Nombre: Janier Chará Cardona

Código: 1113306850

**Patrones de diseño de arquitectura de software**

Los patrones de arquitectura nos dan la estructura general del sistema y nos ayudan a definir donde van cada uno de los componentes y capas. Los patrones son soluciones reusables en problemas recurrentes, gracias a estos patrones podemos garantizar instancias únicas de un objeto en determinada clase o cambiar en tiempo de ejecución el algoritmo utilizado para algún fin en específico.

Algunos ejemplos de patrones de arquitectura son:

**Arquitecturas Orientadas A Eventos**

Este modelo de arquitectura nos permite interconectar componentes lanzando eventos a través de distintas partes del sistema, esta arquitectura está basada generalmente en 5 componentes.

**Generadores o productores**: Estos componentes son los encargados de generar los distintos eventos del sistema.

**Mensajes**: Contienen la información del evento (paquetes con información generada por los productores).

**Componente de mensajería**: Se encarga de recibir el mensaje, procesarlo si es necesario y luego informar a los distintos consumidores o partes interesadas acerca de que el evento ocurrió.

**Canales**: Los canales son el puente entre el componente de mensajería y el componente que está interesado acerca del evento, pueden ser implementados de dos formas normalmente. 1) utilizando colas de mensajes. 2) patrón de publicado suscriptor.

**Procesadores o consumidores**: Son los componentes que están interesados en los eventos. Se administran de tal forma que tengan una única tarea relacionada con el negocio y que esta tarea sea lo más específica posible.

Los procesadores o consumidores no deberán tener conocimiento de quienes generan los eventos, ni de otros procesadores del sistema, haciéndolos lo más independiente posibles del resto de la arquitectura.

**Arquitectura De Micro Servicios**

Este modelo de arquitectura permite desarrollar una aplicación como un conjunto de servicios pequeños donde cada servicio corre su propio proceso (procesos independientes), se comunican entre ellos utilizando mecanismos livianos (como APIs o aplicaciones) y se desarrollan alrededor de la necesidad del negocio dando la capacidad de una escalabilidad fácil y rápida empleando proceso de despliegue automático. Algunas topologías de esta arquitectura son:

**API REST**: En un conjunto de servicios pequeños, cuando se desea acceder a uno o varios de ellos, no son accedidos directamente por el usuario, sino que pasan por una capa intermedia (API) donde se encarga de traer el o los servicios necesarios en la petición para ser utilizados o consumidos. De esta forma si se necesita acceder a un servicio externo (desarrollado por terceros) el API funciona de mediador.

**Aplicación**: La aplicación que accede a los micro servicios es una aplicación propietaria, es decir del ecosistema de la arquitectura. Esta aplicación se conecta directamente a los micro servicios y los clientes o usuarios interactúan con esta, la cual encapsula los llamados a los micro servicios. En esta topología los micro servicios suelen ser más grandes o robustos que los utilizados en la topología API.

**Mensajería**: No se interactúa con los servicios a través de REST, sino que se interactúa a través de un broker de mensajería o cola. Este enfoque se utiliza generalmente en aplicaciones grandes donde se necesita utilizar mayor control en la capa de transporte y se quieren utilizar funcionalidades de cola, monitoreo y manejar de mejor forma el acceso asíncrono.

**Arquitectura De Microkernel**

Esta arquitectura también se conoce como arquitectura de plugins, donde existen dos componentes básicos, el sistema principal (flujo general de la aplicación o servicios internos) y los plugins (partes que se conectan al sistema principal para aumentar la funcionalidad o servicios externos).

**Sistema principal**: Contiene la lógica de dominio que es independiente (servicios internos) y el acceso a los recursos globales del sistema.

**Plugins**: Contiene funciones especiales que no pueden ir en el sistema principal, tales como

* Procesos demasiado específicos o de interés muy particular para algunos usuarios.
* Procesos complejos que ejecutan tareas muy pesadas como para incluirlas en el sistema principal.
* Incluir integraciones a servicios externos que no están soportadas en la plataforma principal.

El desarrollo e integración de estos plugins se conectan a los recursos globales del sistema principal y no implican ningún tipo modificación en la aplicación base.

El patrón de micro kernel tiene algunos aspectos de diseño fundamentales que se deben tener en cuenta:

**El registro**: Indica al sistema principal que plugins se deben cargar cuando se inicie el servicio.

**La Interacción**: Se decide cómo interactúan los plugins con el sistema principal, ya sea por eventos, invocando objetos o utilizando servicios a través de REST o SOAP, etc.