¿Cómo definir un estilo arquitectónico para una aplicación?

Linda María Fernández Olvera
UPIICSA-IPN
Ingeniería de Diseño
lfernandezo1500@alumno.ipn.mx

Abstract – Este ensayo explica los diferentes patrones arquitectónicos, identificando sus características y las situaciones en las que es más conveniente usarlos.

Índice de Términos – patrones arquitectónicos,

I. INTRODUCCIÓN

Los patrones arquitectónicos abordan un problema de aplicación específica dentro de un contexto dado y sujeto a limitaciones y restricciones. El patrón propone una solución arquitectónica que sirve como base para el diseño de la arquitectura.

Se distinguen de los patrones de diseño debido a que tienen un alcance global sobre los componentes, ya que afectan su funcionamiento, su integración o la forma en que se comunican con otros componentes.

II. DESARROLLO

Un patrón arquitectónico establece un marco de referencia a partir del cual es posible construir aplicaciones que comparten un conjunto de atributos y características mediante el cual es posible identificarlos clasificarlos.

Los patrones arquitectónicos descritos en este ensayo son:

- Data Transfer Object (DTO)
- Data Access Object (DAO)
- Polling
- Webhook
- Load Balance
- Service Registry
- API Gateway
- Access Token
- Single Sing On (Inicio de sesión único)
- Store and forward
- Circuit breaker
- Log Aggregation

A. Data Transfer Object (DTO)

Un Data Transfer Object (DTO) es un patrón de diseño arquitectónico utilizado para transferir datos entre capas de una aplicación. Es un objeto simple que contiene datos que deben transferirse de una capa de la aplicación a otra.

Tiene como finalidad la creación de objetos planos con una serie de atributos que pueden ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarlas en una única clase simple, como se muestra en la Figura 1, DTO propone usar clases especiales para la transmisión de datos.

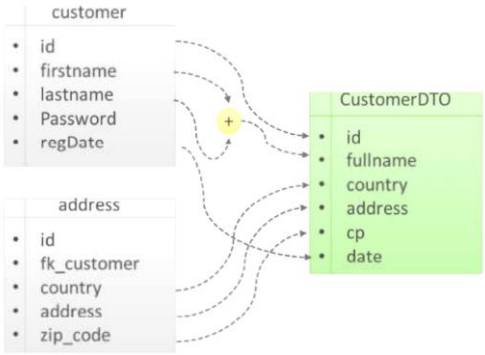

Figura 1. Implementado un DTO

El patrón Converter nos ayuda evitando la repetición de código, pues encapsula la lógica de conversión de dos tipos de datos, complementando al patrón DTO.

B. Data Access Object (DTO)

Es un patrón de diseño arquitectónico que se utiliza para abstraer la capa de acceso a datos de una aplicación.

La DAO proporciona una interfaz estándar para acceder a datos de una base de datos u otra fuente de datos. Esto permite que la lógica empresarial de una aplicación sea independiente de la tecnología de almacenamiento de datos subyacente otorgando una amplia reutilización en la aplicación.

En ocasiones donde requerimos obtener datos de más de una fuente podemos emplear el patrón Abstract Factory ya que

permite conectarnos a diferentes fuentes de datos y determinar en tiempo de ejecución que instancia concreta del DAO debemos instanciar.

En teoría deberíamos tener al menos un DAO por cada Entidad de nuestro proyecto, con la finalidad de controlar el acceso a los datos.

C. Polling

Es un patrón utilizado para comprobar el estado de un dispositivo o proceso a intervalos regulares. Debemos utilizarlo cuando no tenemos una mejor forma de consultar las actualizaciones de un sistema externo, ya que su uso puede afectar el rendimiento.

Implica enviar solicitudes a un servidor o dispositivo para verificar nuevos datos o eventos, y esperar una respuesta antes de enviar la siguiente solicitud. El sondeo se usa a menudo en aplicaciones que requieren datos o notificaciones en tiempo real, como aplicaciones de chat o juegos en línea.

Es excelente cuando debemos monitorear los cambios en sistemas de terceros, donde no tenemos control sobre ellos o no fueron diseñados para notificar sus cambios.

D. Webhook

Un webhook es un patrón para recibir notificaciones de un servidor cuando ocurren ciertos eventos.

Es una alternativa al sondeo, que implica la comprobación periódica de nuevos datos o eventos. Con un webhook, el servidor envía una notificación a una URL específica cuando ocurre un evento, lo que permite que la aplicación cliente reaccione al evento en tiempo real.

Puede ser implementado por enviar en el payload la información del evento o enviar la información mínima para que el remitente sepa lo que cambio y luego consumir el servicio.

El uso de Webhook evita el uso de Polling, evitando afectar el performance por el número de peticiones.

E. Load Balance

Es el proceso de distribución de cargas de trabajo entre múltiples recursos informáticos para optimizar el rendimiento, la confiabilidad y la escalabilidad.

En un sistema distribuido, los balanceadores de carga se utilizan para distribuir las solicitudes entrantes entre varios servidores o servicios, lo que garantiza que ningún servidor único se vea abrumado y que el sistema en general pueda manejar la carga de trabajo mejorando el rendimiento.

Permite que las aplicaciones escalen de forma fácil y casi ilimitada, con solo agregar nuevos servidores al cluster y repartir la carga entre ellos.

Hoy en día es indispensable ya que permite aprovisionar rápidamente nuevos servidores que se agregan dinámicamente a la red.

F. Service Registry

Un service registry es un patrón que propone crear un servidor centralizado donde todos los servicios se registren al momento de encender el sistema.

Proporciona una forma para que los servicios se registren y para que los clientes descubran y accedan a los servicios que necesitan.

Los service registry se usan comúnmente en arquitecturas de microservicios para permitir que los servicios se comuniquen y colaboren entre sí.

G. API Getaway

Un API Gateway es un tipo de servicio que proporciona un único punto de entrada para que los clientes accedan a múltiples servicios en una arquitectura de microservicios.

El API Gateway actúa como un proxy inverso, enrutando las solicitudes de los clientes al servicio adecuado y devolviendo la respuesta al cliente. También proporciona funciones adicionales, como autenticación, limitación de velocidad y almacenamiento en caché.

Es la cara que damos a los clientes que se comunicarán con nosotros, ofrece servicios simples y de alto nivel, como se muestra en la Figura 2, su implementación.

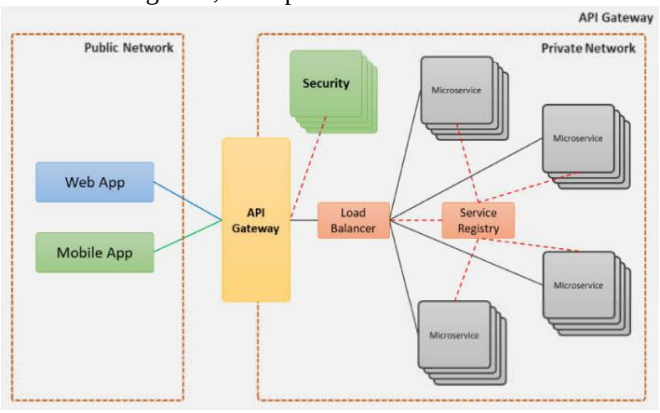

Figura 2. Implementación de API Gateway

Al ocultar la complejidad de la API a los clientes reducimos la complejidad de usarla, tenemos una mejor experiencia de usuario y controlamos la forma y los servicios que se exponen al público.

H. Access Token

Un Access token es una cadena de caracteres que representa la autorización de un cliente para acceder a un recurso protegido.

Los tokens de acceso se usan comúnmente en OAuth, que es un estándar para otorgar acceso a recursos protegidos sin compartir las credenciales del usuario.

Los tokens de acceso suelen ser de corta duración y solo son válidos por un tiempo limitado.

El uso de tokens ha favorecido mucho a las API tipo REST, pues ya no es necesario mantener una sesión del lado del servidor y permite ahorrar recursos.

I. Single Sign On (SSO)

El inicio de sesión único es un patrón que permite a los usuarios autenticarse con un conjunto único de credenciales para acceder a múltiples aplicaciones o servicios.

Con SSO, los usuarios solo necesitan iniciar sesión una vez para obtener acceso a todas las aplicaciones y servicios que están autorizados a usar.
Esto simplifica la experiencia del usuario y reduce la necesidad de que los usuarios recuerden varios conjuntos de credenciales de inicio de sesión.

J. Store and Forward

Almacenar y reenviar es un patrón utilizado en redes para garantizar la entrega confiable de datos.

En un sistema de almacenamiento y envío, los datos se reciben en un nodo y se almacenan temporalmente antes de enviarse al siguiente nodo de la red.

Esto permite que el sistema se recupere de fallas temporales o retrasos en la red, asegurando que los datos se entreguen incluso si uno o más nodos no están disponibles.

En la práctica este patrón es altamente utilizado porque garantiza la entrega de los mensajes sino que permite que el consumidor de los mensajes pueda ir extrayendo y procesando los mensajes a su propio ritmo.

K. Circuit breaker

Es un patrón de diseño utilizado para proteger un sistema contra sobrecargas o fallas.

Su objetivo es evitar que un sistema realice demasiadas solicitudes a otro sistema o recurso, lo que puede causar problemas de rendimiento o interrupciones.

El circuit breaker supervisa la cantidad de solicitudes que se realizan y, si alcanza un cierto umbral, se "disparará" y evitará que se realicen más solicitudes hasta que haya transcurrido un período de tiempo específico. Esto permite que el sistema se recupere y evite que ocurran más problemas.

Impide que inundemos nuestra aplicación con gran cantidad de solicitudes que van a fallar, es muy utilizado en procesos críticos, donde no podemos cancelar la operación y regresar un error al cliente, sino tratamos de proporcionar el servicio de una forma diferente.

L. Log Aggregation

Este patrón realiza un proceso de recopilar mensajes de registro de varias fuentes y almacenarlos en una ubicación central, generalmente una herramienta de administración de registros o una plataforma de análisis de registros.

Esto le permite buscar, analizar y monitorear fácilmente los registros, lo que puede ser útil para solucionar problemas, identificar tendencias y detectar posibles amenazas de seguridad.

La agregación de registros se usa a menudo en sistemas de software grandes y complejos donde múltiples aplicaciones, servicios y servidores generan registros. Al centralizar los registros, es más fácil obtener información sobre el estado general y el comportamiento del sistema.

III. CONCLUSIÓN

Hoy en día es necesaria la implementación de patrones que nos permitan manejar adecuadamente los datos de los clientes, aprovechar los recursos de los servidores y mejorar el performance del sistema, además de evitarnos repetir las soluciones y facilitar la instanciación de los servicios.

También es importante tomar las medidas necesarias para proteger los datos que manejamos ya sea con la API Gateway o con el uso de tokens, ya que es información que podría ser delicada.

El manejo de actualizaciones y cambios del sistema también es importante, ya que buscamos dar una experiencia de usuario buena, a través de iteraciones podemos comprobar el estado del sistema y actualizar lo que sea necesario, además de enviar la información correctamente.

REFERENCES

[1] Oscar, Blancarte, "*Introducción a la Arquitectura de Software, un enfoque práctico*" 1a ed. Oscar Blancarte Blog. 2020