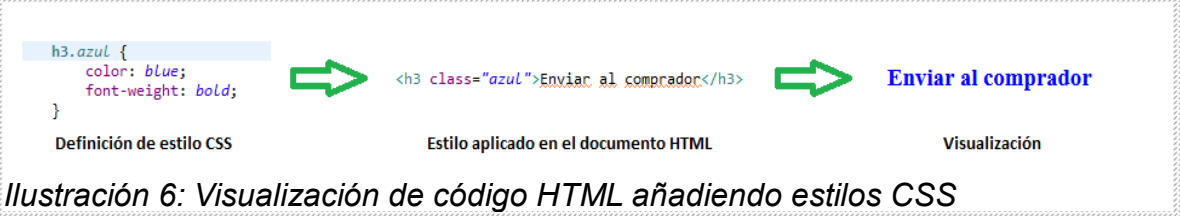


Ilustración 6: Visualización de código HTML añadiendo estilos CSS

- JavaScript: es un lenguaje de programación concebido inicialmente para su ejecución en el nivel de cliente. El contenido de una página HTML es estático, por lo que una página mostrada en el cliente no puede interactuar con el usuario (el navegador simplemente la interpreta y la muestra al usuario). Incluyendo código JavaScript en un documento HTML, podemos hacer que la página mostrada en el navegador pueda ser dinámica (cambiar en función de las acciones realizadas por el usuario) y comportarse de forma similar a como se comporta una aplicación de escritorio. De esta forma podemos mejorar la experiencia del usuario.
- JSON (JavaScript Object Notation): es un formato ligero y fácilmente interpretable por humanos diseñado para el intercambio de datos en la red. Aunque JSON se basa en la notación de objetos de JavaScript, está considerado como un lenguaje independiente.

Ejemplo:

```
{
    "postres": [
        {
            "nombre": "tarta",
            "precio": 5
        },
        {
            "nombre": "helado",
            "precio": 3
        }
    ]
}
```

Ilustración 7: Documento JSON

- XML (Extensible Markup Language): es un lenguaje de marcado utilizado para el intercambio de datos en la red y permite el intercambio de información entre diferentes plataformas. Al igual que JSON, está diseñado para su fácil interpretación por personas.

  Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<postres>
    <postre>
        <nombre>tarta</nombre>
        <precio>5</precio>
    </postre>
    <postre>
        <nombre>helado</nombre>
        <precio>3</precio>
    </postre>
</postres>
```

- Adobe Flash: Tecnología propietaria de la empresa Adobe. Se popularizó en gran al permitir en el nivel de cliente acceso a contenido multimedia y el desarrollo de interfaces ricas (interfaces complejas e interactivas que permiten crear una buena experiencia de usuario) a través del navegador web. La principal desventaja de Flash es la necesidad de instalar un plug-in en el navegador cliente con los problemas que esto conlleva (actualizaciones, incompatibilidades entre distintas plataformas, etc). Con las últimas evoluciones de HTML y CSS y la aparición de los dispositivos móviles (smartphones y tablets por ejemplo), esta tecnología está

perdiendo gran parte del mercado. Podemos encontrar muchas aplicaciones web en Internet que utilizan Flash en su nivel de cliente por que ha gozado de una amplia aceptación.

- Applets: Tecnología procedente de la plataforma de desarrollo Java cuya finalidad es análoga a Adobe Flash. Nació para los mismos fines y tiene el mismo inconveniente ya que hay que instalar un plug-in en el navegador del cliente para permitir la ejecución de los applets. Además, es necesario tener instalada la máquina virtual de java en el equipo. Está cayendo en desuso, pero podemos encontrar aún muchas aplicaciones que utilizan esta tecnología, sobre todo en entorno empresarial y aplicaciones de gestión.

- Cliente pesado: son aplicaciones que se instalan en el cliente y, solo en apariencia, se asemejan a las aplicaciones de escritorio convencionales. Sin embargo, los datos, lógica y reglas de negocio se encuentran en un servidor y no en la propia aplicación.

- Java Web Start: Es otra tecnología procedente del mundo java. Permite descargar una aplicación cliente escrita en java desde un link convencional. La aplicación en si no requiere instalación, se descarga automáticamente al pinchar el link. Su principal inconveniente es que requiere tener instalada la máquina virtual de java en el ordenador en el que se quiere utilizar la aplicación.

Tecnologías en el nivel intermedio

Como ya hemos comentado anteriormente, el nivel intermedio es el que alberga y ejecuta nuestra aplicación. A continuación, exponemos alguna de las tecnologías de este nivel.

- Servidores HTTP: son servidores que sirven contenido estático (por ejemplo, páginas HTML). Cada contenido se encuentra accesible mediante una dirección URL diferente (ej: http://httpd.apache.org). El intercambio de datos entre el cliente y el servidor se realiza mediante el

protocolo HTTP (lo veremos más adelante).
Uno de los servidores más populares es Apache2, desarrollado por Apache Software Foundation y cuya distribución es libre. Es un servidor modular, es decir, podemos añadir nuevas funcionalidades a nuestro servidor mediante la configuración de módulos adicionales. Algunos de estos módulos permiten ejecutar programas o interpretar cierto código que genera contenido HTML dinámicamente cuando desde el cliente se invocan ciertas direcciones URL. Algunos de estos módulos son:

- CGI / FastCGI: permite ejecutar un programa instalado en el servidor web. El servidor es capaz de ejecutar el programa y capturar los datos que devuelve por su salida estándard para enviarlos posteriormente al cliente. Es una de las formas más rudimentarias de generar contenido web de manera dinámica.
- PHP: permite la generación de contenido dinámico mediante la interpretación de código escrito en lenguaje PHP y la utilización de frameworks de desarrollo como DIY o Symphony2.
- Ruby: permite la generación de contenido dinámico mediante la interpretación de código escrito en lenguaje Ruby y frameworks como Ruby on Rails. Veremos en detalle Ruby y Ruby on Rails en módulos posteriores.

- Servidores de aplicaciones JAVA: es un servidor HTTP destinado única y exclusivamente a la generación de contenido dinámico generado por aplicaciones escritas en lenguaje Java.
Al conjunto de tecnologías y especificaciones para el desarrollo de aplicaciones web dinámicas en Java se le denomina Java Enterprise Edition (Java EE).
Los servidores de aplicaciones necesitan tener instalada la máquina virtual de Java para poder funcionar, ya que los servidores de aplicaciones son programas escritos también en lenguaje Java.
El servidor de aplicaciones se encarga, entre otras cosas, de gestionar el ciclo de vida de las aplicaciones, procesar el contenido de las páginas dinámicas que devuelve al cliente e interpretar las peticiones externas transformándolas en información estructurada que pasa a la aplicación correspondiente.

Algunos de los servidores más populares son Tomcat (de Apache), Jboss (de RedHat), OC4J y WebLogic (ambos de Oracle).

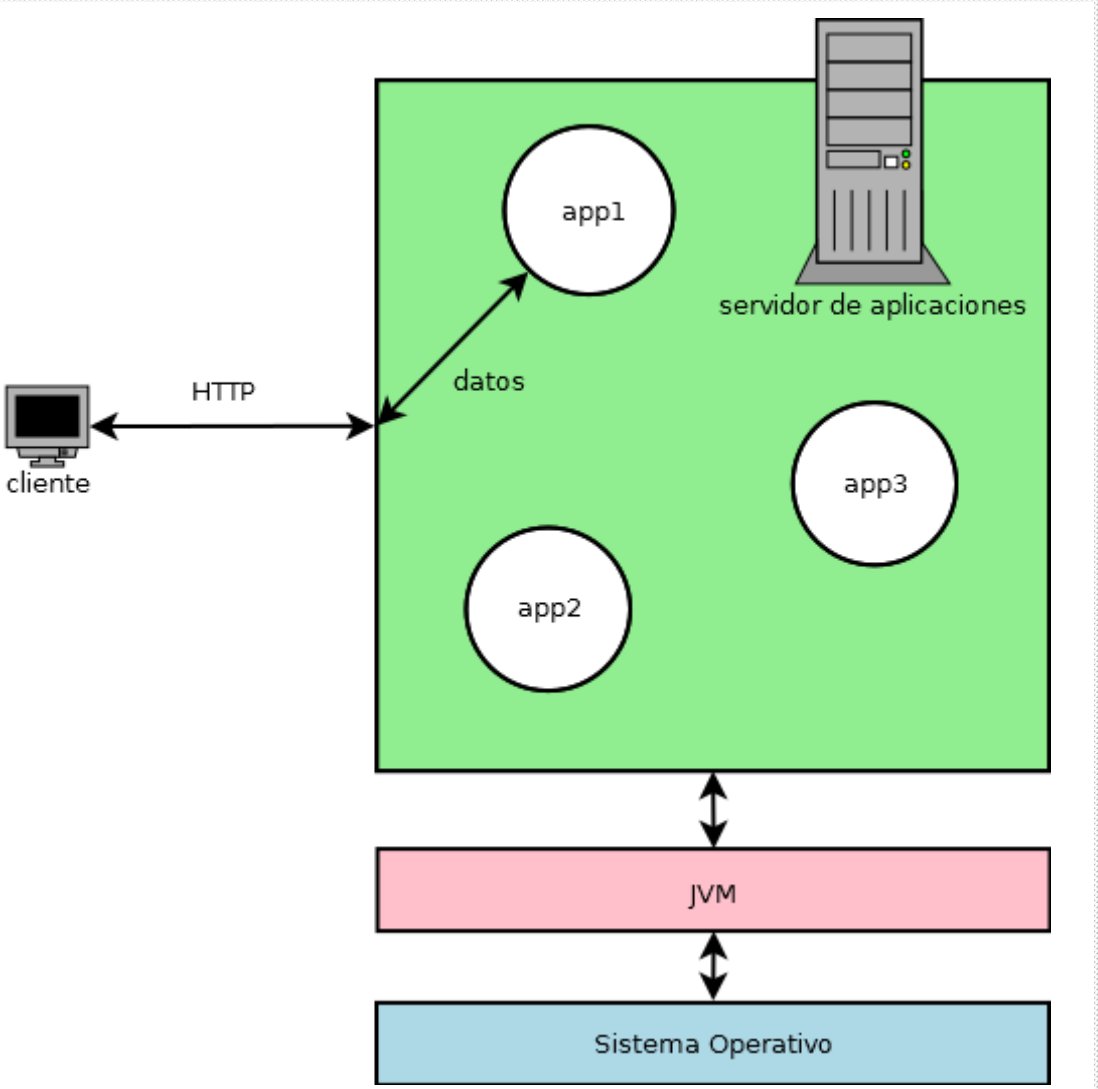


Ilustración 9: servidor de aplicaciones java

Tecnologías en el nivel de datos

El nivel de datos se suele implementar con un servidor en el que se instalan uno o varios Sistemas Gestores de Base de datos (SGBD). Podemos diferenciar los principales SGDB en dos tipos:

- Bases de datos relacionales (SQL): es un modelo propuesto a finales de los años 60 y actualmente es el modelo de base de datos más extendido. En este modelo, las unidades lógicas o entidades (elementos de almacenamiento de datos) solo pueden almacenarse en tablas. En este modelo también se contempla la existencia de relaciones entre tablas. Deben cumplir las características "ACID" (atomicidad, consistencia, aislamiento y durabilidad). Veremos en detalle estas características en módulos posteriores.
  La manipulación y consulta de los datos se realiza mediante la ejecución de sentencias escritas en lenguaje SQL (Standard Query Language). Es un lenguaje de cuarta generación (más cercano al lenguaje natural inglés, que a un lenguaje de programación convencional de alto nivel). En SQL existen dos subconjuntos del lenguaje:
  - DDL (Lenguaje de Descripción de Datos): utilizado en la construcción, modificación y eliminación de las estructuras almacenamiento de los datos.

```
create table clientes (
        nombre varchar(50),
        edad numeric(3)
);
```

Texto 1: sentencia DDL

  - DML (Lenguaje de Manipulación de Datos): utilizado para la manipulación de los datos contenidos en las estructuras anteriormente citadas.

```
select * from clientes; /* devuelve todos los datos de la tabla "clientes" */
```

Algunos SGDB ampliamente utilizados son:

- Oracle: SDDB propiedad de la compañía Oracle Corporation. Es la base de datos más extendida en el mundo empresarial y es uno de los más completos.
  Además de contar con SQL como lenguaje de consulta de datos, permite el la creación y ejecución de procedimientos y procesos almacenados mediante el lenguaje PL/SQL. Estos procedimientos y procesos almacenados pueden llevar a cabo operaciones muy complejas sobre los datos.
  Todas las versiones de Oracle son de pago excepto "Oracle Express Edition".

- MySql / MariaDB: Base de datos relacional desarrollada bajo licencia libre GPL. Al igual que Oracle, cuenta con SQL como lenguaje de consulta y manipulación de datos.
  Inicialmente era propiedad de MySQL AB hasta 2008 que fue adquirida por Sun Microsystems (empresa desarrolladora de la plataforma Java). Posteriormente Sun Microsystems fue comprada por Oracle Corporation en 2009 por lo que actualmente MySQL se encuentra bajo el control de Oracle.
  Es un SGDB ampliamente implantado. En parte, su éxito se debe a que, inicialmente, su licencia era GPL y por lo tanto libre para modificar o utilizar. Actualmente se encuentra en un esquema de licencia dual (parte es GPL y otra parte es propiedad de Oracle Corporation).
  Es utilizada por muchas empresas de reconocido prestigio como Facebook, Google o Wikipedia entre otros.
  Debido a la adquisición de MySQL por parte de Oracle Corporation, parte de los desarrolladores de MySQL (principalmente Michael Widenius, fundador de MySQL) crearon MaríaDB en 2009. Se trata de una bifurcación (o "fork") de MySQL que parte del código libre de MySQL y sobre la cual, continuan desarrollando.

- Bases de datos no relacionales (noSQL): son las que no cumplen con los aspectos más importantes del modelo relacional.
  La mayoría no usan el lenguaje SQL para manipulación o consulta de datos (aunque algunas también lo permiten).
  En la mayoría de los casos, no tienen la tabla como única forma conceptual de almacenamiento de datos.

En muchos casos tampoco cumplen con algunas de las características "ACID" (atomicidad, consistencia, aislamiento y durabilidad).
Suelen estar enfocadas a sistemas que requieren alta escalabilidad.

- Bases de datos orientadas a documentos: la información no se guarda en tablas como en las bases de datos relacionales, si no que se almacena con un formato de documento concreto como XML o JSON. Un ejemplo de este tipo de base de datos sería MongoDB que almacena documentos en formato JSON. Su lenguaje de consulta y manipulación de datos es propio, pero también implementa el formato JSON.
- Bases de datos clave / valor: almacena los datos en estructuras de datos de tipo diccionario: el dato que queremos almacenar (valor) se almacena en un contenedor de la base de datos asociado a una clave. Por ejemplo:

  Imaginemos que almacenamos los datos de una persona asociados a un DNI:

  "111233A": {"nombre" : "alberto", "apellidos": "garcía"}
  "445533B": {"nombre" : "jesús", "apellidos": "robledo"}

  Si accedemos a nuestra colección de datos con la clave *111233A*, nos devolverá el valor

  {"nombre" : "alberto", "apellidos": "garcía"}

  Se usan en sistemas en los que los accesos siempre se hacen por clave y donde hay muchos más almacenamientos que consultas.
  Un ejemplo de SGDB que implementa este modelo de base de datos sería Redis.
- Bases de datos orientadas a Grafos: Almacenan la información en una estructura de datos de tipo "Grafo".
  Un grafo es una estructura compuesta de nodos (o vértices) y aristas (o arcos). Los nodos se relacionan entre si mediante las aristas.
  En una base de datos orientada a grafos, la información se almacena

en los nodos, y añadimos relaciones entre los nodos mediante aristas. De esta forma, podemos aplicar los algoritmos de la teoría de grafos para recorrer la información existente en la base de datos.
Están destinadas a sistemas en los que prima el estudio de las relaciones entre los elementos de información (por ejemplo: calcular un trayecto entre dos puntos de un mapa, estudiar el comportamiento de una red, o las relaciones entre personas en un experimento socio-cultural).
Un ejemplo de este tipo de base de datos sería Neo4j.

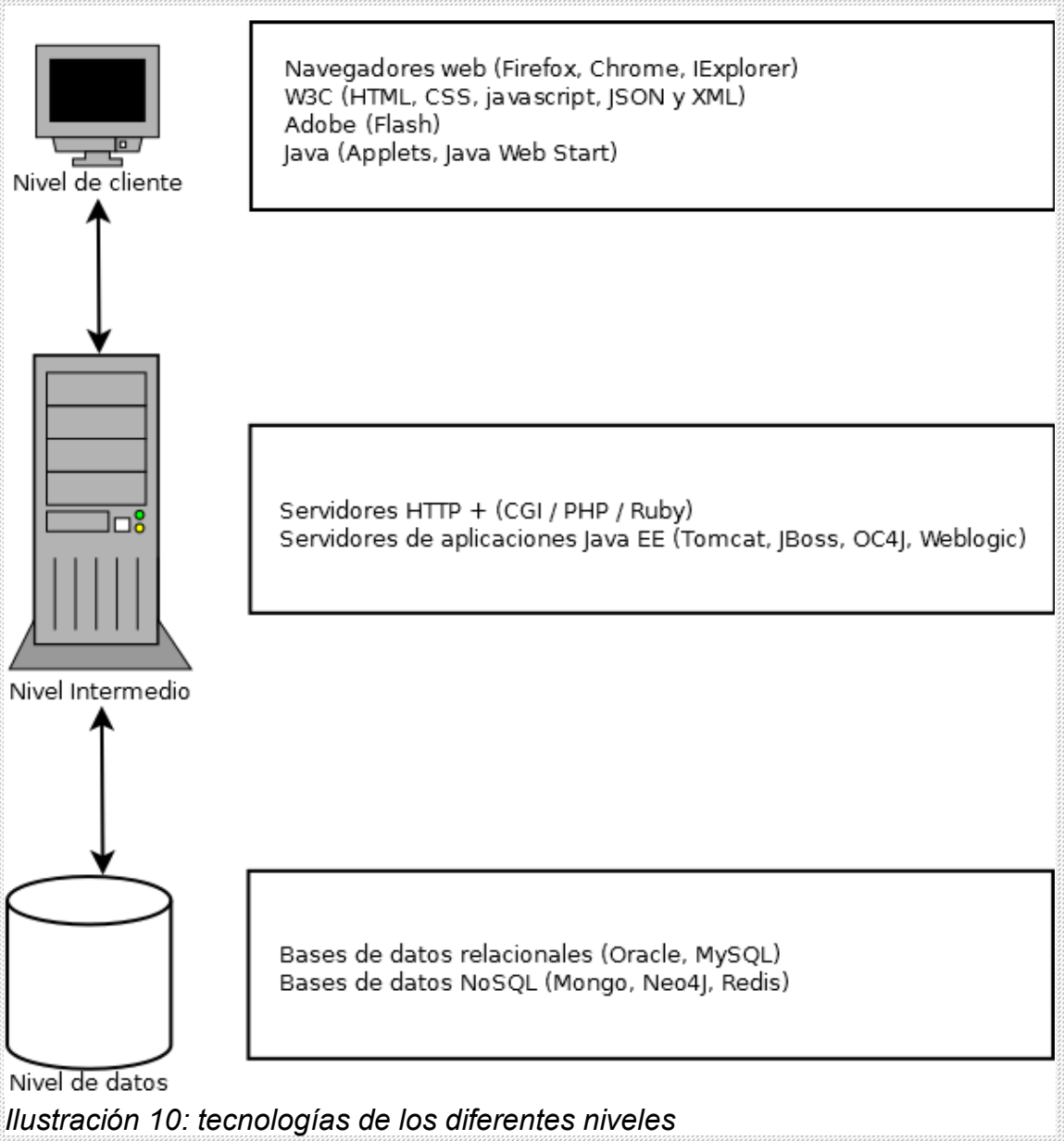


Ilustración 10: tecnologías de los diferentes niveles

Protocolos de comunicación entre nivel cliente y nivel intermedio

Son los mecanismos o estándares que debemos cumplir para poder relacionarnos con un elemento de una red. Cada elemento puede usar un protocolo distinto y debemos usarlo para poder comunicarnos con l. A continuación, vemos algunos de los más comunes:

- URL (Uniform Resource Locator): es una especificación que define la manera en la que accedemos a los recursos de la red. En una url se define el protocolo, la dirección de la máquina donde se encuentra el recurso y su ruta en dicha máquina entre otras cosas. Por ejemplo, de URL podría ser:
  http://www.google.es
- Protocolos HTTP (Hypertext Transfer Protocol): es un protocolo que define los diferentes actores existentes en una arquitectura web (clientes, servidores, proxies, etc) y como se comunican entre ellos.
  Mediante este protocolo se pueden transferir recursos (datos como paginas HTML, documentos JSON, ficheros de imágenes, etc) entre los distintos elementos que son solicitados utilizando una URL.
  La última versión es la 1.2 y se publicó en el año 2000, pero la más importante es la versión 1.1.
  Veremos este protocolo con más detalle en la siguiente sección.
- HTTPS: versión segura del protocolo HTTP. Se utiliza para la transmisión de información sensible. Cifra los datos contenidos en un datagrama HTTP con cifrado SSL/TLS.
- RPC / RMI: ambos protocolos se utilizan ampliamente en la comunicación cliente-servidor. Se utilizan principalmente en aplicaciones con cliente pesado (aplicación cliente) y para comunicar los niveles intermedios de diferentes aplicaciones. Son protocolos que permiten invocar la ejecución de código alojado en otra máquina.
  - La parte servidora debe tener expuestas las funciones o métodos que queremos ejecutar desde el cliente.
  - La parte cliente puede ejecutar las funciones o métodos expuestos por la aplicación que se encuentra en el servidor.

La diferencia entre RPC y RMI viene determinada por el tipo de lenguale utilizado:

- RPC (Remote Procedure Call): se utiliza en lenguajes de programación estructurados (por ejemplo C) donde los artefactos de código que ejecutan las funcionalidades son los procedimientos. Los procedimientos son los elementos que se exponen desde la parte servidora para que se puedan llamar desde el cliente.
- RMI (Remote Method Invocation): se utiliza en lenguajes orientados a objetos (como es el caso de Java). Estos lenguajes se organizan en objetos que tienen ciertas cualidades y métodos. Los métodos son las acciones que ejecutamos sobre un objeto y estos son los artefactos de código que exponemos en la parte servidora.

Protocolos de comunicación entre nivel intermedio y nivel de datos

- Protocolos propietarios: es habitual que los SGDB implementen protocolos de comunicación propietarios. El propio desarrollador de la solución de base de datos debe proporcionar un conector adecuado que implemente dicho protocolo para poder acceder a la base de datos desde una aplicación.
- HTTP + JSON: la combinación de documentos en JSON enviados mediante el protocolo HTTP se está extendiendo cada vez más y es más común encontrarlos sobre todo en bases de datos NoSQL. Un ejemplo claro es la incipiente MongoDB.

Protocolo HTTP. Principios de funcionamiento

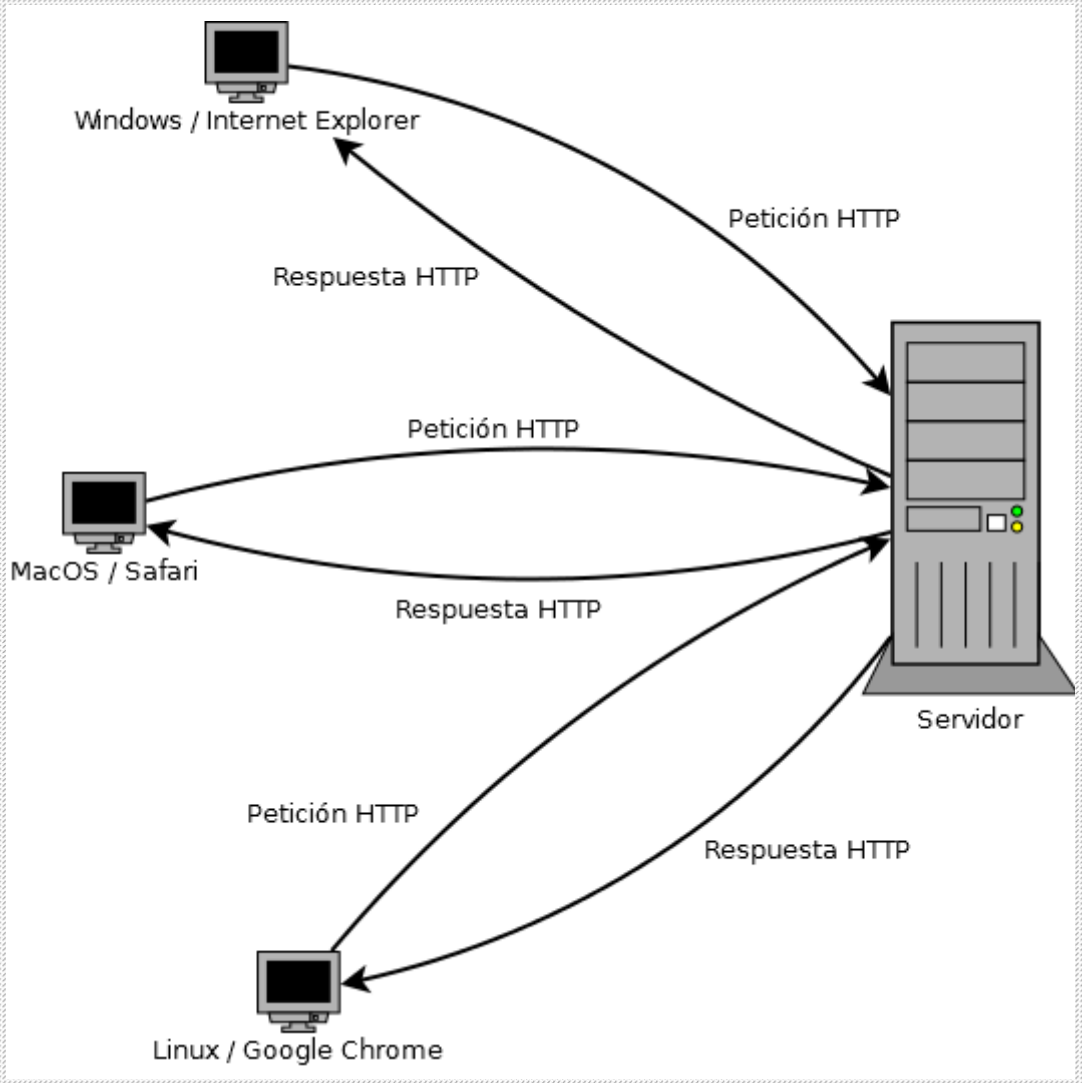
El protocolo HTTP ha sido desarrollado por el W3C (World Wide Web Consortium). La versión del protocolo más utilizada es HTTP 1.1.

En este protocolo se define la manera en la que interactúan los diferentes elementos de la web (clientes, servidores y proxies).

Es un protocolo orientado al intercambio de mensajes mediante el esquema de petición / respuesta entre un cliente y un servidor.

Los mensajes se envían en texto plano.

Es independiente de la plataforma que estemos usando. Un servidor puede responder a clientes de distinta naturaleza y viceversa.



URL

Antes de ver como trabaja HTTP, necesitamos conocer como se construye una URL. La URL es el elemento que nos va a permitir indicar cuales son recursos a los que queremos acceder (ya sea mediante HTTP o cualquier otro protocolo).

Cuando nosotros escribimos en un navegador una URL (por ejemplo: http://httpd.apache.org/ABOUT_APACHE.html, lo que realmente estamos haciendo es solicitar a nuestro navegador que acceda a ese recurso.

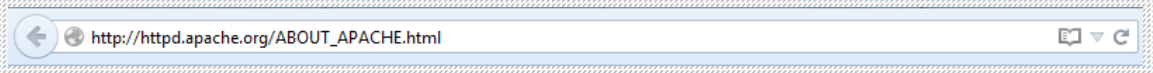
Estas son las partes principales de una URL:

protocolo://máquina[:puerto][/ruta del recurso]

- **protocolo**: es el protocolo con el que vamos a comunicarnos con el servidor. En el caso del ejemplo sería HTTP.
  La especificación URL también admite otros protocolos:
  - HTTPS: versión segura de http
  - FTP: protocolo para transferencia de ficheros
  - FILE: recursos existentes en el propio sistema o en una red local
- **máquina**: nombre de dominio o dirección IP en la que se encuentra el servidor con el que queremos comunicarnos.
- **puerto** (opcional): puerto en el que escucha el servidor al que queremos acceder. Si no se especifica, se utiliza el puerto por defecto que tiene asignado el protocolo. En el caso del protocolo HTTP es el puerto 80.
- **ruta del recurso** (opcional): ruta del recurso al que queremos acceder. En el caso del ejemplo, estamos solicitando la página "ABOUT_APACHE.html"

Transacción HTTP

Que es lo que sucede cuando en nuestro navegador web escribimos una ruta como la del ejemplo:



1. El navegador inicia una conexión TCP con la máquina que se encuentra en la dirección *httpd.apache.org*. Como no indicamos explícitamente un

puerto, se la conexión se establecería a través del puerto 80, que es el puerto por defecto para el protocolo HTTP.

2. El servidor acepta la conexión TCP y lo notifica al cliente.
3. El navegador envía una petición HTTP a través de la conexión establecida solicitando el recurso *ABOUT_APACHE.html*.
4. El servidor recibe la petición HTTP y genera una respuesta donde va a incluir el contenido del recurso solicitado (la página HTML *ABOUT_APACHE.html*) y cuando termina, cierra la conexión TCP
5. El cliente recibe la respuesta HTTP y lee el contenido en el que se encuentran los datos de la página solicitada, procesándolo y mostrándolo al cliente.

Petición HTTP

Una petición HTTP 1.1 consta principalmente de las siguientes secciones separadas por un salto de linea:

- Petición HTTP. Tiene el siguiente formato:
  METODO dirección_recurso VERSION_PROTOCOLO
  - Método (o verbo): indica la acción que deseamos realizar sobre el recurso solicitado. Pueden ser *HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS y CONNECT*. Los más utilizados son *GET, POST y HEAD* que explicaremos en detalle más adelante.
  - Dirección del recurso: ruta donde se encuentra el recurso dentro del servidor.
  - Versión del protocolo: versión de HTTP que estamos usando. En este caso su valor sería *HTTP/1.1*.
- Cabeceras del mensaje: las cabeceras se añaden con el siguiente formato:
  nombre-cabecera: valor cabecera
  Algunas cabeceras que se suelen incluir en una petición son:

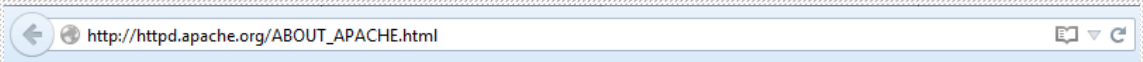
- Host: el valor que se asigna a esta cabecera tendría el siguiente formato: *host[:puerto]*
  "host" es nombre o dirección IP del ordenador donde está alojado el servidor. El puerto es opcional (si no lo indicamos, se accedería por el 80).
- User-Agent: indica al servidor el tipo de cliente que está realizando la petición.

- Linea en blanco
- Cuerpo del mensaje (opcional): en esta sección podemos incluir el contenido oportuno. Por ejemplo, un documento XML o JSON.
  Si en una petición POST se añaden parámetros, se incluyen aquí.

Los métodos más habituales en una petición HTTP son los siguientes:

- GET: solicita un recurso al servidor. Se pueden añadir parámetros en la propia URL.
- POST: solicita al servidor que añada un recurso. Los parámetros de la petición, así como el recurso a insertar no se envían en la URL, si no el él cuerpo del mensaje.
- HEAD: similar a GET, pero en la respuesta solo se incluye información del recurso solicitado (por ejemplo, la fecha de última modificación).

Ejemplo de una petición GET



El navegador interpreta la URL, establece una conexión TCP con el servidor y envía la siguiente petición HTTP:

```
GET /ABOUT_APACHE.html HTTP/1.1
Host: www.apache.org
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
(línea en blanco)
```

Respuesta HTTP

Una respuesta HTTP 1.1 consta principalmente de las siguientes secciones separadas por un salto de linea:

- Respuesta: tiene el siguiente formato:
  VERSION_PROTOCOLO código_estado descripción_estado
  - Versión del protocolo: versión de HTTP que estamos usando. En este caso su valor sería *HTTP/1.1*.
  - Código de respuesta: código que indica si la petición ha sido exitosa o se ha producido algún error indicando cual. Por ejemplo: *200* => la petición ha sido exitosa
  - Descripción de la respuesta: es la descripción asociada al código de respuesta. Por ejemplo, la descripción asociada al código *200* es *OK*.
- Cabeceras del mensaje: las cabeceras se añaden con el siguiente formato:
  nombre-cabecera: valor cabecera.
  Algunas cabeceras que se suelen incluir en una respuesta son:
  - Server: nombre que identifica el servidor que nos responde.
  - Content-Type: tipo de contenido que nos devuelve en el cuerpo del mensaje. Es muy importante para que el navegador sepa el tipo de datos que va a manejar. Por ejemplo, si nos devuelve HTML, el valor para esta cabecera sería *text/html*.
  - Content-Length: numero de bytes que hay en el cuerpo del mensaje.
- Linea en blanco.
- Cuerpo del mensaje (opcional): contiene los datos de la respuesta. En las peticiones *HEAD*, por ejemplo, no se añade cuerpo al mensaje ya que solo nos interesan sus cabeceras.

1xx: mensaje informativo	
2xx: éxito	200: OK 201: Created 202: Accepted 204: No Content
3xx: redirección	300: Multiple Choice 301: Moved Permanently 302: Foud 304: Not Modified
4xx: error del cliente	400: Bad Request 401: Unauthorized 403: Forbiden 404: Not Found
5xx: error del servidor	500: Internal Server Error 501: Not Implemented 502: Bad Gateway 503: Service Unavailable

Tabla 1: Codigos de respuesta HTTP

Ejemplo de respuesta a la petición GET del ejemplo anterior:

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 44
Connection: close
Content-Type: text/html
[ línea en blanco ]
<html>
     <body>
          <h1>Pagina solicitada</h1>
     </body></html>
```

Las capas software del nivel intermedio

En las aplicaciones cliente-servidor se suelen usar arquitecturas multinivel en el nivel intermedio. Estas arquitecturas se basan en la división del software en diferentes capas que desempeñan tareas muy especializadas y simples.

Actualmente, el modelo más extendido es el de tres capas y consta de los siguientes elementos:

- Presentación: constituye el elemento que genera las interfaces de usuario para comunicarse con él, captura los datos generados por las acciones del usuario y las envía a la capa de negocio para su procesamiento. Algunas estrategias de implementación podrían ser las siguientes:
  - Aplicación con cliente pesado, esta capa se suele implementar completa en el propio cliente.
  - Aplicación web: la generación de interfaces y la invocación a la capa de negocio se realiza en la capa intermedia. La visualización del interfaz y la interacción con el usuario se realiza en el propio navegador del cliente.
- Negocio: su tarea es procesar los datos que vienen de la capa de presentación, acceder a la capa de datos y aplicar las reglas de negocio pertinentes para devolver un resultado a la capa de presentación.
- Acceso a datos: es la encargada de acceder al nivel de datos (se conecta con el servidor de datos) para consultar y manipular la información alojada en dicho nivel. Cuando desarrollamos con un lenguaje orientado a objetos, esta capa se encuentra habitualmente asociada al patrón DAO (Data Access Object).

Tecnologías como Java, Ruby o PHP son utilizadas para la implementación de esta capa. Las veremos más adelante.

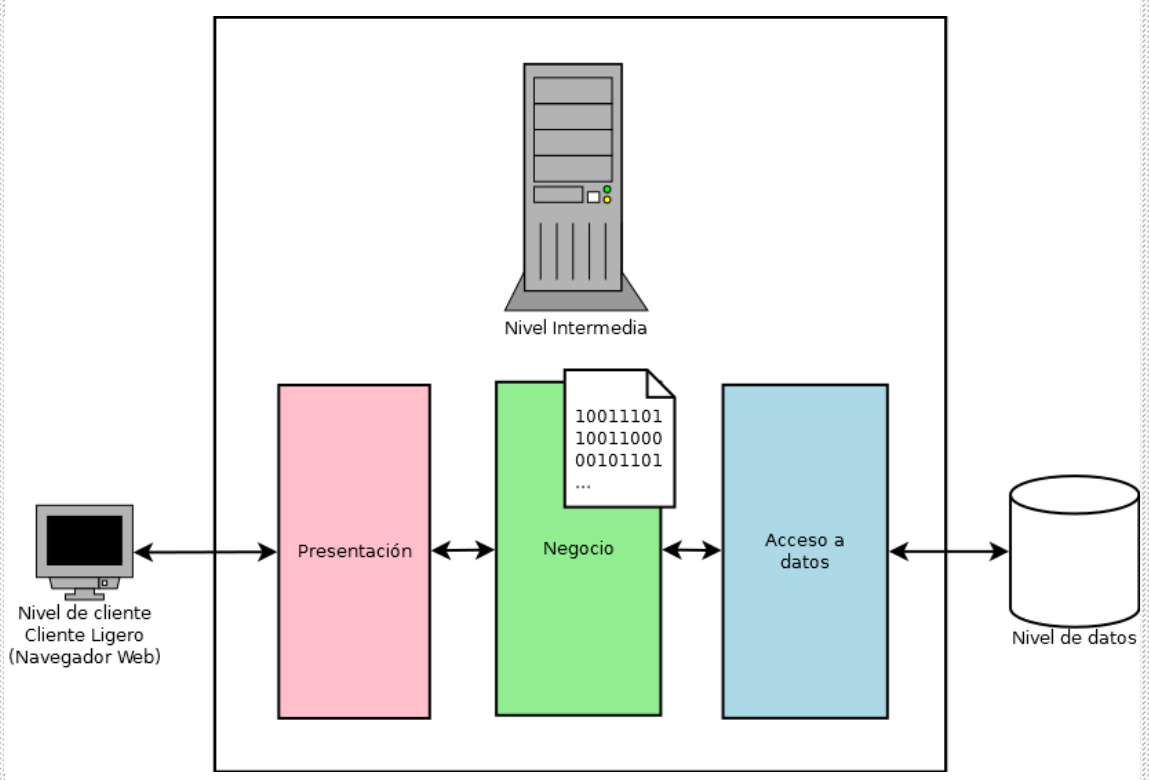


Ilustración 12: Modelo de tres capas en el nivel intermedio

Características generales de Java, PHP y Ruby

Estos tres lenguajes son muy utilizados en el desarrollo de aplicaciones actualmente. A continuación, vemos algunas características de cada uno:

- PHP: es un lenguaje de programación de propósito general interpretado originalmente diseñado para el desarrollo de aplicaciones web con contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página Web resultante. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.
  PHP se considera uno de los lenguajes más flexibles, potentes y de alto rendimiento conocidos hasta el día de hoy, lo que ha atraído el interés de múltiples sitios con gran demanda de tráfico, como Facebook, para optar por el mismo como tecnología de servidor.

Como framework de desarrollo de aplicaciones web con PHP destaca Symfony. Implementa el patrón Modelo-Vista-Controlador y proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web.
Para el despliegue de aplicaciones Ruby y Ruby on Rails, solo necesitamos un servidor HTTP como Apache y un módulo para ejecución de ruby como *mod_php* y *mod_php5* para la última versión de PHP.

- Ruby: es un lenguaje de programación interpretado y orientado a objetos. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Ruby es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre.
  Como framework de desarrollo de aplicaciones web con Ruby destaca Ruby on Rails (RoR) o simplemente Rails. Implementa el patrón Modelo-Vista-Controlador y trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración .
  Para el despliegue de aplicaciones Ruby y Ruby on Rails, solo necesitamos un servidor HTTP como Apache y un módulo para ejecución de ruby como *mod_ruby* o *passenger*.
- Java: es un lenguaje de programación de propósito general orientado a objetos. Fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como *WORA*, o "*write once, run anywhere*"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. compila el código fuente escrito en lenguaje Java, para generar un código conocido como ”bytecode” (instrucciones máquina simplificadas específicas de la plataforma Java). Esta pieza está “a medio camino” entre el código fuente y el código máquina que entiende el

dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Para desarrollar aplicaciones web con Java, tenemos que escribirlas de manera que cumplan la especificación Java EE.
La plataforma Java cuenta con infinidad de frameworks entre los que destaca Spring, un framework modular y multipropósito. Cuenta con módulos que implementan el patrón Modelo-Vista-Controlador, seguridad, acceso a datos, etc.
Para el despliegue de aplicaciones web con Java, necesitamos tener instalado en el equipo una máquina virtual de Java compatible con la versión de Java con la que hemos compilado nuestra aplicación y un servidor de aplicaciones que implemente la especificación Java EE.

Despliegue de aplicaciones

El despliegue de una aplicación es su puesta en producción. Existen diversas formas de desplegar aplicaciones, pero dependen principalmente de la tecnología que estemos usando para desarrollar.

Sistemas manuales

El sistema más habitual y sencillo es la copia de ficheros. Copiamos los ficheros de nuestra aplicación o página web en directorio configurado en el servidor. Las páginas web estáticas HTML o tecnologías como Java, PHP o Ruby admiten este sistema.

En el caso concreto de Java, se pueden empaquetar en un único fichero comprimido. Hay varios tipos:

- Jar (Java ARchive): agrupa clases java compiladas para poder utilizarlo como librería en otras aplicaciones.
- War (Web ARchive): agrupa clases java y documentos web como páginas estáticas HTML o dinámicas como JSP.
- Ear (Enterprise ARchive): aplicación Java EE completa que incluye ficheros *jar* y *war*.

Integración continua

Se trata de un sistema que realiza tareas de manera automática tales como la compilación de un proyecto, ejecución de pruebas, despliegue de aplicaciones en entornos de prueba, etc.

Por ejemplo, si tenemos una aplicación en java, nuestro sistema de integración continua podría descargarse el código fuente del lugar donde lo almacenamos, compilarlo, generar informes y, por último, desplegar nuestra aplicación en el servidor de pruebas si todas las operaciones anteriores han sido exitosas.

Jenkins, por ejemplo, es uno de los sistemas de integración continua más extendidos.