IES Velázquez
Sevilla

IES Velázquez

Desarrollo de Aplicaciones Web

Desarrollo Web en Entorno Servidor

Ciclo Formativo de Grado Superior

Junta de Andalucía
Consejería de Desarrollo Educativo y Formación Profesional

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES

Plan de Recuperación, Transformación y Resiliencia

Financiado por la Unión Europea
NextGenerationEU

ÍNDICE: Tema1. Arquitecturas y tecnologías de programación web

1. Modelos De Ejecución De Código En Entornos Cliente/Servidor
    - 1.1. Modelo cliente-servidor
    - 1.2. Modelo de desarrollo de 3 capas
    - 1.3. El modelo MVC
    - 1.4. Arquitecturas y plataformas.
        - Arquitecturas FÍSICAS multicapa (multitier)
        - Arquitecturas LÓGICAS multicapa (multilayer)
        - Ventajas de las arquitecturas multicapa

Modelos De Ejecución De Código En Entornos Cliente/Servidor

En este apartado vamos a **estudiar la arquitectura de las aplicaciones web**. *Tanto* las que funcionan con la arquitectura *cliente-servidor de dos niveles como* la más profesional, de *tres niveles, que nos permite centralizar toda la información corporativa en el servidor de base de datos.* Tanto unas como otras hacen uso de clientes livianos, que *no necesitan un hardware potente para poder ejecutar el navegador web.*

RECORDEMOS QUE: Los diferentes ***clientes y servidores web*** requieren de otros ***protocolos de la familia TCP/IP***, *como el DNS*, para averiguar la IP pública de cualquier dominio que queramos consultar o con el que queramos trabajar. *Y que nos dirán cuál es el servidor web de ese dominio al que deberemos pedir la información de nuestras páginas web estáticas o dinámicas*, que se construirán en el momento, a partir del lenguaje de la parte de servidor que estemos utilizando.

Modelo cliente-servidor

El **modelo de desarrollo web** se apoya, en una primera aproximación **desde un punto de vista centrado en el hardware,** en lo que se conoce como **arquitectura Cliente / Servidor**, que define un **patrón de arquitectura donde existen dos actores**, *cliente y servidor*, de forma que *el primero es quién se conecta con el segundo para solicitar algún servicio.*

> *El cliente* es la parte de la aplicación con la que i*nteractúa el usuario final.* Por lo general, s*e ejecuta en un navegador web* y *puede ser una aplicación web o una aplicación móvil.* El cliente solicita recursos o servicios al servidor y presenta los resultados al usuario.
>
> *El servidor* es el componente central que *almacena los datos y la lógica de la aplicación*. *Responde* a las solicitudes del cliente *proporcionando recursos, procesando datos o realizando cálculos*. *Utiliza protocolos de comunicación para interactuar con el cliente, como HTTP en el caso de la web.*
>
> > El servidor web es un software que recibe peticiones HTTP (GET, POST, DELETE, ...) y devuelve el recurso solicitado (HTML, CSS, JS, JSON, imágenes, etc...).

En el caso que nos ocupa (el modelo cliente-servidor), l**os clientes solicitan que se les sirva una web para visualizarla,** aunque t**ambién es posible solicitar información si hablamos del caso de los servicios web** que también veremos más adelante. En cualquier caso, en ambos casos aparece el mismo **escenario,** donde un **servidor** se encuentra **ejecutándose ininterrumpidamente a la espera de que los diferentes clientes realicen una solicitud.**

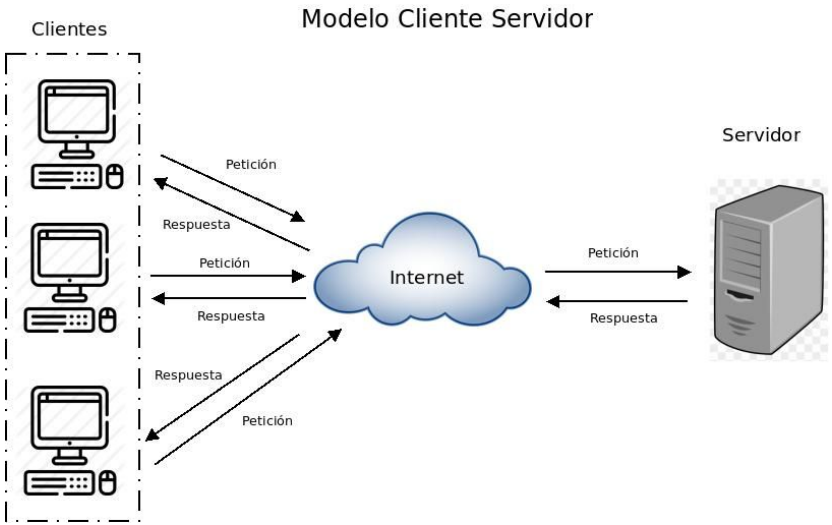
Modelo cliente-servidor

1. El cliente se conecta al servidor para solicitar algún servicio. *Petición .
2. El servidor se encuentra en ejecución de forma ininterrumpida a la espera de que los diferentes clientes realicen una solicitud. *Respuesta .
3. Se suele tratar de obtener el contenido de una página web. También podemos hablar de servicios web donde no se generan páginas web sino contenido en XML o JSON para ser consumido por una aplicación cliente.

La solicitud que hacen los clientes al servidor se le llama petición (request)

Lo que el servidor devuelve a dicho cliente le llamamos respuesta (response).

Estos términos son los usados por el protocolo HTTP y los encontraréis en cualquier lenguaje de desarrollo de aplicaciones web.



También hay que tener en cuenta que **esta arquitectura cliente-servidor** plantea la **posibilidad de numerosos clientes atendidos por un mismo servidor.** El **servidor será un software multitarea c**apaz de atender peticiones simultáneas de numerosos clientes.

Cuando una aplicación o servicio web tiene muchas solicitudes por unidad de tiempo puede ser que un conjunto de servidores o cluster desempeñen este servicio en equipo. O un software multitarea que será capaz de atender peticiones simultáneas de numerosos clientes.

Modelo cliente-servidor

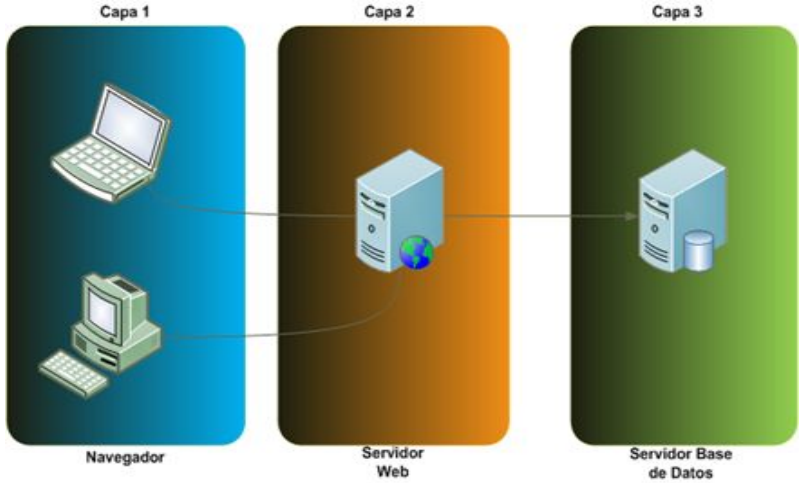
Ventajas	Desventajas
• **Puede ser más escalable,** tanto verticalmente como horizontalmente, si lo tenemos en la nube en la parte servidor. Por tanto, podemos aumentar el servicio que podemos ofrecer a más clientes. • **Mayor control,** al estar c*entralizado en la parte servidor,* lo que nos permite tener un *mantenimiento más cómodo y fácil* solo en esta parte. • Las **tecnologías para los servidores** de aplicaciones web han **evolucionado** para ofrecer una experiencia de usuario más amigable [UX/UI], **facilitando su uso**, **sin olvidar la seguridad al utilizar transacciones y sistemas de bloqueo** sobre nuestros datos.	• Al incrementar el tráfico en la red, requiere **tolerancia a fallos**, **añadiendo más hardware** con otro servidor secundario, que es un espejo del primario.

Modelo de desarrollo de 3 capas

Desde un punto de vista de desarrollo *una aproximación más detallada para este modelo de ejecución* es lo que se conoce como **modelo en 3 capas**.

Es un modelo más profesional donde se muestra más en detalle cómo se distribuye el software que participa en cualquier desarrollo 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 cómo interactúan las diferentes tecnologías o aplicaciones.

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



Modelo de desarrollo de 3 capas

Capa de presentación (nivel 1): Es el **front-end,** la capa que muestra la **interfaz de usuario y permite interactuar con el sistema al usuario**. Básicamente es la GUI (*Graphical User Interface, Interfaz Gráfica de Usuario*). *En el caso de una aplicación web sería el código HTML* que se carga directamente en el navegador web. En cualquier caso, *se ejecuta directamente en el equipo del cliente.* Ha de *verse bien en todos los navegadores* existentes del mercado de manera adaptativa [*responsive*] con diferentes tamaños de pantalla, *en todo tipo de dispositivos electrónicos,* desde wearables hasta smartTV, pasando por tabletas u ordenadores personales.

Capa de negocio (nivel 2): Es el **back-end**, la capa intermedia donde **se lleva a cabo toda la lógica de la aplicación,** la capa que *conoce y gestiona las funcionalidades que esperamos del sistema o aplicación web*. Siempre *se ejecutará en el lado servidor.* **Conoce el** modelo o la lógica del negocio o **servicio que queremos dar**; corresponde al servidor de aplicaciones. 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 1. No obstante, *cada vez existen más tendencias de dividirlo en microservicios, serverless o servicios en la nube.*

Capa de datos (nivel 3): 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 equipo,* aunque también es posible que se tengan que separar por cuestiones de rendimiento. **La capa de datos sirve todas la información necesaria a la capa de negocio para llevar a cabo sus operaciones. Permite centralizar los datos**, y que estos sean accesibles desde más de uno o varios servidores web, y **puede tener un conjunto de servidores web** (clúster) **que puedan balancear la carga** si nuestra aplicación web recibe miles o millones de peticiones concurrentes cada segundo o minuto. **E incluso podemos disponer de bases de datos distribuidas** en varios servidores de datos por diferentes ubicaciones para que estos estén lo más cerca de los usuarios finales. Básicamente, su función es entregar los datos que le pida la capa intermedia, es decir, el servidor de aplicaciones.

Modelo de desarrollo de 3 capas

La **capa de presentación** no **se comunica** directamente con la capa de datos, sino que lo hace **a través de la capa de negocio**. *La capa de negocio recibe las acciones del usuario provenientes de la capa de presentación* y, en caso necesario, *se comunica con la capa de datos para consultar o modificar la base de datos.* Asimismo, **la capa de negocio traslada la salida de la capa de datos hacia la capa de presentación.**

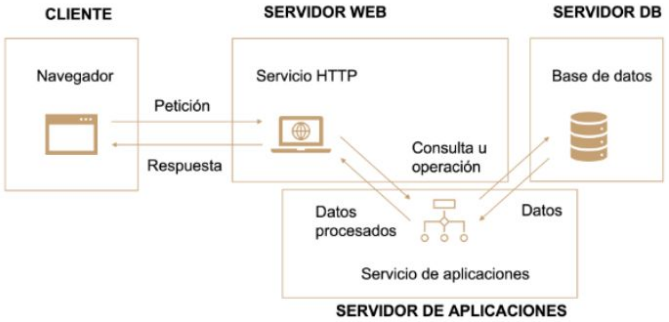
En una aplicación web, ***la capa de presentación*** *se ejecuta en un* ***navegador en el ordenador del usuario***, mientras que ***la capa de negocio*** *se encuentra en el* ***servidor web*** *y la* ***capa de datos*** *en el* ***servidor de bases de datos***. *Estas dos últimas pueden estar alojadas en equipos diferentes o en el mismo servidor*.

El **objetivo principal** es **desacoplar la lógica de negocio de la capa de presentación,** con el fin de lograr un **código más reutilizable.** Además, al separar los componentes se obtienen aplicaciones más fáciles de mantener y ampliar.

Si nos imaginamos 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**.

Modelo de desarrollo de 3 capas

La **capa lógica de negocio** también se puede expandir en dos capas: servidor web (la arquitectura) y servidor de aplicaciones (que se encarga de la programación).



Si nos centramos en un caso concreto con software y tecnologías ya definidos, un modelo de 3 capas podría ser el siguiente:

Navegador web: En este caso, Mozilla Firefox, Internet Explorer o Google Chrome podrían ser cualquiera de las aplicaciones que ocuparían esta capa.

Servidor web: Apache + PHP / IIS + ASP: Un servidor web acompañado de su correspondiente lenguaje de programación web permiten desarrollar la parte que ocupa la capa de negocio. También podríamos colocar la combinación Apache Tomcat + Servlets

Base de datos: MySQL/PostgreSQL: Finalmente en la capa de datos podemos colocar cualquier SGBD relacional o sistemas no relacionales como MongoDB.

Modelo de desarrollo de 3 capas

Ventajas	Desventajas
• Un **mayor** grado de **flexibilidad,** al delegar en cada capa diferentes tareas. • Una **mayor seguridad**, al definir la arquitectura *independientemente para cada servicio y en cada nivel.* • Un **mejor rendimiento**, ya que las tareas se comparten entre servidores. • Una **mayor reutilización** de *la capa de lógica intermedia por múltiples aplicaciones*, si tiene un buen diseño modular. • Una **mayor facilidad para cambiar o modificar parte de una capa** sin que se vean afectadas las restantes capas.	• Al incrementar el tráfico en la red, **requiere balancear la carga e introducir tolerancia a fallos.** • Los **navegadores actuales** pueden variar y **no ser compatibles al 100 %.** • La **estandarización** a nivel de tecnologías o proveedores **tarda** en evolucionar o desarrollarse.

El modelo MVC

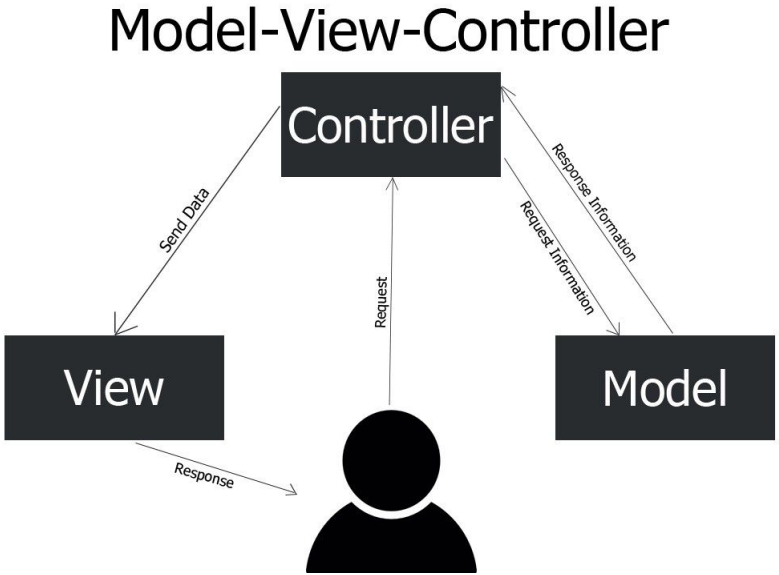
Model-View-Controller o **Modelo-Vista-Controlador** es un modelo de arquitectura que **separa los datos y la lógica de negocio respecto a la interfaz de usuario y el componente encargado de gestionar los eventos y las comunicaciones.**

Al *separar los componentes en elementos conceptuales* permite reutilizar el código y mejorar su organización y mantenimiento. Sus elementos son:

Modelo: representa la **información y gestiona todos los accesos a ésta**, tanto consultas como actualizaciones provenientes, normalmente, de una base de datos. Se accede vía el controlador.

Controlador: Responde a las acciones del usuario, **y realiza peticiones al modelo para solicitar información**. Tras recibir la respuesta del modelo, le envía los datos a la vista.

Vista: Presenta al usuario de forma visual el modelo y los datos preparados por el controlador. El usuario interactúa con la vista y realiza nuevas peticiones al controlador.



Pensamos

Respondemos a las siguientes preguntas

¿El aula virtual Moodle, qué arquitectura seguirá?. ¿Y Discord?

Si uso JS (JavaScript) para comprobar el DNI en un formulario, ¿en qué capa estaría trabajando?

Si uso PHP para insertar el DNI en la BD, ¿en qué capa estaría trabajando?

El modelo MVC

Ejemplo de Arquitectura MVC (Modelo-Vista-Controlador)
Modelo (Model): Aquí se define la lógica del negocio y la interacción con la base de datos. En este caso, el modelo se encargará de acceder a la base de datos y obtener la información de los usuarios, actualizar datos, crear nuevos registros y eliminarlos.
Ejemplo:php

```
class UserModel {
    public function getAllUsers() {
        // Aquí se hace una consulta a la base de datos
        $query = "SELECT * FROM users";
        return $this->executeQuery($query);
    }
}
```

***Vista (View)**: Es la parte que el usuario final ve e interactúa con. Es la interfaz gráfica que presenta la información obtenida del modelo, como un formulario para crear un nuevo usuario o una tabla para listar los usuarios.*
Ejemplo, html

```
<table>
    <tr><th>ID</th><th>Nombre</th><th>Email</th></tr>
    <?php foreach($users as $user): ?>
        <tr><td><?= $user['id'] ?></td><td><?= $user['name'] ?></td><td><?= $user['email'] ?></td></tr>
    <?php endforeach; ?>
</table>
```

***Controlador (Controller):** Es el intermediario entre el modelo y la vista. Recibe las solicitudes del usuario (por ejemplo, a través de un formulario), se comunica con el modelo para recuperar o modificar los datos, y luego actualiza la vista con la información correspondiente.*
Ejemplo,php

```
class UserController {
    public function listUsers() {
        $userModel = new UserModel();
        $users = $userModel->getAllUsers();
        include 'views/userListView.php';
    }
}
```

El modelo MVC

Comparación entre la Arquitectura MVC y la Arquitectura en 3 Capas
Separación de responsabilidades: Ambas arquitecturas separan responsabilidades, pero **MVC** está más **orientada a la interacción directa entre la presentación, el controlador y el modelo**, mientras que la **arquitectura en 3 capas** tiende a s**eparar más claramente la lógica de negocio del acceso a datos**.

Flexibilidad: La **arquitectura en 3 capas permite una mayor modularidad y separación física entre las capas** (por ejemplo, tener una capa de datos en un servidor separado). **MVC** tiende a ser más **utilizada dentro de un solo entorno de aplicación.**

Comunicación: En **MVC**, el **controlador** es quien **gestiona la comunicación entre el modelo y la vista**. En la arquitectura en **3 capas**, las capas se **comunican** generalmente a **través de interfaces o APIs** definidas entre ellas.

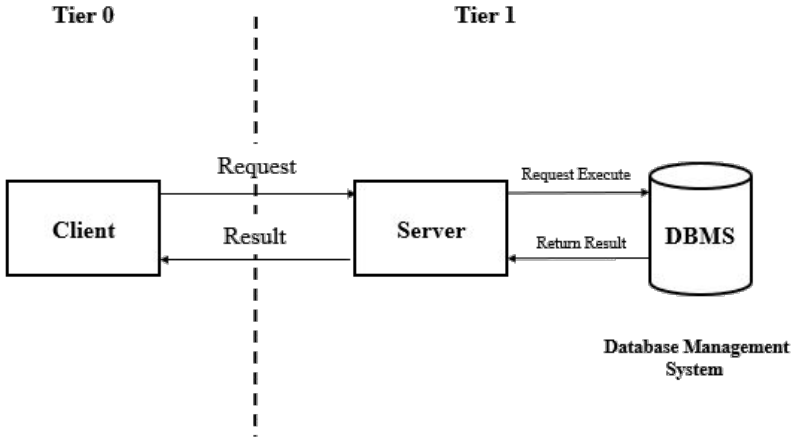
Arquitecturas y plataformas.

Las **arquitecturas web definen** la forma en que **las páginas** de un sitio web están **estructuradas y enlazadas** entre sí.
Pero usamos este término "**arquitectura de una aplicación**" *para referirnos a* dos cosas distintas: **la arquitectura física y la arquitectura lógica.** En este apartado aprenderemos a distinguir entre capas físicas (tier) y capas lógicas (layer).

Arquitecturas FÍSICAS multicapa (multitier)

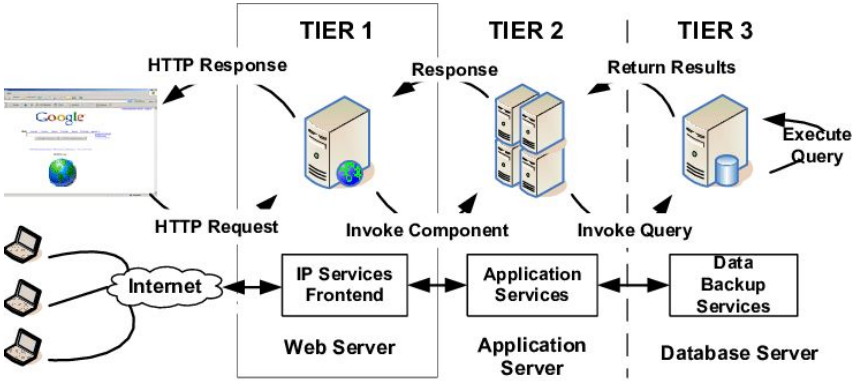
Nos referimos a la **capa física de una arquitectura** a cómo está construida la infraestructura (**el hardware**). Supone un nuevo elemento hardware separado físicamente. *Las capas físicas más alejadas del cliente están más protegidas, tanto por firewalls como por VPN.*

Una a**rquitectura física de varios niveles** (multinivel o *multitier*, en inglés) **consiste en un conjunto de ordenadores conectados a una red que ejecutan de forma conjunta una aplicación.**
El e*jemplo* más sencillo es la a*rquitectura cliente-servido*r, la más popular en aplicaciones web sencillas: una máquina cliente y una máquina servidor ejecutan alternativamente fragmentos del código, proporcionando al usuario final la sensación de una aplicación unificada.



Arquitecturas FÍSICAS multicapa (multitier)

Por supuesto, **nada impide que tengamos más de dos máquinas colaborando en red para ejecutar una aplicación web.** Podemos tener, por ejemplo, un cliente, un servidor web y un servidor de bases de datos (estos dos últimos en dos máquinas físicas diferentes). Esto sería una arquitectura de 3 niveles físicos.
Esta arquitectura se puede generalizar. Una con N niveles físicos tendría este aspecto:

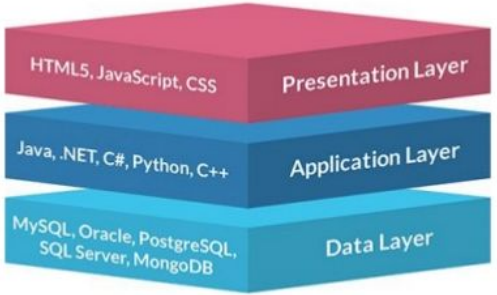


La arquitectura física es algo que **incumbe sobre todo a los administradores de sistemas**, que son los encargados de montarla, configurarla y mantenerla. **Para el programador,** sin embargo, suele resultar **transparente.**
Por ejemplo, cuando nos conectamos a un servidor de bases de datos desde nuestra aplicación, poco importa que ese servidor esté en la misma máquina física o en otra máquina diferente: la forma de conectarse y operar con la base de datos es exactamente la misma.
Así que la arquitectura física, siendo importante, no lo es demasiado para nosotros y nosotras como desarrolladores.

Arquitecturas LÓGICAS multicapa (multilayer)

Nos referimos a la a**rquitectura lógica** *a cómo está construido el servicio (***el software***).*
Las capas lógicas (layers) organizan el código respecto a su funcionalidad:

- Presentación.
- Negocio / Aplicación / Proceso.
- Datos / Persistencia.



L**a arquitectura de una aplicación también puede referirse a sus capas** (layers en inglés) l**ógicas** y consiste en *dividir una aplicación en capas que colaboran entre sí por medio de interfaces bien definidos.*
La idea es fragmentar nuestra aplicación en capas de niveles de abstracción cada vez mayor. **En un extremo**, la **capa más abstracta** es la que interacciona con el usuario: ahí se implementará nuestro interfaz de usuario, o lo que en aplicaciones web se llama **front-end.**
En el **otro extremo**, **la capa menos abstracta** es la que está e**n contacto con el hardware** de la máquina. Bueno, *en el caso de una aplicación web, no hay contacto directo con el hardware, sino con otras capas aún menos abstractas que están fuera de nuestra aplicación, como el sistema operativo, el servidor web o el gestor de bases de datos*. Esas capas externas a nuestra aplicación son las que, realmente, interaccionan con el hardware en última instancia.

Ventajas de las arquitecturas multicapa

- Las arquitecturas multicapa permiten varias cosas que no pueden hacerse con los códigos monolíticos. Entre otras:
- Desarrollar en paralelo cada capa (mayor rapidez de desarrollo).
- Aplicaciones más robustas gracias al encapsulamiento. ¿Te suena? ¡Programación orientada a objetos! Cada capa se implementa en una clase, y cada clase hace su trabajo sin importunar a las demás y sin preocuparse por cómo funcionan las otras internamente.
- El mantenimiento es más sencillo.
- Más flexibilidad para añadir módulos.
- Más seguridad, al poder aislar (relativamente) cada capa del resto.
- Mejor escalabilidad: es más fácil hacer crecer al sistema.
- Mejor rendimiento (aunque esto podría discutirse: puedes hacer un sistema multicapa con un rendimiento desastroso y un sistema monolítico que vaya como un tiro. Pero, en general, es más fácil mejorar el rendimiento trabajando en cada capa por separado).
- Es más fácil hacer el control de calidad, incluyendo la fase de pruebas.

El **único inconveniente** reseñable de las arquitecturas multicapa es que **pueden hacer que una aplicación muy simple se vuelva artificialmente más compleja de lo necesario.**

Pensamos

Completa los huecos con el término correcto.

En referencia a las diferentes capas de la arquitectura de tres capas, vamos a centralizar nuestra información en la capa de (1)_____, donde nuestro SGBDR los almacenará. Sobre esta, accederá nuestra capa de (2)_____, que es el back-end, es decir, la capa que conoce y gestiona las funcionalidades que esperamos del sistema o aplicación web; conoce el modelo de negocio o servicio que queremos dar, aunque cada vez existen más tendencias de dividirlo en microservicios, serverless o servicios en la nube. Y, por último, los usuarios accederán a la capa de (3)_____, que es el front-end que ven y utilizan, y que ha de verse bien en todos los navegadores adaptativos (responsive) existentes, con diferentes tamaños de pantalla.

¿Cuáles son las ventajas de la arquitectura de tres capas?
Un mejor rendimiento, ya que las tareas se comparten entre servidores.
Los navegadores actuales pueden variar y no ser compatibles al 100 %.
Una mayor facilidad para cambiar o modificar parte de una capa sin que se vean afectadas las restantes capas.
La estandarización en lo relativo a tecnologías o proveedores tarda en evolucionar o desarrollarse.

Verdadero o Falso
Las aplicaciones web necesitan utilizar los protocolos de la familia TCP/IP, como DNS, HTTP y HTTPS, entre otros.
Las aplicaciones web en producción más profesionales están desplegadas en dos capas.