Investigación arquitecturas de software

La arquitectura de software es un concepto que surge ya en los años 60 y se refiere a una planificación basada en modelos, patrones y abstracciones teóricas, a la hora de realizar una pieza de software de cierta complejidad y como paso previo a cualquier implementación 1. En la actualidad, existen varios tipos de arquitecturas de software que se utilizan en el desarrollo de aplicaciones. Algunos de los tipos más comunes son:

Descomposición Modular: donde el software se estructura en grupos funcionales muy acoplados.

Cliente-servidor: donde el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.

Arquitectura de tres niveles: donde el software se divide en tres capas: presentación, aplicación y datos.

Arquitectura orientada a eventos (EDA): donde el software se divide en pequeñas unidades de procesamiento que se comunican entre sí mediante eventos.

Arquitectura de Microservicios: donde el software se divide en pequeños servicios independientes que se comunican entre sí mediante una API 234.

Es importante tener en cuenta que la elección de una arquitectura de software depende del tipo de aplicación que se esté desarrollando y de las necesidades específicas del proyecto. Cada arquitectura tiene sus ventajas y desventajas, y es importante evaluar cuidadosamente cada opción antes de tomar una decisión 1.

Descomposición Modular

Descripción: La descomposición modular es un enfoque de diseño que implica dividir un sistema de software en módulos o componentes más pequeños y autónomos. Cada módulo es una entidad independiente que cumple una función o desempeña una responsabilidad específica dentro del sistema. Estos módulos son diseñados de manera que puedan operar de manera autónoma y, al mismo tiempo, interactuar de manera controlada con otros módulos a través de interfaces definidas.

Ventajas: Facilita el mantenimiento, la reutilización y la escalabilidad del software al permitir que los módulos se desarrollen, prueben y mantengan de manera independiente.

Ejemplo: En un sistema de gestión de inventario, puede haber módulos separados para la gestión de productos, pedidos y clientes.

Cliente-Servidor

Descripción: La arquitectura cliente-servidor es un modelo en el que los sistemas de software se dividen en dos partes principales: el cliente, que solicita servicios, y el servidor, que proporciona esos servicios. Los clientes envían solicitudes al servidor, que luego procesa y responde a esas solicitudes.

Cliente: El cliente es la parte de la aplicación que inicia la interacción. Puede ser un software o una interfaz de usuario que se ejecuta en un dispositivo, como un navegador web o una aplicación de teléfono móvil. El cliente envía solicitudes al servidor para solicitar información o servicios específicos.

Servidor: El servidor es la parte central que responde a las solicitudes del cliente. Puede ser un programa o un conjunto de programas que se ejecutan en un servidor dedicado o en la nube. El servidor procesa las solicitudes del cliente, realiza las operaciones necesarias y devuelve una respuesta adecuada.

Ventajas: Permite la distribución de la carga de trabajo y la centralización de la lógica de negocio en el servidor. Facilita la escalabilidad y la gestión de recursos.

Ejemplo: Aplicaciones web, donde el navegador actúa como cliente y un servidor web proporciona contenido y servicios.

Arquitectura de Tres Niveles

Descripción: Esta arquitectura divide una aplicación en tres capas distintas: la capa de presentación (front-end), la capa de lógica de negocio (back-end) y la capa de almacenamiento de datos. Cada capa tiene una responsabilidad específica en el procesamiento de la información.

Capa de Presentación (Front-End): La capa de presentación, también conocida como front-end, es la parte de la aplicación que interactúa directamente con los usuarios finales. Su responsabilidad principal es proporcionar una interfaz de usuario intuitiva y atractiva para que los usuarios interactúen con la aplicación.

Funcionalidad: Esta capa se encarga de la presentación de datos y la interacción con el usuario, incluyendo la presentación de páginas web, formularios, botones y cualquier otro elemento de la interfaz de usuario.

Tecnologías comunes: HTML, CSS, JavaScript y diversos frameworks de desarrollo front-end, como React, Angular o Vue.js.

Ventajas: La separación de la capa de presentación permite cambios en la interfaz de usuario sin afectar la lógica subyacente o la base de datos. Esto facilita la adaptación a diferentes dispositivos y la mejora de la experiencia del usuario.

Capa de Lógica de Negocio (Back-End): La capa de lógica de negocio, o back-end, es el núcleo funcional de la aplicación. Aquí se procesan las solicitudes del cliente, se realiza la lógica empresarial y se toman decisiones basadas en los datos recibidos de la capa de presentación y la capa de almacenamiento de datos.

Funcionalidad: La capa de lógica de negocio se encarga de procesar solicitudes, aplicar reglas empresariales, realizar cálculos, interactuar con bases de datos y coordinar el flujo general de la aplicación.

Tecnologías comunes: Lenguajes de programación como Python, Java, C#, Ruby, y frameworks de desarrollo back-end como Express.js (Node.js), Django (Python), Ruby on Rails, etc.

Ventajas: La separación de la lógica de negocio permite cambios en la presentación o la base de datos sin afectar las reglas de negocio. Esto facilita el mantenimiento, las actualizaciones y la escalabilidad.

Capa de Almacenamiento de Datos: La capa de almacenamiento de datos se encarga de gestionar la persistencia de los datos utilizados por la aplicación. Almacena y recupera información de una base de datos o de otros sistemas de almacenamiento.

Funcionalidad: Esta capa interactúa con sistemas de gestión de bases de datos (DBMS) para almacenar y recuperar datos. También se encarga de garantizar la integridad y la seguridad de los datos.

Tecnologías comunes: Bases de datos relacionales (por ejemplo, MySQL, PostgreSQL), bases de datos NoSQL (por ejemplo, MongoDB, Cassandra), sistemas de almacenamiento en la nube (por ejemplo, Amazon S3, Azure Cosmos DB).

Ventajas: La capa de almacenamiento de datos garantiza la persistencia de los datos a largo plazo y permite un acceso eficiente. Separar esta capa facilita la elección y el reemplazo de sistemas de almacenamiento según las necesidades cambiantes.

Ventajas: Facilita la separación de preocupaciones y el mantenimiento, permitiendo actualizaciones independientes en cada capa. Es común en aplicaciones empresariales.

Ejemplo: Una aplicación web en la que el navegador actúa como la capa de presentación, un servidor web como la capa de lógica de negocio y una base de datos para el almacenamiento de datos.

Arquitectura Orientada a Eventos (EDA)

Descripción: La Arquitectura Orientada a Eventos (EDA) es un paradigma de diseño de sistemas de software que se centra en la comunicación y la interacción de los componentes del sistema a través de eventos. En EDA, un evento representa un cambio o una acción significativa en el sistema y puede ser cualquier cosa, desde una solicitud de usuario hasta un cambio en el estado de un sensor o una actualización de datos en una base de datos. Cuando un evento ocurre, se dispara y se envía a los componentes interesados, que pueden reaccionar y tomar acciones en consecuencia.

Ventajas: Permite sistemas altamente escalables y flexibles, ya que los componentes pueden responder a eventos de manera independiente. Es adecuada para aplicaciones en tiempo real y sistemas distribuidos.

Ejemplo: Un sistema de monitoreo de sensores donde los eventos son datos enviados por los sensores, y los componentes reaccionan a cambios en esos datos.

Arquitectura de Microservicios

Descripción: La Arquitectura de Microservicios es un enfoque de diseño de software que se basa en la descomposición de una aplicación en múltiples servicios pequeños, independientes y autónomos. A diferencia de las arquitecturas monolíticas, donde toda la lógica de la aplicación se encuentra en una sola unidad, en la arquitectura de microservicios, cada función o capacidad se implementa como un servicio separado y específico. Estos servicios se comunican entre sí a través de interfaces bien definidas, generalmente a través de API (Interfaz de Programación de Aplicaciones).

Ventajas: Escalabilidad, mantenimiento y despliegue independientes de servicios. Facilita la adopción de tecnologías adecuadas para cada servicio.

Ejemplo: Un servicio de autenticación independiente, un servicio de gestión de productos, un servicio de carrito de compras, etc., que componen una aplicación de comercio electrónico.