Centro de Enseñanza Técnica Industrial

Plantel Colomos

Ingeniería en Desarrollo de Software

Nombre Alumno: José Rafael Ruiz Gudiño

Registro: 20110374

Arquitectura de Software

Actividad 3 - Patrones Arquitectónicos

5°P

T/M

28/02/2022

	Propósito	Características	Ventajas	Desventajas	Usos
Microkernel	• 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. • La idea central es permitir la extensión de su funcionalidad o personalización, pero respetando el principio Open-Closed, es decir, está abierto para extender la funcionalidad, pero cerrado para modificar su funcionalidad principal.	• Se dividen en dos tipos de componentes, en sistema Core y los plugins. • El sistema Core contiene los elementos mínimos para hacer que la aplicación funcione y cumpla el propósito para el cual fue diseñada. • Los plugins on componentes periféricos que se añaden o instalan al componente Core para extender su funcionalidad. • Es un patrón natural para implementar aplicaciones basadas en productos.	• Gran flexibilidad y extensibilidad. • Algunas implementaciones web permiten añadir plugins mientras la aplicación se está ejecutando. • Buena portabilidad. • Facilidad de despliegue. • Respuesta rápida a un entorno en constante cambio que implica un entorno empresarial. • Alto rendimiento. • Testabilidad. • Performace. • Construcción Modular. • Reutilización	• Escalabilidad: Las aplicaciones basadas en Microkernel son generalmente desarrolladas para ser ejecutadas en modo Standalone. • Alta complejidad: Las aplicaciones basadas en Microkernel son difíciles de desarrollar.	• Se utiliza típicamente cuando los equipos de software crean sistemas con componentes intercambiables. • Se aplica a los sistemas de software que deben ser capaces de adaptarse a los requisitos cambiantes del sistema. • Aplicaciones que toman datos de diferentes fuentes, los transforman y los escriben a diferentes destinos • Aplicaciones de flujo de trabajo • Solicitudes de programación de tareas y trabajos
Microservicios	• Consiste en crear pequeños componentes de software que solo hacen una tarea, la hace	• Son altamente cohesivos, pues todas las funcionalidades que tiene dentro están relacionadas para resolver un único problema.	• Alta escalabilidad: permite montar numerosas instancias del mismo componente y balancear la carga entre todas las instancias.	• Performance: La naturaleza distribuida de los microservicios agrega una latencia significativa que puede ser un impedimento para aplicaciones donde	• Sitios web con pequeños componentes • Centros de datos corporativos con límites bien definidos

	bien y son totalmente autosuficientes. • Pequeño programa que se especializa en realizar una pequeña tarea y se enfoca únicamente en eso.	• Es común que los microservicios se comuniquen con otros microservicios para delegar ciertas • tareas, de esta forma, crean una red de comunicación entre ellos mismos. • Trabajan en una arquitectura distribuida, lo que significa todos los Microservicios son desplegados de forma independiente y están desacoplados entre sí.	• Agilidad: Permite que el componente tenga ciclo de desarrollo diferente del resto, que resulta en despliegues rápidos a producción sin afectar al resto de componentes. • Puedes escribir, mantener y desplegar cada microservicio por separado • Fácil de escalar, ya que sólo se pueden escalar los microservicios que necesitan ser escalados • Es más fácil reescribir las piezas de la aplicación porque son más pequeñas y menos acopladas a otras partes • Los nuevos miembros del equipo deben ser rápidamente productivos • La aplicación debe ser fácil de entender y modificar • Altamente mantenible y comprobable • Desplegable de forma independiente	el performance es lo más importante. • Múltiples puntos de falla: La arquitectura distribuida de los microservicios hace que los puntos de falla de una aplicación se multipliquen. • Trazabilidad: La naturaleza distribuida de los microservicios complica recuperar y realizar una traza completa de la ejecución de un proceso. • Madurez del equipo de desarrollo: Una arquitectura de microservicios debe ser implementada por un equipo maduro de desarrollo y con un tamaño adecuado.	• El rápido desarrollo de nuevos negocios y aplicaciones web • Los equipos de desarrollo que se extienden, a menudo por todo el mundo
Arquitectura en capas	Consta en dividir la aplicación en capas, con la intención de que cada capa tenga un rol muy definido, como podría ser, una capa de presentación (UI),	• La mayoría de las veces este estilo arquitectónico es implementado en 4 capas, presentación, negocio, persistencia y base de datos. • Todas las capas se colocan de forma horizontal, de tal forma que	• Alta comprobabilidad porque los componentes pertenecen a capas específicas de la arquitectura. • Alta facilidad de desarrollo porque este patrón es muy	• Complejidad de despliegue: En este tipo de arquitecturas es necesarios desplegar los componentes de abajo arriba, lo que crea una dependencia en el despliegue.	• Aplicaciones estándar de línea de negocios que hacen más que sólo operaciones CRUD. • Nuevas aplicaciones que necesitan ser

	una capa de reglas de negocio (servicios) y una capa de acceso a datos (DAO).	cada capa solo puede comunicarse con la capa que está inmediatamente por debajo. • Cada capa debe de ser un componente independiente, de tal forma que se puedan desplegar por separado. • Respetar el orden de las capas es muy importante, ya que brincarnos una capa para irnos sobre una más abajo suele ser un grave error.	conocido y no es excesivamente complejo de implementar. • Mantenible. • Fácil de asignar «roles» separados. • Fácil de actualizar y mejorar las capas por separado. • Separación de responsabilidades. • Fácil de desarrollar. • Fácil de probar. • Fácil de mantener. • Seguridad.	• Anclado a un Stack tecnológico. • Performance: La comunicación por la red o internet es una de las tareas más tardadas de un sistema. • Escalabilidad: Las aplicaciones que implementan este patrón por lo general tienda a ser monolíticas, lo que hace que sean difíciles de escalar. • Tolerancia a los fallos: Si una capa falla, todas las capas superiores comienzan a fallar en cascada.	construidas rápidamente. • Equipos de desarrolladores inexpertos que aún no entienden otras arquitecturas. • Aplicaciones que requieren normas estrictas de mantenimiento y comprobabilidad.
Arquitectura basada en eventos	• Es una arquitectura asíncrona y distribuida, pensada para crear aplicaciones altamente escalables. • Se espera que las aplicaciones lancen diversos “eventos” para que otros componentes puedan reaccionar a ellos, procesarlos y posiblemente generar nuevos eventos para que otros	• El Event Bus se encargará de recibir el evento y colocarlo en los Event Channels, las cuales son básicamente una serie de Colas (Queues) o Temas (Topics). • Los Event Processing son componentes de negocio que está a la escucha de nuevos eventos depositados en los Event Channels. Son los encargados de procesar los eventos, es decir, realizar acciones concretas sobre el sistema. • Los MOM son aplicaciones especializadas en el	• Escalabilidad: Permite que cada consumidor puede escalar de forma independiente y reduce al máximo el acoplamiento entre los componentes. • Despliegue: Debido al bajo acoplamiento entre los componentes, es posible el despliegue sin preocuparse por dependencias. • Performance. • Flexibilidad: EDA permite responder rápidamente a un entorno cambiante, debido a que cada componente procesador de eventos tiene una	• Testabilidad: Una arquitectura distribuida y asíncrona agrega cierta complejidad a las pruebas, pues no es posible generar un evento y esperar un resultado para validar el resultado. • Desarrollo: Codificar soluciones asíncronas es complicado, pero aún más, es la necesidad de crear manejadores de errores más avanzados que permitan recuperarse en una arquitectura asíncrona.	• Sistemas asíncronos con flujo de datos asíncronos. • Interfaces de usuario

	componentes continúen con el trabajo.	transporte confiable de mensajes.	sola responsabilidad y está completamente desacoplado de los demás.		
Arquitectura basada en el espacio	• Está diseñada para evitar el colapso funcional bajo una gran carga al dividir tanto el procesamiento como el almacenamiento entre múltiples un servidor y otro. • Está diseñado específicamente para abordar y resolver problemas de escalabilidad y concurrencia.	• Es un patrón de arquitectura útil para las aplicaciones que tienen volúmenes de usuarios concurrentes variables e impredecibles. • La alta escalabilidad se logra eliminando la restricción de la base de datos central y utilizando en su lugar cuadrículas de datos replicados en memoria		• Responde rápidamente a un entorno en constante cambio. • Son dinámicas, y las sofisticadas herramientas basadas en la nube permiten «empujar» fácilmente las aplicaciones a los servidores, simplificando su despliegue. • Se logra un alto rendimiento en el servidor gracias al acceso a los datos en memoria y a los mecanismos de almacenamiento en caché incorporados en esta pauta. • La elevada escalabilidad se debe a que se depende poco o nada de una base de datos centralizada.	• Datos de gran volumen como flujos de clicks y registros de usuarios • Datos de bajo valor que pueden perderse ocasionalmente sin grandes consecuencias • Redes sociales

Conclusiones

Los patrones de arquitectura son algo muy importante a tomar en cuenta al momento de comenzar a realizar un proyecto, esto debido a que va a ser el molde del diseño de nuestro proyecto. Al momento de elegir el patrón que se va a usar, hay que tener previamente muy definido y estructurado las metas y objetivos, para que, a partir de aquí, analicemos que patrón nos conviene o adapta mas al modelo de proyecto que se desea realizar. Como por ejemplo un IDE que tiene un patrón de tipo microkernel, que gracias a ello se le pueden añadir funciones añadidas o plugins a el programa central, o quizás si queremos hacer un sitio web podemos hacer uso de las bondades del patrón de microservicios para realizarlo.