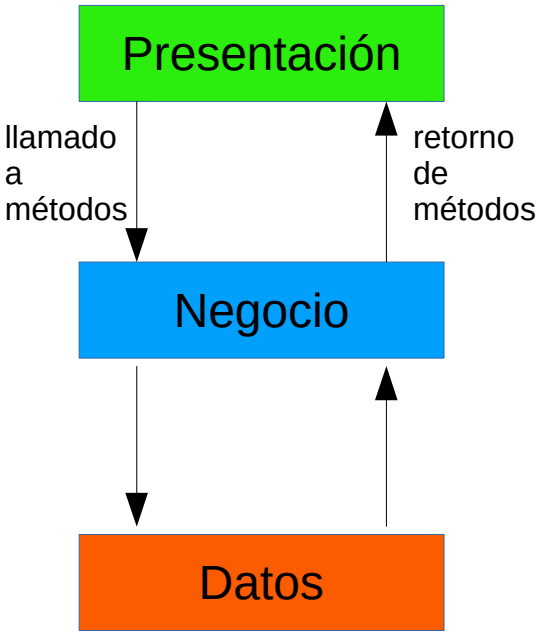
Arquitectura en capas

- La arquitectura de capas es un modelo de desarrollo software en el que el objetivo primordial es la separación (desacoplamiento) de las partes que componen un sistema software. Lo que propone es pensar nuestro sistema en capas, donde cada capa debe exponer en forma clara las operaciones que puede realizar. Estas operaciones se deben exponer mediante un api que nos digan qué servicio me ofrece esa capa y cuál será su retorno sin importar cómo este implementado.
- Pueden existir n capas, pero cada una debe tener una responsabilidad única. Una separación muy utilizada es la de tres capas “presentación”, “lógica de negocio” y “acceso a datos”.



Qué es una capa?

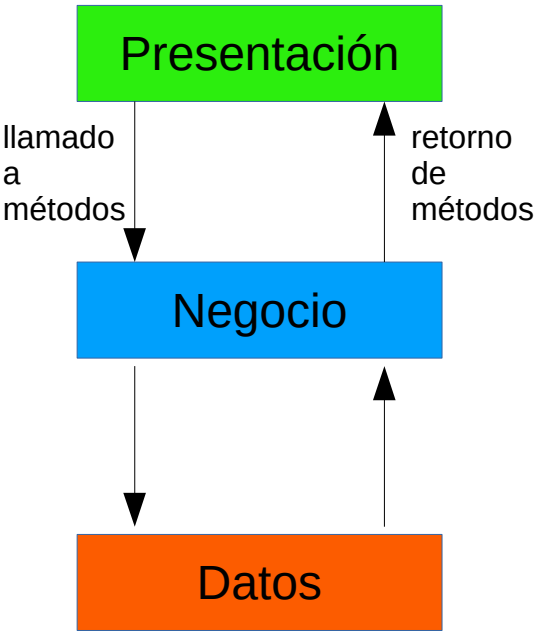
El término «capa» hace referencia a la forma como una solución es segmentada desde el punto de vista lógico.

En una arquitectura de tres niveles, los términos “capas” y “niveles” no significan lo mismo ni son similares.

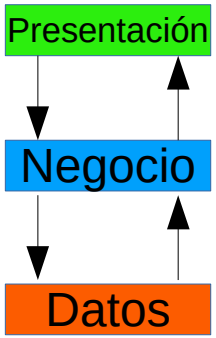
El término “capa” hace referencia a la forma como una solución es segmentada desde el punto de vista lógico: Presentación/ Lógica de Negocio/ Datos.

En cambio, el término “nivel” corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física. Por ejemplo:

- Una solución de tres capas (presentación, lógica del negocio, datos) que residen en una sola computadora (Presentación+lógica+datos). Se dice que la arquitectura de la solución es de tres capas y un nivel.
- Una solución de tres capas (presentación, lógica del negocio, datos) que residen en dos computadoras (presentación+lógica, lógica+datos). Se dice que la arquitectura de la solución es de tres capas y dos niveles.
- Una solución de tres capas (presentación, lógica del negocio, datos) que residen en tres computadoras (presentación, lógica, datos). La arquitectura que la define es: solución de tres capas y tres niveles.



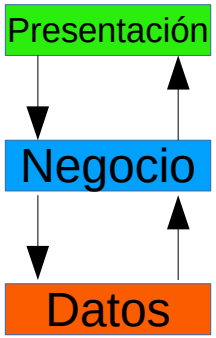
Capa de Presentación



Atiende los eventos del cliente y representa los datos para el mismo. Teniendo en cuenta que el cliente puede ser un humano u otro sistema, esta capa será encargada en caso del humano de atender los clics (u otros eventos) en un HTML y de renderizar la información de manera visual. En caso de que sea otro sistema puede atender peticiones rest y devolver información en un formato estructurado (json, xml, etc) que es más fácil de interpretar por un sistema.

Es la capa que ve el usuario (también se la denomina **«capa de usuario»**), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser «amigable» (entendible y fácil de usar) para el usuario. **Esta capa se comunica únicamente con la capa de negocio.**

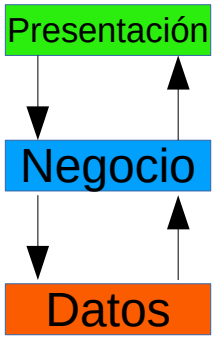
Capa de Lógica de Negocio



En esta capa se encuentra todo lo que refiere a las reglas que se encuentran en el negocio, o sea los requerimientos funcionales de nuestro sistema. Por ejemplo, si se tiene un alta de usuario esta capa debe proveer el medio para altaDeUsuario y dentro del método se debe realizar todos los pasos para dar de alta un usuario (enviar mail, validar nombre, etc).

Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa **se comunica con la capa de presentación**, para recibir las solicitudes y presentar los resultados, **y con la capa de datos**, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

Capa de Acceso a Datos



Mediante esta se pueden obtener o guardar los datos que utilizaran en la aplicación. No habla de cómo se realiza la persistencia, si es en base de datos o en archivo o etc, solo habla de acceso a datos. Por ejemplo, esta capa debería proveer un medio para poder guardar el usuario y otro para obtenerlo.

Es donde residen los datos y es la encargada de acceder a los mismos. Puede estar formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, **reciben solicitudes** de almacenamiento o recuperación de información **desde la capa de negocio.**

Reglas de comunicación entre las capas

Primera regla

- **Cada capa debe tener una responsabilidad única**. Es decir que las capas deben estar perfectamente delimitadas de que se ocupa cada una de ellas, por ejemplo, podemos tener una capa de “presentación” que será la encargada de atender los eventos del cliente y encargada de representar la información para el mismo. Por otro lado, podemos tener la capa de “acceso a datos” que será la encargada de guardar y acceder a los datos.

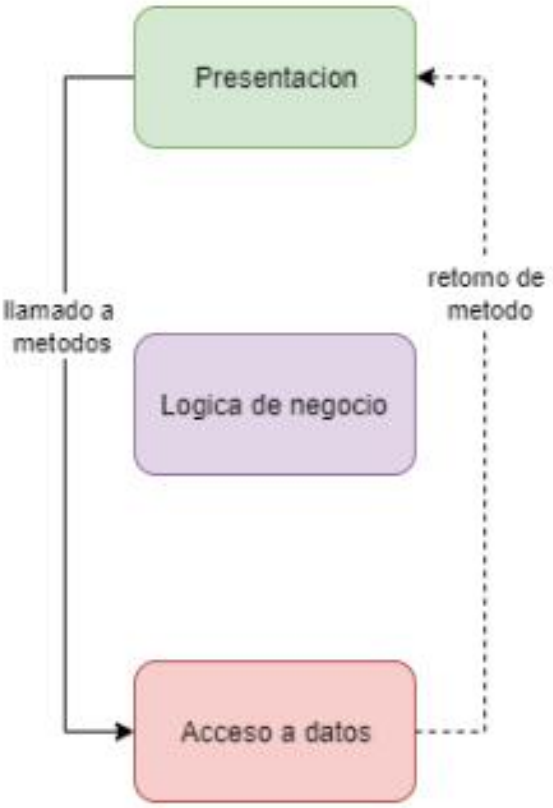
Reglas de comunicación entre las capas

Segunda regla

- **Las capas deben respetar una estructura jerárquica estricta**. Quiere decir que cada capa puede comunicarse sólo con la que está debajo suyo, pero NO al revés. Por ejemplo, una clase ubicada en la capa de presentación puede llamar a un método ubicado en la capa de acceso a datos, pero nunca la capa de acceso a datos puede llamar a un método de la capa de presentación. Y cuando nos referimos a la próxima más baja significa que no se puede saltar capas. Veamos un ejemplo gráfico:

Para este ejemplo utilizaremos la separación de tres capas “presentación”, “lógica de negocio” y “acceso a datos”.

Esto es lo que se debería hacer y lo que no se debería hacer.



Diferencias con MVC

La Arquitectura MVC puede implementarse como una relación triangular es decir que hay una relación entre la vista y el controlador, entre el controlador y el modelo y entre la vista y el modelo.

La Arquitectura a 3 Capas, es lineal es decir que no hay una comunicación directa entre las diferentes capas. Por ejemplo la capa del cliente no se comunica directamente con la capa de datos, todas las comunicaciones deben pasar por una capa intermedia.

Diferencias con MVC

La Arquitectura de 3 capas es un Estilo arquitectónico y MVC es un Patrón de diseño,pero podríamos utilizar el patrón mvc en el estilo de la arquitectura de 3 capas. Así que:

- Nivel de presentación : "Controladores y vistas" del patrón MVC.
- Nivel de negocio : "Modelo(Datos)" del patrón MVC.
- Nivel de acceso a los datos : Nivel de acceso a los datos original.

Diferencias con MVC

Qué hay que cambiar para ir de 3 niveles a MVC?

Hay que cambiar la capa de presentación.

Si el programa está bien hecho en una estructura multicapa, de forma que toda la lógica del programa se encuentre fuera de la capa de presentación, y las capas inferiores no contengan ninguna dependencia de la capa de presentación, entonces debería ser relativamente simple la conversión.

Suponiendo que la versión anterior también sea de tipo Web (no una aplicación de escritorio), entonces el diseño del HTML de las páginas así como los estilos y el código javascript que haya en las páginas se podrán en general reutilizar para construir las Vistas de MVC. Lo que sí que habrá que cambiar es todos los controles de servidor que se estuvieran utilizando en la versión anterior, que no valen para MVC, y la lógica de funcionamiento de los postbacks, que ahora ya no se basan en eventos sino en los métodos de acción del Controlador (que serán los que luego internamente llamen a la capa de reglas de negocio para implementar la lógica de la aplicación).

Si la estructura multicapa estaba basada en clases que se transmiten desde la capa de reglas de negocio a la capa de presentación para encapsular los datos intercambiados entre ambas, entonces posiblemente en muchos casos esas mismas clases sean válidas para usarlas como Modelo en el programa MVC, así que también en este sentido es posible que se pueda reutilizar gran parte del código existente. Todo depende de qué tan bien hecha esté la estructura multicapa.

Tema pendiente: Cuándo validar, lanzar excepciones, establecer un contrato?

Luego de ver el tema Excepciones se analizará la forma de determinar lo conveniente para un método en cuanto a:

- Validar un dato de entrada o el estado del objeto
- Lanzar una excepción
- Establecer condiciones esperadas en base al contato