Implementación de patrones de diseño en arquitectura Api Rest

Rodrigo Sánchez Peregrino, Miguel Pérez Vasconcelos, Rosa Gómez Domínguez, Eutimio Sosa Silva, Fidelio Castillo Romero

Tecnológico Nacional de México- Instituto Tecnológico de Villahermosa.

Resumen

En la actualidad arquitectura api rest es una de las más utilizadas en el mundo del desarrollo de software es por eso por lo que se debe implementar un patrón de diseño para poder manipular de manera óptima cada uno de los componentes que este contiene. Este documento presenta un ejemplo de implementación de un patrón de diseño personalizado en arquitectura api rest.

Abstract

Today the api rest architecture is one of the most used in the world of software development, that is why a design pattern must be implemented to be able to optimally manipulate each of the components it has. This document presents an example of implementing a custom design pattern in api rest architecture.

Palabras claves: Api, Rest, Arquitectura, Patrón, Diseño, Esquema, Angular
Keywords: Api, Rest, Architecture, Pattern, Design, Schema, Angular

Hoy en día crear una arquitectura de software e implementar un patrón de diseño se ha convertido en la clave para que en el proceso del desarrollo se tengan los menos errores posibles y que la aplicación sea creada de manera óptima y funcione correctamente.

Para la autora Ester Ribas (2018) un api rest es un conjunto de reglas y especificaciones que las aplicaciones pueden seguir para que se puedan comunicar entre sí, cada una de ellas está diseñada en un lenguaje de programación determinado, y con unas características específicas. Por la facilidad de comunicación entre otras aplicaciones, esta se ha convertido en una de las más utilizadas en el mundo del desarrollo.

Figura 1. Arquitectura Api Rest

Se considera que al diseñar una arquitectura api rest debemos de pensar en los patrones de diseño como una estrategia para que el proyecto pueda ser manipulado por cada una de las áreas de un equipo de programación, desde la base de datos hasta el consumo de las api's en alguna aplicación el implementarlo de manera correcta ayudará a los programadores a tener mejores prácticas y tener un mejor orden de los componentes para poder dar un soporte más preciso en caso de fallas. En caso contrario, el no implementar un patrón de diseño de manera correcta implica:

1. no tener un orden en el código
2. reutilización de clases y componentes de manera innecesaria.

3. Código ilegible.
4. Dificultad para el mantenimiento.
5. Entre otros.

A la fecha ha crecido el interés por implementar patrones de diseño en los proyectos, especialmente en los desarrollos de aplicaciones web ya que estos contienen una arquitectura más amplia a diferencia de otros, por lo que muchos programadores optan por implementar un patrón de diseño de los ya existentes o personalizar uno de acuerdo con sus necesidades.

Según Christopher Alexander (pattern language 1977), "cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces".

Aunque Alexander se refería a patrones en ciudades y edificios, lo que dice también es válido para patrones de diseño orientados a objetos. Nuestras soluciones se expresan en términos de objetos e interfaces, en vez de paredes y puertas, pero en la esencia de ambos tipos de patrones se encuentra una solución a un problema dentro de un contexto.

En resumen, un patrón de diseño en software es una estructura que se crea para poder solucionar uno o varios problemas al momento de realizar un proyecto de programación, son estándares que se establecen para tener una mejor organización dentro de nuestra arquitectura.

Al trabajar con un equipo de desarrollo debe estar definido el patrón a utilizar para poder trabajar de manera sincronizada entre sus miembros y así poder comprender en qué contexto está trabajando cada uno.

Al implementar un patrón de diseño a nuestra arquitectura Api Rest, esta nos quedaría de la siguiente manera:

Esquema en Base de Datos

La primera capa de la arquitectura Api Rest es la base de datos, es ahí donde nosotros guardamos nuestros datos de una aplicación, podremos consultar, actualizar o eliminar datos. Para hacer más eficiente esta capa podemos implementar un esquema.

Un esquema en una base de datos es un conjunto de reglas que debemos seguir para poder asignar tablas con una estructura más fácil de comprender para el desarrollador de la base de datos (DBA). Así, el DBA puede asignar una tabla directamente a un esquema y saber que función va a tener esta misma.

Así mismo se pueden aplicar reglas de implementación o normalización de los campos donde se establece el nombre que pueden llevar, si la primera letra es mayúscula, minúscula o si se tiene que escribir alguna palabra clave para su identificación.

Al establecer esta estructura sabremos identificar directamente la información de los campos, así como poder cambiar datos de una manera eficiente para que la base de datos no se vea afectada al momento de una modificación.

Al visualizar la Figura II podemos apreciar un diagrama de base de datos que se encuentra bajo un esquema personalizado, donde encontramos tres tipos objetos: catalogo, reporte y seguridad, cada uno contiene sus tablas correspondientes, también podemos apreciar que el esquema sigue una regla de campos donde el primer campo de cada tabla debe ser identificado con un "id" para hacer referencia al campo llave y por consiguiente el nombre del campo.

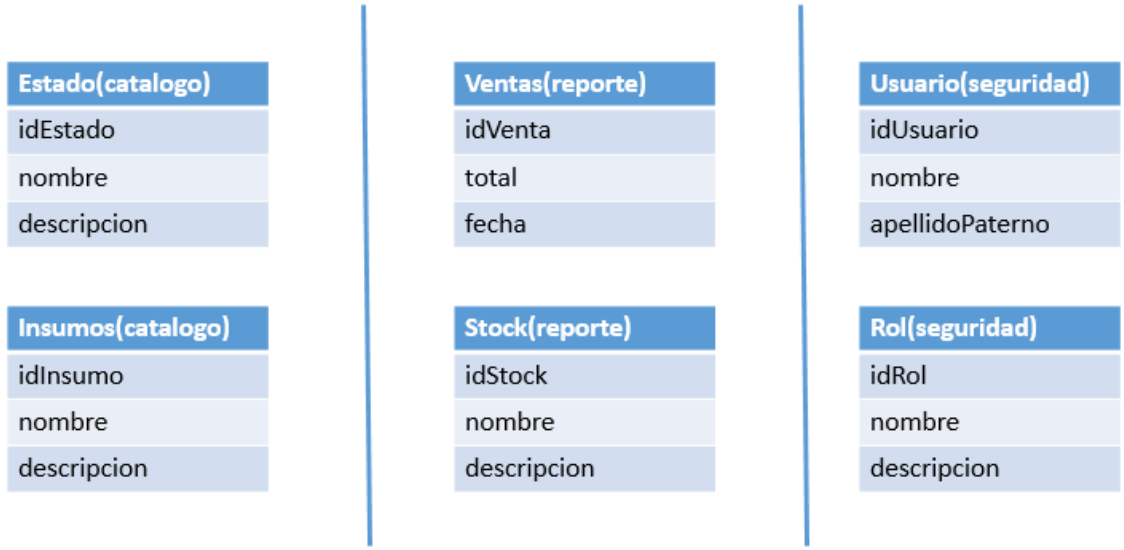

Figura 2. Ejemplo de esquema en base de datos

Al utilizar este tipo de esquema estamos hablando de un patrón que debemos de seguir para continuar trabajando en la base de datos, mantener su performance e impactar favorablemente en su mantenimiento.

Patrón DAL, MODEL, CONTROLLER

La segunda capa dentro de la arquitectura es donde construiremos toda la lógica de nuestras Api's, para ello debemos implementar un patrón de diseño personalizado que se ajuste a las necesidades del proyecto.

Nuestro patrón personalizado llevara una capa de DAL (Data Access Layer) que será nuestro acceso a la conexiones con la base de datos; una capa Model (Modelado de datos) que incluirá nuestros modelos de las entidades a utilizar y Controller (Controlador) que llevará la lógica para la implementación de la Api, dicho patrón es muy parecido al Modelo Vista Controlador (MVC) que es muy utilizado para las aplicaciones web pero para este proyecto quitamos la vista ya que esta será definida por el programador Front-End que se encargara de la manipulación de los datos.

La figura III nos muestra un ejemplo del patrón a utilizar, como vemos en la capa DAL tenemos tres clases donde cada una inicia en minúscula con la palabra que las identifica seguido de la palabra "Repository" que nos sirve para identificar que dentro de cada una de esas clases tendremos el repositorio con la información con la que se identifican.

Seguido tenemos la capa Model donde en sus clases utiliza un patrón de nombre que inicia con la letra "m" en minúscula para identificar que esa clase es un modelo y seguido se encuentra el nombre que las identifica con su inicial en mayúscula.

ISSN: 2007-4786

Por último, tenemos la capa Controller donde sus clases inician en minúscula con el nombre que las identifica seguido de la palabra "Controller" que identifica que esa clase es un controlador para poder manipular los recursos que esta contiene.

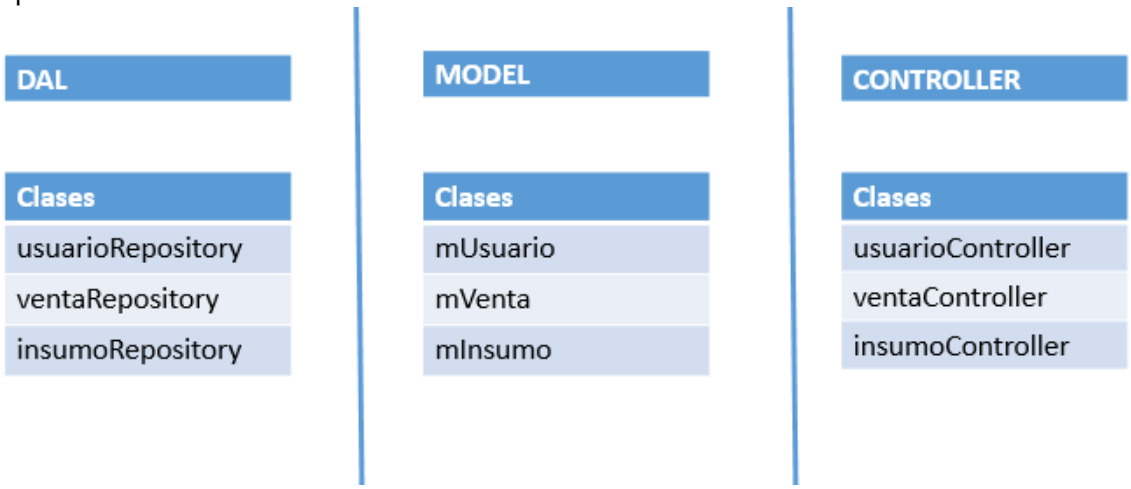

Figura 3. ejemplo de patrón de diseño personalizado

Al seguir este patrón de diseño podemos identificar de manera más rápida que clase pertenece a cada capa y de acuerdo con su nombre que función tiene esta dentro del proyecto, así al crear más recursos sabremos a que capa pertenece y podremos visualizar de manera más puntual los errores que pudieran aparecer durante el proceso del desarrollo.

Aplicación Web

Como capa final tendremos el desarrollo de la aplicación web que es donde se consumirán las Api's desarrolladas y se podrá manipular cada uno de los recursos mediante Endpoint's definidos por el desarrollador Back-End.

Existe una variedad de lenguajes de programación (por ejemplo, javascript, php, etc.) para aplicaciones web, y para ellos también existen diferentes frameworks (por ejemplo, angular, vuejs, laravel, entre otros) y librerías que facilitan el desarrollado.

De acuerdo con el artículo de MDN (2020) "JavaScript (JS) es un lenguaje ligero e interpretado, orientado a objetos con funciones de primera clase, más conocido como el lenguaje de script para páginas web, pero también usado en muchos entornos sin navegador, tales como node.js, Apache CouchDB y Adobe Acrobat. Es un lenguaje script multi-paradigma, basado en prototipos, dinámico, soporta estilos de programación funcional, orientada a objetos e imperativa".

Javascript puede ser utilizado por diferentes frameworks en los que se encuentra Angular que de acuerdo con su creador Google, "Angular (comúnmente llamado Angular 2+ o Angular 2) es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles".

El patrón de diseño que comúnmente se utiliza en angular es MVC. La autora Yenisleidy Fernández Romero (2012) define que el patrón MVC es un paradigma que divide las partes que conforman una aplicación en el Modelo, las Vistas y los Controladores, permitiendo la implementación por separado de cada elemento, garantizando así la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo.

Pero de igual manera podemos personalizar este patrón agregándole la capa "Services" que es donde se refleja el consumo de la Api's.

En la figura IV podemos identificar que al patrón MVC agregamos la capa Services cuyas clases inician con la palabra que las identifica en minúscula, seguido de la palabra "Service" para hacer referencia que se trata de un servicio donde se gestionan los datos.

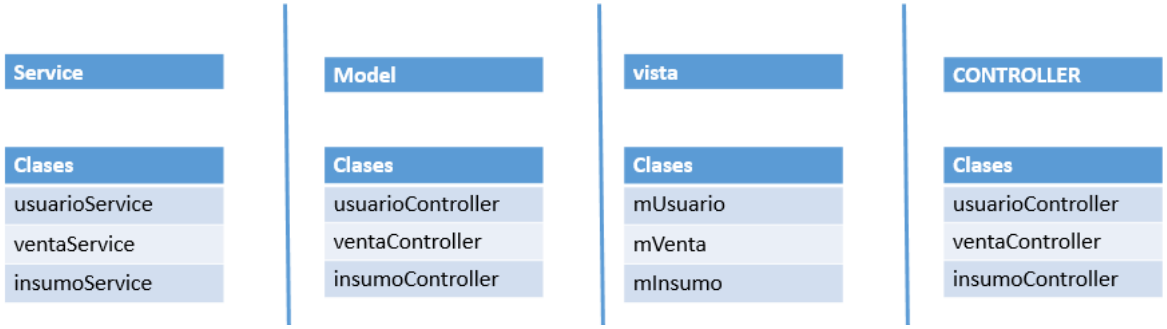

Figura 4. ejemplo de patrón de diseño MVC con la implementación de la capa "Services"

Conclusión

Al implementar patrones de diseño podremos tener una mejor organización de los componentes en un desarrollo, así como identificar de manera más puntual los recursos y disminuir los costos de mantenimiento. Un patrón de diseño se puede personalizar de acuerdo con las necesidades del proyecto teniendo las siguientes ventajas.

1. Reutilización de código.
2. Mejor estructura en el proyecto.
3. Simplicidad en el código
4. Evitar redundancia de código.
5. Mejor compresión para el programador.

Es por eso por lo que es utilizado cada vez más en cada uno de los proyectos de desarrollo.

REFERENCIAS

[1] Qué es Api Rest y por qué debes de integrarla en tu negocio- ESTER RIBAS el 29 MAYO, 2018. https://www.iebschool.com/blog/que-es-api-rest-integrar-negocio-business-tech/
[2] Patrón Modelo-Vista-Controlador. Yenisleidy Fernández Romero. http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15/10
[3] JavaScript MDN Web Docs https://developer.mozilla.org/es/docs/Web/JavaScript
[4] libro Patrones de Diseño ERICH GAMMA RICHARD HELM RALPH JOHNSON JOHN VLISSIDES (2015)

[5] libro pattern language- Christopher Alexander 1977
[6] Qué es Web 2.0. Patrones del diseño y modelos del negocio para la siguiente generación del software Autor: Tim O'Reilly. Presidente y CEO de O'Reilly Media, Inc
https://f72a491e-a-ef8ef19a-s-sites.googlegroups.com/a/espacio3i.com/cibercultura_publico/modulo-1---el-contexto-de-la-cibercultura/2-historiadeinternet/Qu%C3%A9%20es%20Web%202.pdf?attachauth=ANoY7coqVaVMoQ0NLyZZOU-FGSM-CA4Uy5hGf0SLUN2rIXrhi0iYJhwUwcTXr_-3gnbscOHQbn1oofEVr-uymGoQGTbxB_k2s-5WHygrVHidrFvBPUY8S3j5RHSSQ762rrrjBzvE_DNMusm_2bf_g1TjU-F47mqOQTGYO6ofeXAuNrN2GxJdTg8n2QWAzxeEv1sKukK_F05TAKOSncn9VkamDOci6JgBp-FRZrsKt6NtZ1zQw4ecRZP5Mu-2lfxgctf8a7adycFjWDmMAaNL4s-MYUhYuUW7UrEic_hLpaARa3BW-2M5QPGAVVv0XAHfzHorcMpcMqPY&attredirects=0

Correo electrónico autor: *rodrigo.sanchezprg@gmail.com*