TEMA VI

ARQUITECTURA DE SOFTWARE

Introducción

La arquitectura de software es una de las disciplinas más cruciales en el mundo del desarrollo web. Supone la estructura que se utiliza para encajar las piezas con las que un desarrollador va a jugar a la hora de construir un sistema determinado.

¿Qué es la 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 la estructura que ha de tener un software, las partes que vamos a construir y la forma en la que las vamos a combinar y juntar para poder trabajar con ellas.

1. Tipos de arquitetura de software

1.1. Arquitetura Spaghetti

Este tipo de arquitectura tiene la particularidad de que no es arquitectura. Pertenece a los inicios de las aplicaciones web, cuando se programaban en el lado del servidor y no en el Front end.

Se le llama spaghetti porque mezcla todo tipo de funcionalidades que suponen dificultades en la programación, en la legibilidad y el mantenimiento cuando las aplicaciones empezaban a crecer.

1.2. Arquitectura en capas

La arquitectura por capas nace para suplir los problemas derivados de la arquitectura spaghetti. **Su principal objetivo es contar con capas que tienen diferentes usos**.

Por ejemplo, mientras que una de ellas se encarga de la visualización de datos, otra tiene que servir para acceder a la base de datos. De esta manera, cada tarea se reparte en una capa diferente, de modo que se facilita el trabajo en ellas.

1.3. Arquitectura hexagonal

La arquitectura hexagonal **aísla las entradas y salidas de aplicación de la lógica interna de la aplicación**. Gracias a este aislamiento, se generan partes independientes que no dependen de los cambios externos, de esta forma, estos pueden cambiarse de forma individual sin afectar al resto.

¿Qué son los patrones de arquitectura de software?

Los patrones de arquitectura son **maneras de copiar estructuras de diseño que ya funcionaban y aprovecharlas en otras creaciones**. Si un patrón ya ha funcionado, lo mejor que se puede hacer es recuperarlo y utilizarlo para ganar en eficiencia.

Hoy en día, un arquitecto de software se encarga de seleccionar, adaptar y combinar diferentes patrones para encajarlos en un contexto específico.

1. Sistemas de Software micro kernel

El patrón arquitectónico Microkernel se usa cuando se realizan **sistemas con componentes intercambiables**. Se considera un patrón natural para la implementación de aplicaciones basadas en productos, es decir, aquellas que están empaquetadas y disponibles para su posterior descarga. Este patrón 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.

2. Patrón de arquitectura microservicios

Los patrones de arquitectura de microservicios lo que hacen, en realidad, 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 webs con pequeños componentes, centros de datos no muy grandes y el desarrollo rápido de aplicaciones web.

3. Patrón de arquitectura Event-based pattern

Es una arquitectura asíncrona y se emplea para **desarrollar sistemas altamente escalables**, lo que supone su gran ventaja. Este sistema se basa en componentes de procesamiento de eventos de un solo propósito.

Se utiliza para cuestiones como las interfaces de usuario.

4. Patrón CQRS

CQRS son las siglas de Segregación de Responsabilidades de Comandos y Consultas. Este patrón **se utiliza para maximizar el rendimiento, la escalabilidad y la seguridad de una aplicación**. De esta forma, permite que el sistema evolucione mejor con el tiempo y no se vea dañando por los comandos de actualización.

5. Patrón de arquitectura basado en el 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 utiliza para manejar datos de gran volumen como, por ejemplo, los registros de usuarios.

MVC

MVC se usa inicialmente en sistemas donde se requiere el uso de interfaces de usuario, aunque en la práctica el mismo patrón de arquitectura se puede utilizar para distintos tipos de aplicaciones. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, *Model, Views & Controllers*, si lo prefieres en inglés. Estudiaremos con detalle estos conceptos, así como las ventajas de ponerlos en marcha cuando desarrollamos.

MVC es un "invento" que ya tiene varias décadas y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web.

Por qué MVC

La rama de la ingeniería del software se preocupa por crear procesos que aseguren calidad en los programas que se realizan y esa calidad atiende a diversos parámetros que son deseables para todo desarrollo, como la estructuración de los programas o reutilización del código, lo que debe influir positivamente en la facilidad de desarrollo y el mantenimiento.Los ingenieros del software se dedican a estudiar de qué manera se pueden mejorar los procesos de creación de software y una de las soluciones a las que han llegado es la arquitectura basada en capas que separan el código en función de sus responsabilidades o conceptos. Por tanto, cuando estudiamos MVC lo primero que tenemos que saber es que está ahí para **ayudarnos a crear aplicaciones con mayor calidad.**

Quizás, para que a todos nos queden claras las ventajas del MVC podamos echar mano de unos cuantos ejemplos:

Por qué MVC

1. Aunque no tenga nada que ver, comencemos con algo tan sencillo como son el HTML y las CSS. Al principio, en el HTML se mezclaba tanto el contenido como la presentación. Es decir, en el propio HTML tenemos etiquetas como "font" que sirven para definir las características de una fuente, o atributos como "bgcolor" que definen el color de un fondo. El resultado es que tanto el contenido como la presentación estaban juntos y si algún día pretendíamos cambiar la forma con la que se mostraba una página, estábamos obligados a cambiar cada uno de los archivos HTML que componen una web, tocando todas y cada una de las etiquetas que hay en el documento. Con el tiempo se observó que eso no era práctico y se creó el lenguaje CSS, en el que se **separó la responsabilidad de aplicar el formato** de una web.

Por qué MVC

2. Al escribir programas en lenguajes como PHP, cualquiera de nosotros comienza mezclando tanto el código PHP como el código HTML (e incluso el Javascript) en el mismo archivo. Esto produce lo que se denomina el "Código Espagueti". Si algún día pretendemos cambiar el modo en cómo queremos que se muestre el contenido, estamos obligados a repasar todas y cada una de las páginas que tiene nuestro proyecto. Sería mucho más útil que el **HTML estuviera separado del PHP.**
3. Si queremos que en un equipo intervengan perfiles distintos de profesionales y trabajen de manera autónoma, como diseñadores o programadores, ambos tienen que tocar los mismos archivos y el diseñador se tiene necesariamente que relacionar con mucho código en un lenguaje de programación que puede no serle familiar, siendo que a éste quizás solo le interesan los bloques donde hay HTML. De nuevo, sería mucho más fácil la **separación** del código.

Por qué MVC

4. Durante la manipulación de datos en una aplicación es posible que estemos accediendo a los mismos datos en lugares distintos. Por ejemplo, podemos acceder a los datos de un artículo desde la página donde se muestra éste, la página donde se listan los artículos de un manual o la página de *backend* donde se administran los artículos de un sitio web. Si un día cambiamos los datos de los artículos (alteramos la tabla para añadir nuevos campos o cambiar los existentes porque las necesidades de nuestros artículos varían), estamos obligados a cambiar, página a página, todos los lugares donde se consumían datos de los artículos. Además, si tenemos el código de acceso a datos disperso por decenas de lugares, es posible que estemos repitiendo las mismas sentencias de acceso a esos datos y por tanto no estamos **reutilizando código.**

Quizás te hayas visto en alguna de esas situaciones en el pasado. Son solo son simples ejemplos, habiendo decenas de casos similares en los que resultaría útil aplicar una arquitectura como el MVC, con la que nos obliguemos a separar nuestro código atendiendo a sus responsabilidades.

Ahora que ya podemos tener una idea de las ventajas que nos puede aportar el MVC, analicemos las diversas partes o conceptos en los que debemos separar el código de nuestras aplicaciones.

Modelos

Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes *selects*, *updates*, *inserts*, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

Vistas

Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite **mostrar la salida.**

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera.

Controladores

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad es una capa que sirve de **enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación.** Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

Arquitectura de aplicaciones MVC

A continuación encontrarás un diagrama que te servirá para entender un poco mejor cómo colaboran las distintas capas que componen la arquitectura de desarrollo de software en el patrón MVC.