udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA Y CIENCIAS

Tutoría para Examen de Grado

Sesión 3: Arquitecturas y Patrones de Diseño de Software

Docente: Mauricio Hidalgo Barrientos

Temario de la sesión

- Recordar algunos estilos Arquitectónicos de Software
- Identificar algunos patrones de Diseño de Software

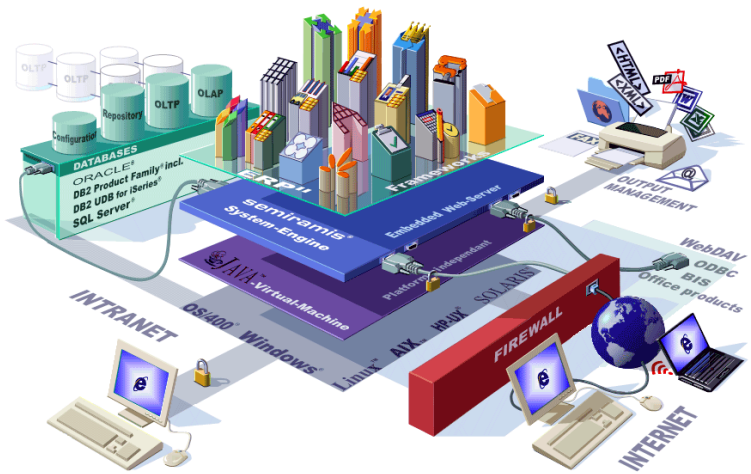
ARQUITECTURAS DE SOFTWARE

Arquitectura de Software

- Según el Software Engineering Institute (SEI), la arquitectura de software hace referencia a "las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos”.

- La arquitectura de software se refiere a las partes que vamos a construir y la forma en la que las vamos a combinar y juntar para poder trabajar con ellas.

- Veamos los estilos de arquitectura de SW más conocidos.



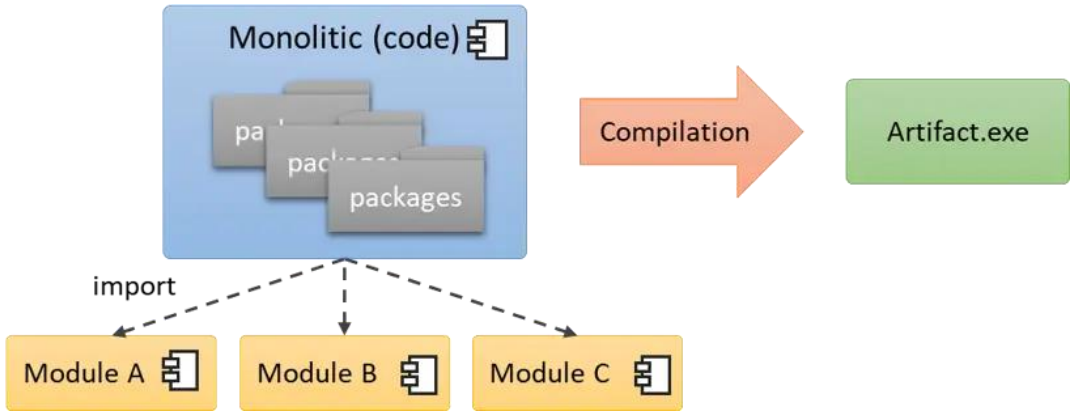
Arquitectura Monolítica

El estilo arquitectónico monolítico consiste en:

- Crear una aplicación autosuficiente que contenga absolutamente toda la funcionalidad necesaria para realizar la tarea para la cual fue diseñada.

Se basa en crear una aplicación que sea:

- Completamente autosuficiente y
- que abarque toda la funcionalidad requerida para su tarea,
- sin depender de componentes externos para complementar su funcionalidad.



Características de la Arq. Monolítica

Componentes integrados

En este enfoque, los componentes de la aplicación trabajan de manera conjunta y comparten recursos y memoria, lo que significa que están altamente integrados.

Unidad cohesiva de código

Una aplicación monolítica se presenta como una única y cohesiva unidad de código, donde todas las partes de la aplicación están combinadas en un solo programa o sistema.

IMPORTANTE

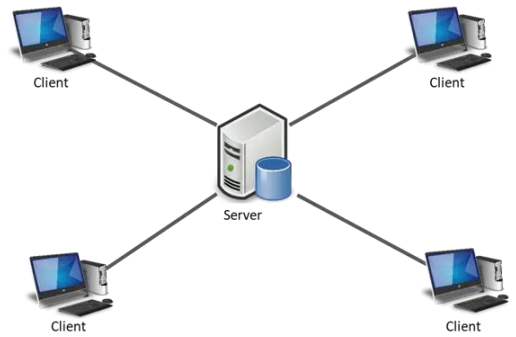
Debido a que todo el software es una sola pieza, implica que utilicemos el mismo Stack tecnológico para absolutamente todo, lo que impide que aprovechemos todas las tecnologías disponibles.

Arquitectura Cliente-Servidor

- Cliente-Servidor es uno de los estilos arquitectónicos distribuidos más conocidos, el cual está compuesto por dos componentes, el proveedor y el consumidor.
- El proveedor es un servidor que brinda una serie de servicios o recursos los cuales son consumido por el Cliente.

En una arquitectura Cliente-Servidor:

- Existe un servidor y múltiples clientes que se conectan al servidor para recuperar todos los recursos necesarios para funcionar, en este sentido,
- El cliente solo es una capa para representar los datos y se detonan acciones para modificar el estado del servidor, mientras que el servidor es el que hace todo el trabajo pesado.



Ventajas de la Arq. Cliente-Servidor

- Centralización

El servidor fungirá como única fuente de la verdad, lo que impide que los clientes conserven información desactualizada.

- Seguridad

El servidor por lo general está protegido por firewall o subredes que impiden que los atacantes pueden acceder a la base de datos o los recursos sin pasar por el servidor.

- Fácil de instalar (cliente)

El cliente es por lo general una aplicación simple que no tiene dependencias, por lo que es muy fácil de instalar.

Ventajas de la Arq. Cliente-Servidor

- Separación de responsabilidades

La arquitectura cliente-servidor permite implementar la lógica de negocio de forma separada del cliente.

- Portabilidad

Una de las ventajas de tener dos aplicaciones es que podemos desarrollar cada parte para correr en diferentes plataformas, por ejemplo, el servidor solo en Linux, mientras que el cliente podría ser multiplataforma.

Desventajas de la Arq. Cliente-Servidor

- Actualizaciones (clientes)

Una de las complicaciones es gestionar las actualizaciones en los clientes, pues puede haber muchos terminales con el cliente instalado y tenemos que asegurar que todas sean actualizadas cuando salga una nueva versión.

- Concurrencia

Una cantidad no esperada de usuarios concurrentes puede ser un problema para el servidor, quien tendrá que atender todas las peticiones de forma simultánea, aunque se puede mitigar con una estrategia de escalamiento, siempre será un problema que tendremos que tener presente.

- Todo o nada

Si el servidor se cae, todos los clientes quedarán totalmente inoperables.

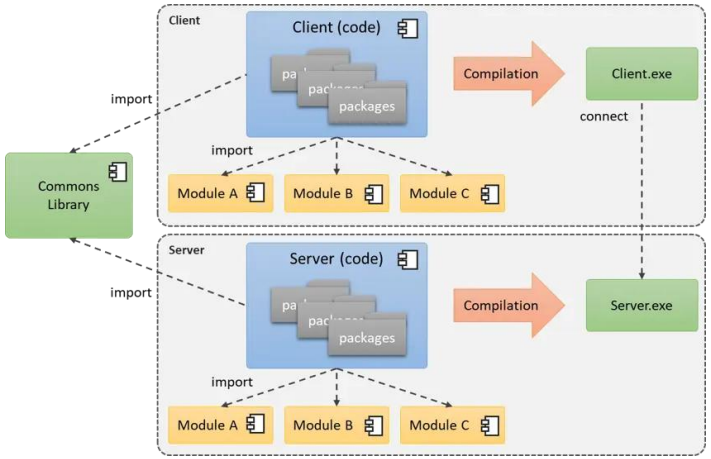
Desventajas de la Arq. Cliente-Servidor

- Protocolos de bajo nivel

Los protocolos más utilizados para establecer comunicación entre el cliente y el servidor suelen ser de bajo nivel, como Sockets, HTTP, RPC, etc. Lo que puede implicar un reto para los desarrolladores.

- Depuración

Es difícil analizar un error, pues los clientes están distribuidos en diferentes máquinas, incluso, equipos a los cuales no tenemos acceso, lo que hace complicado recopilar la traza del error.



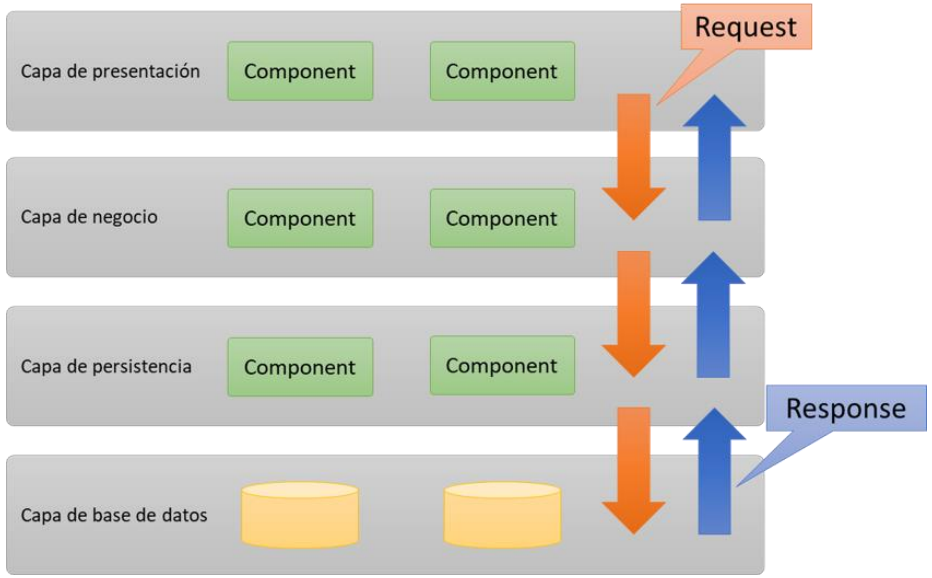
Arquitectura en capas

La arquitectura en capas consta en dividir la aplicación en capas, con la intención de que cada capa tenga un rol muy definido:

Capa de presentación (UI)

Capa de reglas de negocio (servicios)

Capa de acceso a datos (DAO)



Sin embargo, este estilo arquitectónico no define cuantas capas debe de tener la aplicación, sino más bien, se centra en la separación de la aplicación en capas (Aplica el principio Separación de preocupaciones (SoC)).

Ventajas de la Arquitectura en capas

- Separación de responsabilidades

Permite la separación de preocupaciones (SoC), ya que cada capa tiene una sola responsabilidad.

- Fácil de desarrollar

Este estilo arquitectónico es especialmente fácil de implementar, además de que es muy conocido y una gran mayoría de las aplicaciones la utilizan.

- Fácil de probar

Debido a que la aplicación construida por capas es posible ir probando de forma individual cada capa, lo que permite probar por separada cada capa.

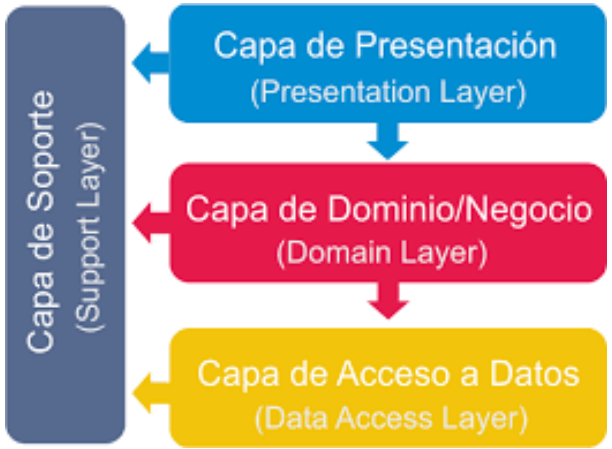
Ventajas de la Arquitectura en capas

- Fácil de mantener

Debido a que cada capa hace una tarea muy específica, es fácil detectar el origen de un bug para corregirlo, o simplemente se puede identificar donde se debe aplicar un cambio.

- Seguridad

La separación de capas permite el aislamiento de los servidores en subredes diferentes, lo que hace más difícil realizar ataques.



Desventajas de la Arquitectura en capas

- Rendimiento

La comunicación por la red o internet es una de las tareas más tardadas de un sistema, incluso, en muchas ocasiones más tardado que el mismo procesamiento de los datos, por lo que el hecho de tener que comunicarnos de capa en capa genera un degrado significativo en el performance.

- Escalabilidad

Las aplicaciones que implementan este patrón por lo general tienda a ser monolíticas, lo que hace que san difíciles de escalar, aunque es posible replicar la aplicación completa en varios nodos, lo que provoca un escalado monolítico.

- Complejidad de despliegue

En este tipo de arquitecturas es necesarios desplegar los componentes de abajo arriba, lo que crea una dependencia en el despliegue, además, si la aplicación tiende a lo monolítico, un pequeño cambio puede requerir el despliegue completo de la aplicación.

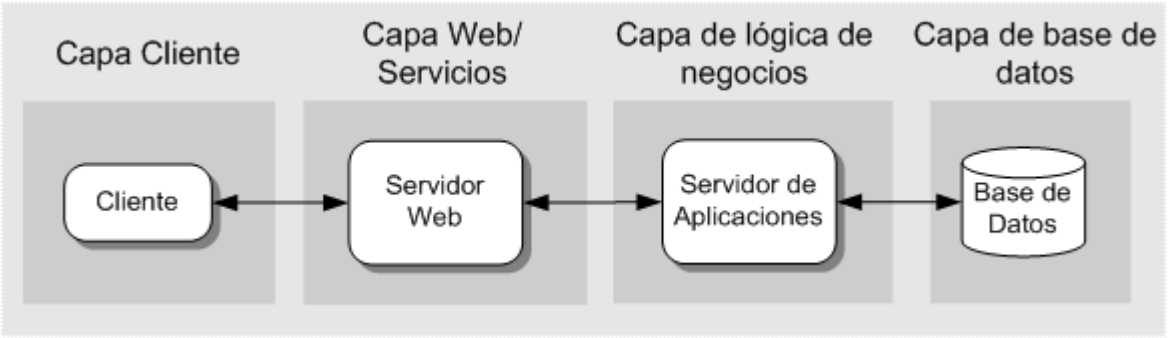
Desventajas de la Arquitectura en capas

- Anclado a un Stack tecnológico

Si bien no es la regla, la realidad es que por lo general todas las capas de la aplicación son construidas con la misma tecnología, lo que hace amarremos la comunicación con tecnologías propietarias de la plataforma utilizada.

- Tolerancia a los fallos

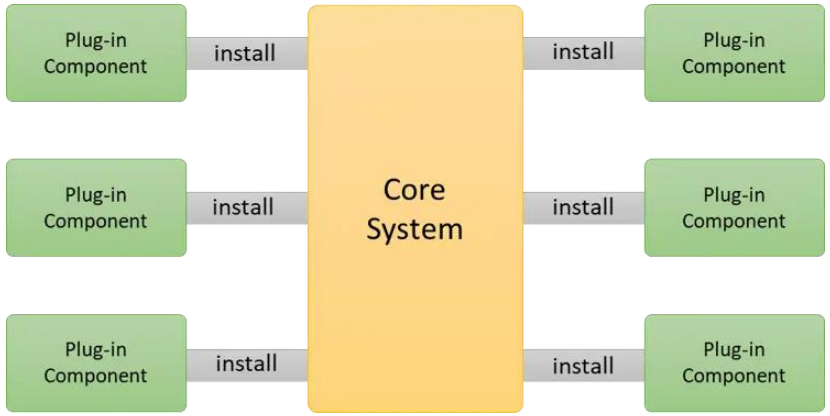
Si una capa falla, todas las capas superiores comienzan a fallar en cascada.



Microkernel

También conocido como arquitectura de Plug-in, permite crear aplicaciones extensibles, mediante la cual es posible agregar nueva funcionalidad mediante la adición de pequeños plugins que extienden la funcionalidad inicial del sistema.

- Usualmente se ocupa en la implementación de aplicaciones basadas en productos, es decir, aquellas que están empaquetadas y disponibles para su posterior descarga.
- Permite añadir características adicionales como plug-ins y es muy flexible y extensible.
- Se usa para aplicaciones que precisan datos de diferentes fuentes y para aplicaciones de flujo de trabajo.



Ventajas de la Arq. Microkernel

- Testabilidad

Debido a que los Plugins y el sistema Core son desarrollados de forma por separado, es posible probarlos de forma aislada.

- Rendimiento

En cierta forma, muchas de las aplicaciones basadas en Microkernel trabajan de forma Monolítica una vez que el Plug-in es instalado, lo que hace que todo el procesamiento se haga en una sola unidad de software.

- Despliegue

Debido a la naturaleza de Plugins es posible instalar fácilmente todas las características adicionales que sea necesarias, incluso, pueden ser agregar en tiempo de ejecución, lo que en muchos casos ni siquiera requiere de un reinicio del sistema.

Ventajas de la Arq. Microkernel

- Dinamismo

Las aplicaciones basadas en Plugins pueden habilitar o deshabilitar características basadas en perfiles, lo que ayuda que solo los plugins necesarios sean activados, incluso, pueden ser activados solo cuando son utilizados por primera vez (hot deploy.

- Construcción modular

El sistema de Plugins permite que diferentes equipos puedan trabajar en paralelo para desarrollar los diferentes Plugins.

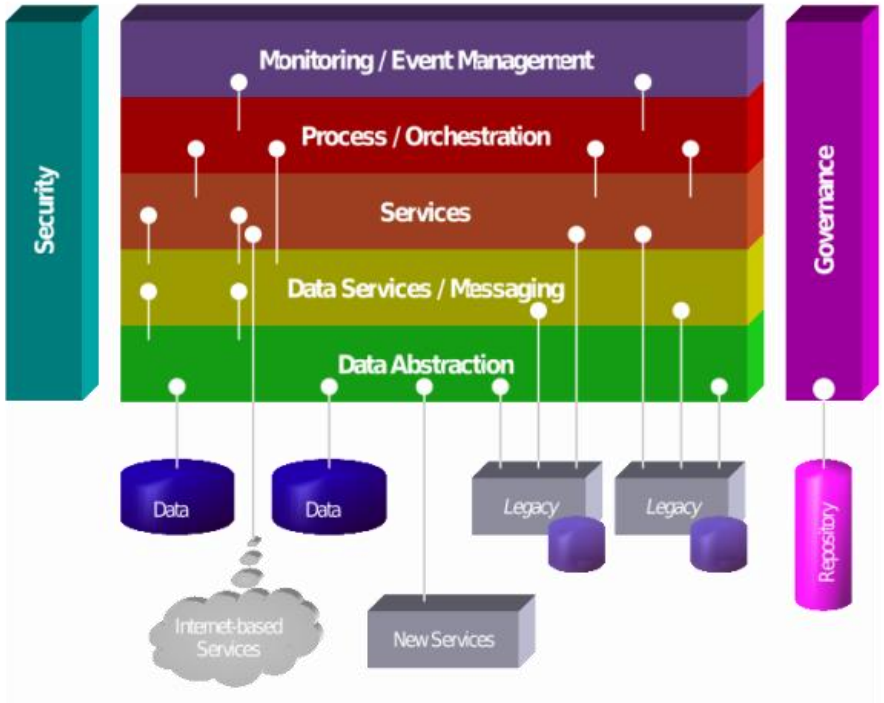
- Reutilización

Debido a que los Plugins puede ser instalados en cualquier instancia del sistema Core, es posible reutilizar los módulos en varias instancias, incluso, es posible comercializarlas de forma independiente.

Service-Oriented Architecture (SOA)

Es un método de desarrollo de software que utiliza componentes de software llamados servicios para crear aplicaciones (usualmente para empresas).

Cada uno de estos servicios brinda una capacidad empresarial y, además, se puede comunicar con el resto de los servicios mediante diferentes plataformas y lenguajes.



Ventajas de SOA

- Reduce el acoplamiento

La comunicación basada en contratos permite desacoplar las aplicaciones, ya que no existe referencias a la aplicación, si no a un servicio que puede estar en cualquier parte.

- Testabilidad

Las aplicaciones basadas en SOA son muy fáciles de probar, pues es posible probar de forma individual cada servicio.

- Reutilización

Los servicios SOA son desarrollados para hacer una sola tarea, por lo que facilita que otras aplicaciones reutilicen los servicios que existen.

Ventajas de SOA

- Agilidad

Permite crear rápidamente nuevos servicios a partir de otros existentes.

- Escalabilidad

Los servicios SOA son muy fáciles de escalar, sobre todo porque permite agregar varias instancias de un servicio y balancear la carga desde el ESB.

- Interoperable

La interoperabilidad es uno de los factores por lo que existe esta arquitectura y es que permite que aplicaciones heterogéneas se comuniquen de forma homogénea mediante el uso de estándares abiertos.

Desventajas de SOA

- Rendimiento

La arquitectura SOA no se caracteriza por el performance, pues la comunicación por la red y la distribución de los servicios agregar una latencia significativa en la comunicación.

- Volumen de peticiones

SOA no se lleva con grandes volúmenes por petición, por lo que para procesar mucha información es recomendable utilizar otras tecnologías.

- Gobierno

Para que SOA pueda ser implementado correctamente, es necesario crear un Gobierno que ponga orden sobre el uso y la creación de nuevos servicios, lo que puede representar una carga adicional a la empresa.

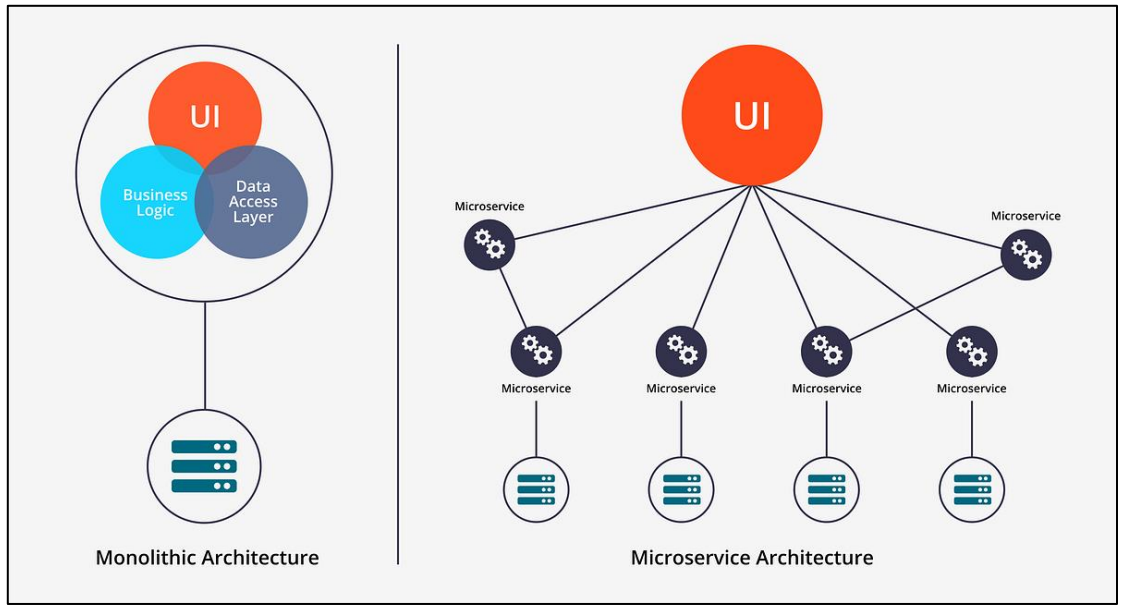
- Más puntos de falla

Debido a que SOA es una arquitectura distribuida, es necesario implementar estrategias de falla, pues se incrementa el número de puntos de falla.

Microservicios

En lo simple, lo que hace este patrón es escribir multitud de solicitudes que van a funcionar juntas.

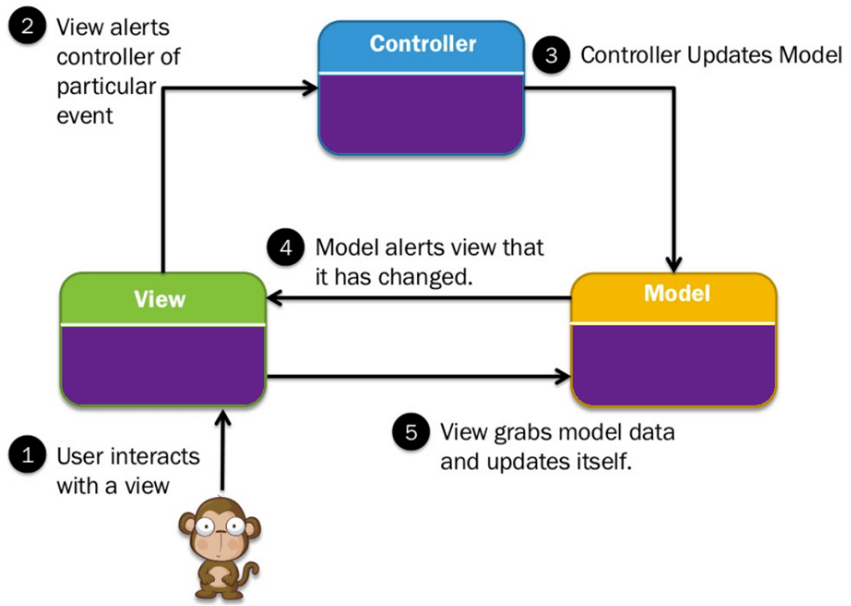
- Su principal ventaja es la posibilidad de escribir, mantener y desplegar cada microservicio de forma individual.
- Se utiliza, sobre todo, para webapps con pequeños componentes, centros de datos no muy grandes y el desarrollo rápido de aplicaciones web.



Modelo Vista Controlador - MVC

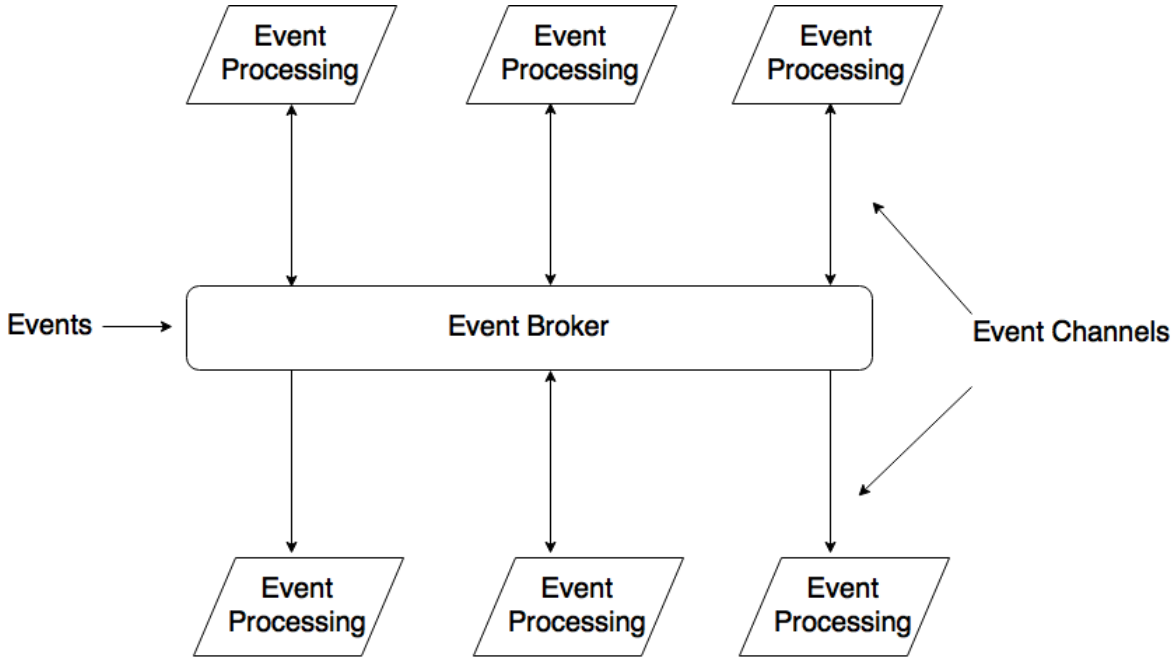
La arquitectura de software basada en el patrón Modelo-Vista-Controlador (MVC), divide el desarrollo del sistema en conceptos.

- Su objetivo es separar la lógica del negocio de la presentación.
- Pudiera parecer similar al Modelo de N-Capas, pero a diferencia de este la lógica puede estar presente en todos los componentes y cada componente se puede comunicar con los demás indistintamente, sin tener que pasar por alguno de ellos de manera obligatoria.



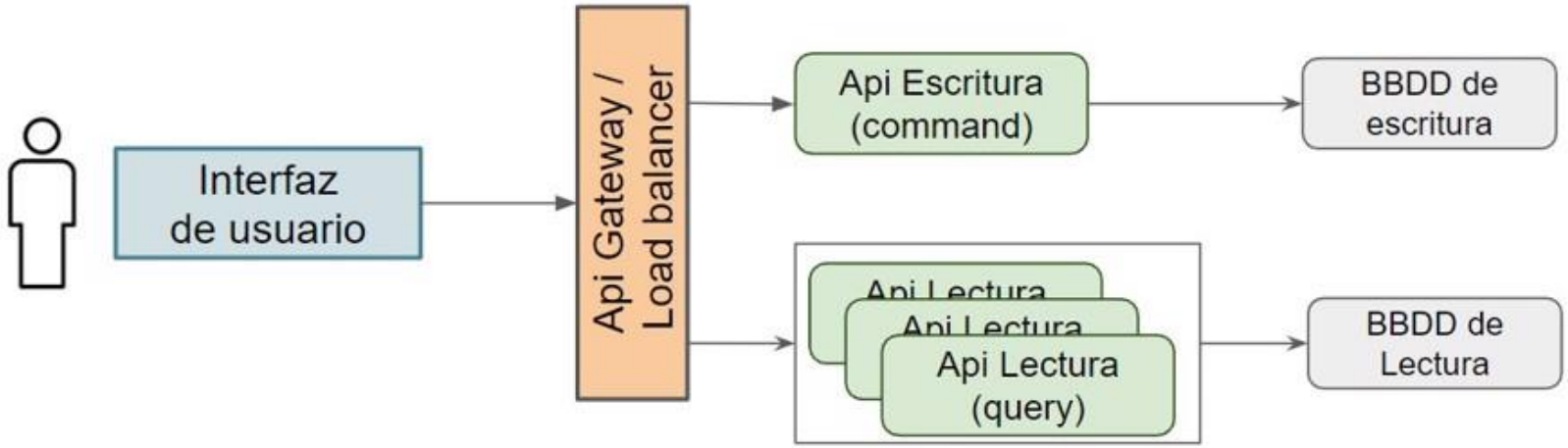
Event-based pattern

- Es una arquitectura asíncrona.
- Se utiliza para desarrollar sistemas altamente escalables (es su gran ventaja).
- Se basa en componentes de procesamiento de eventos de un solo propósito.
- Se utiliza, principalmente, para cuestiones como las interfaces de usuario.



Patrón CQRS

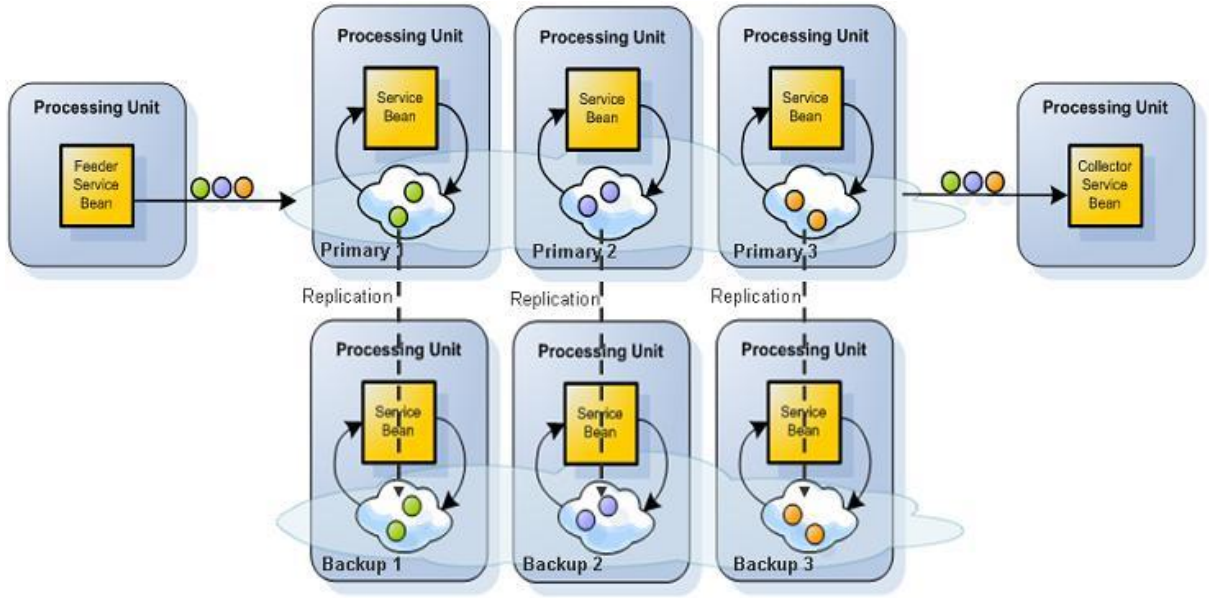
- CQRS: Segregación de Responsabilidades de Comandos y Consultas.
- Separa las operaciones de lectura y actualización de un almacén de datos.
- Se utiliza para maximizar el rendimiento, la escalabilidad y la seguridad de una aplicación.
- Permite que el sistema evolucione mejor con el tiempo y no se vea dañado por los comandos de actualización.



Arquitectura basada en espacio

Este patrón se usa para la resolución de problemas de escalabilidad y concurrencia.

- Para ganar escalabilidad se elimina la restricción de la base de datos central sustituyéndola por cuadrículas de datos replicados en memoria.
- Su principal ventaja es que consigue responder de forma rápida a los cambios y se usa para manejar datos de gran volumen (Por ej. Registros de usuarios).



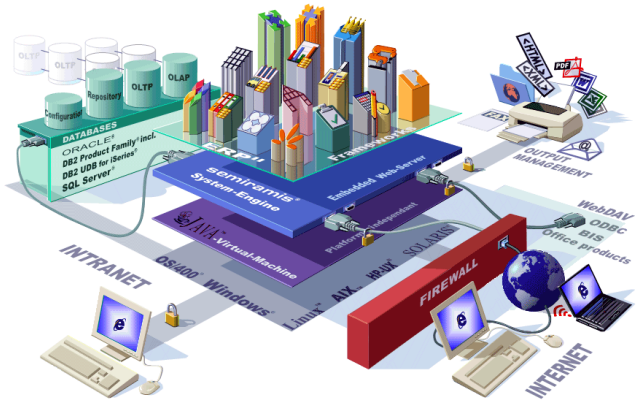
PATRONES DE DISEÑO DE SW

Patrones de Diseño de Software

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

- Cada patrón es como un plano que se puede personalizar para resolver un problema de diseño particular en el código.
- Evitan perder tiempo en soluciones a problemas ya resueltos o conocidos

Profundicemos un poco sobre los patrones de diseño de software



Tipos de patrones de diseño

Patrones creacionales

Su objetivo es resolver los problemas de creación de instancia. Estos ayudan a delegar responsabilidad de creación de objetos en situaciones necesarias.

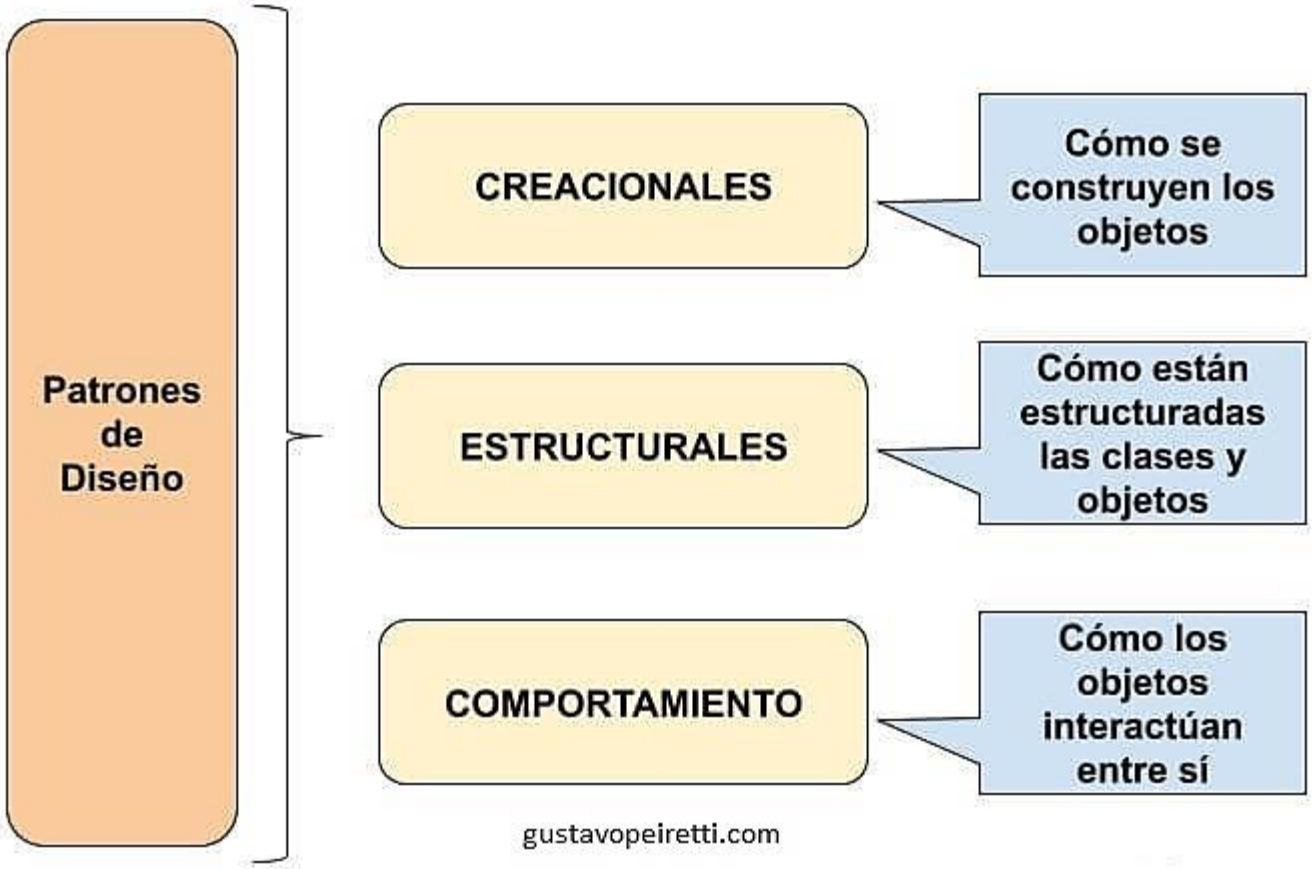
Patrones estructurales

Su nombre es muy descriptivo, se ocupa de resolver problemas sobre la estructura de las clases, es decir, se enfocan en cómo las clases y objetos se componen para formar estructuras mayores.

Patrones de comportamiento

Nos ayuda a resolver problemas relacionados con el comportamiento de la aplicación. Ofrece soluciones respecto a la interacción y responsabilidad entre objetos y clases.

Tipos de patrones de diseño



Patrones creacionales

Singleton

Garantiza la existencia de una única instancia para una clase.

Prototype (prototipo)

Clona las instancias ya existentes.

Abstract Factory

Permite proporcionar una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.

Patrones creacionales

Builder

Ayuda a crear objetos complejos de manera sencilla, legible y escalable. Se utiliza en situaciones en las que debe construirse un objeto repetidas veces.

Factory Method

Nos ayuda a tener instancias de un objeto dado el tipo.

NOTA AL MARGEN

Un muy buen ejemplo de Factory Method se puede encontrar en el siguiente enlace

https://mauriciogc.medium.com/javascript-patrones-de-dise%C3%B1o-en-js-creacional-parte-i-5bddfdb249be

Patrones Estructurales

Bridge (Puente)

Separa la abstracción de la implementación.

Decorator (Decorador)

Agrega funcionalidades a una clase de forma dinámica.

Facade (Fachada)

Nos provee una interfaz unificada y simple para acceder a un sistema más complejo.

Adapter

Cuando dos clases no se entiende, el adapter es mediador y adapta una clase para que la otra la entienda.

Patrones Estructurales

Composite

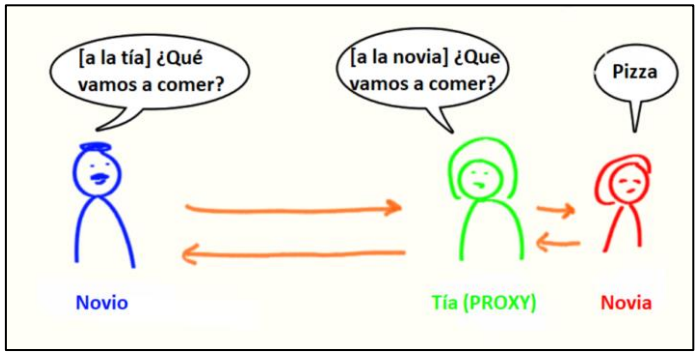
Ayuda a construir objetos complejos a partir de otros más simples.

Flyweight

Se refiere a los objetos que queremos reutilizar para crear objetos más ligeros.

Proxy

Es un elemento que se encarga de introducir un nivel de acceso a una clase. Ese nivel de acceso puede ser por seguridad o por complejidad.



Patrones de Comportamiento

Observer (Observador)

Un objeto le pasa el estado interno a muchos objetos que están interesados.

Chain of Responsibility

Simplifica las interconexiones de objetos.

Command

Separa acciones que pueden ser ejecutadas desde varios puntos diferentes de la aplicación a través de una interfaz sencilla.

Iterator

Este se utiliza en relación a objetos que almacenan colecciones de otros objetos.

Patrones de Comportamiento

Mediator

Define un objeto que media entre otros objetos.

Memento

Se utiliza para restaurar el estado de un objeto a un estado anterior.

State

Permite a un objeto alterar su comportamiento cuando su estado interno cambia.

Strategy

Permite que un objeto tenga parte o todo su comportamiento definido en términos de otro objeto que sigue una interfaz particular.

Patrones de Comportamiento

Template Method

Se centra en la reutilización del código para implementar pasos para resolver problemas.

Visitor

Se utiliza para separar la lógica y las operaciones realizadas sobre una estructura compleja.

NOTA AL MARGEN

Si desea profundizar en los patrones de diseño, vea el siguiente enlace
https://refactoring.guru/design-patterns/catalog

Cierre de la sesión