INVESTIGACIÓN DE ARQUITECTURAS DE CLIENTE SERVIDOR.

1. **Modelo de dos capas.**

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, que le da respuesta

Gran parte de la aplicación corre en el lado del cliente (fat client). Los **componentes** de las capas son:

**Cliente (fat client):** La lógica de negocio está inmersa dentro de la aplicación que realiza el interfaz de usuario, en el lado del cliente.

**Servidor:** Administra los datos.

**Características**

En la arquitectura C/S el remitente de una solicitud es conocido como **cliente**. Sus **características** son:

* Es el que inicia solicitudes o peticiones. Tiene, por tanto, un papel activo en la comunicación (dispositivo maestro o amo).
* Espera y recibe las respuestas del servidor.
* Por lo general, puede conectarse a varios servidores a la vez.
* Normalmente, interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.

Al receptor de la solicitud enviada por el cliente se conoce como **servidor**.

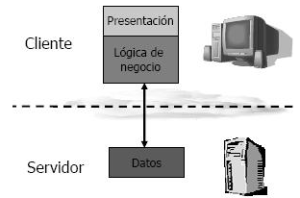
Sus **características** son:

* Al iniciarse espera a que le lleguen las solicitudes de los clientes. Desempeñan entonces un papel pasivo en la comunicación (dispositivo esclavo).
* Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
* No es frecuente que interactúen directamente con los usuarios finales.

**Ventajas**

* **Centralización del control:** los accesos, recursos y la integridad de los datos son controlados por el servidor, de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.
* **Escalabilidad:** se puede aumentar la capacidad de clientes y servidores por separado.
* **Tecnologías:** existen algunas suficientemente desarrolladas, diseñadas para el paradigma de C/S, que aseguran la seguridad en las transacciones, la amigabilidad de la interfaz y la facilidad de empleo.

**Las desventajas de este modelo son:**



* Es difícilmente escalable
* Número de conexiones reducida
* Alta carga de la red.
* La flexibilidad es restringida
* La funcionalidad es limitada.

EJEMPLO: Cualquier aplicación que sea capaz de comunicarse con la base de datos directamente

1. **Modelo de tres capas.**

Es un tipo de arquitectura usada en la gran mayoría de sistemas. Se suele usar en sistemas que implementan un modelo de negocio como podría ser una tienda online, una aplicación para gestionar ciertos datos, etc. En la arquitectura en tres niveles existe un nivel intermedio. Esto significa que la arquitectura generalmente está compartida por:

**Componentes:**

Un cliente, es decir, el equipo que solicita los recursos, equipado con una interfaz de usuario (generalmente un navegador web) para la presentación.

El servidor de aplicaciones (también denominado software intermedio), cuya tarea es proporcionar los recursos solicitados, pero que requiere de otro servidor para hacerlo.

El servidor de datos, que proporciona al servidor de aplicaciones los datos que éste le solicitó.

**Características:**

* Centraliza la gestión de la reglas del negocio en un único lugar (no se duplica en cada aplicación)
* Los clientes pidan o envíen información a esta aplicación centralizada, no al gestor de base de datos en el servidor
* Esta aplicación centralizada que conforma una nueva capa dentro de un sistema Cliente-Servidor, se conoce como capa intermedia o middle-tier.

**Ventajas:**

* Un mayor grado de flexibilidad.
* Mayor seguridad, ya que la seguridad se puede definir independientemente para cada servicio y en cada nivel.
* Mejor rendimiento, ya que las tareas se comparten entre servidores.

**Desventajas:**

* Pueden incrementar el tráfico en la red cuando muchos clientes envían peticiones a un solo servidor.
* Requiere más balance de carga y tolerancia a las fallas.
* Los exploradores actuales no son todos iguales.

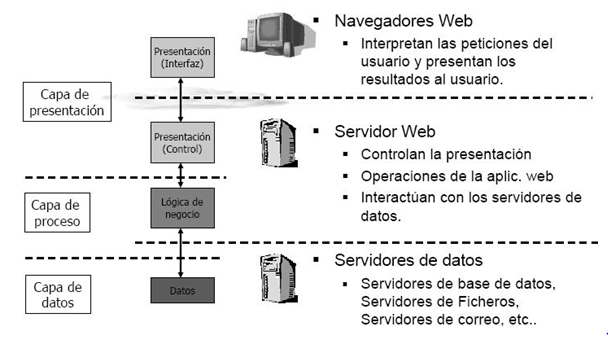
**Capa de presentación**  
Referente a la interacción entre el usuario y el software. Su principal responsabilidad es mostrar información al usuario, interpretar los comandos de este y realizar algunas validaciones simples de los datos ingresados.

**Capa de Negocio**

Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían.

**Capa de datos**

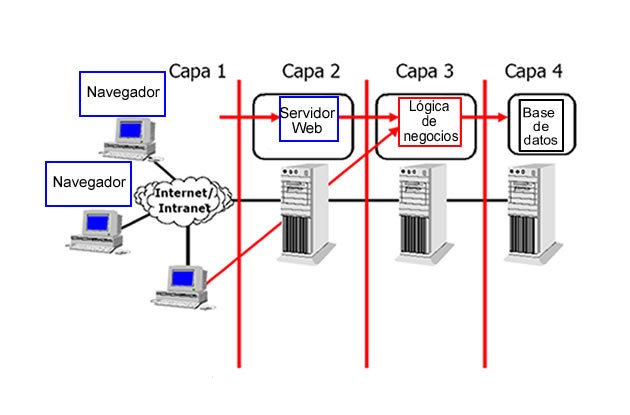
Esta capa es la encargada de almacenar los datos del sistema y de los usuarios. Su función es almacenar y devolver datos a la capa de negocio, esta capa es la única que puede acceder a los mismos datos las respuestas tras el proceso.



**Modelo de n capas.**

En una aplicación distribuida en n capas, los diferentes procesos están distribuidos en diversas capas, no solo lógicas, sino también físicas, los procesos se ejecutan en diferente equipo.

**Componentes:**

****

**Ventajas:**

* Desarrollos paralelos (en cada capa).
* Aplicaciones más robustas debido al encapsulamiento.
* Mantenimiento y soporte más sencillo (es más sencillo cambiar un componente que modificar una aplicación monolítica).
* Mayor flexibilidad (se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad).
* Alta escalabilidad.

**Desventajas:**

* Pone más carga en la red, debido a una mayor cantidad de tráfico de la red.
* Es mucho más difícil programar y probar el software que en arquitectura de dos niveles porque tienen que comunicarse más dispositivos para terminar la transacción de un usuario.

EJEMPLO: En alguna aplicación con compras en línea, como un videojuego al vender paquetes de contenido donde primero se realiza la transacción y luego va a la base de datos

1. **Arquitectura de tres capas con tecnología de monitoreado en el procesamiento de transacciones**

La Arquitectura de 3 Capas Esta arquitectura, la más común en la actualidad, está compuesta por las siguientes capas:

* Presentación: comprende la lógica para manejar la interacción entre el usuario y la aplicación. Posibles opciones son la línea de comandos, sistemas de menús basados en texto, el cliente rico o la interfaz de usuario HTML.
* Fuentes de Datos: tiene que ver con la comunicación con otros sistemas que llevan a cabo tareas en nombre de la aplicación. Monitores de transacciones, otras aplicaciones, sistemas de mensajería,pero generalmente base de datos.
* Lógica de Dominio: la lógica de dominio o empresarial es la funcionalidad específica que la aplicación debe hacer para el dominio de trabajo. Los cálculos sobre la base de las entradas y los datos almacenados, la validación de los datos de entrada o determinar la fuente de datos implicada son ejemplos de funcionalidades en la capa de dominio.

Un aspecto difuso de esta arquitectura es lo que sucede cuando no hay una persona utilizando el software, por ejemplo un servicio web o un proceso por lotes (batch).

En este caso se difumina la diferencia entre presentación y datos, ya que las dos se refieren a conexiones con el exterior de la aplicación. La distinción a hacer es entre una interfaz que la aplicación proporciona como un servicio a otros, y el uso que se haga de los servicios proporcionados por otros como fuentes de datos.

**¿Dónde ejecutar las capas?**

La separación entre capas es útil incluso si se ejecutan en el mismo ordenador. De todas formas, para la mayoría de las aplicaciones la decisión es si ejecutar una parte del procesamiento en el cliente o en el servidor.

El caso más simple es ejecutar todo en el servidor, por ejemplo utilizando una interfaz HTML, lo que facilita la actualización y mantenimiento. Los argumentos en favor de ejecutar parte en el cliente se justifican básicamente por conseguir una mayor capacidad de respuesta o el funcionamiento sin estar conectado (offline). 3 Organización de la Lógica de Dominio El enfoque más sencillo para gestionar la lógica de dominio es el Transaction Script1 .

Es esencialmente un procedimiento que recibe los parámetros de entrada de la presentación, los procesa, almacena información en la base de datos, invoca operaciones de otros sistemas y responde con más datos a la presentación.

La organización fundamental es un procedimiento único para cada acción del usuario, por ejemplo un script. De todas formas no tiene porque ser un procedimiento de código autocontenido, podría haber subrutinas compartidas entre diferentes Transaction Scripts. Una tienda online tendría Transaction Scripts para añadir productos al carro de la compra, pasar por caja, mostrar el estado del pedido.

Ventajas:

* Modelo procedural simple
* Funciona bien con una capa de datos simple.
* Hace obvios los límites de la transacción: se abre al principio del procedimiento y se cierra al acabar.

**Desventajas:**

* Aparecen muchas a medida que la complejidad de la lógica de dominio aumenta.
* A menudo habrá código duplicado, difícil de eliminar.
* Una lógica compleja requiere el uso de objetos para definir un Domain Model. Aunque esta es la mejor elección, cuanto más rico sea el modelo de dominio más complejo será el mapeo de éste a la base de datos relacional.

La tercera opción para estructurar la lógica de domino es el Table Module, diseñado para funcionar con Record Set, el cual se obtiene como resultado de realizar consultas a la base de datos. El Table Module está a medio camino entre Transaction Script y Domain Model. Organizar la lógica de dominio en base a las tablas permite una mayor estructura que con los procedimientos de Transaction Script, pero carece de herencia, estrategias y otros patrones de orientación a objetos. La mayor virtud de Table Module es que encaja perfectamente en arquitecturas como .NET.

EJEMPLO: un ejemplo seria la app de banco azteca.