<codoacodo/>

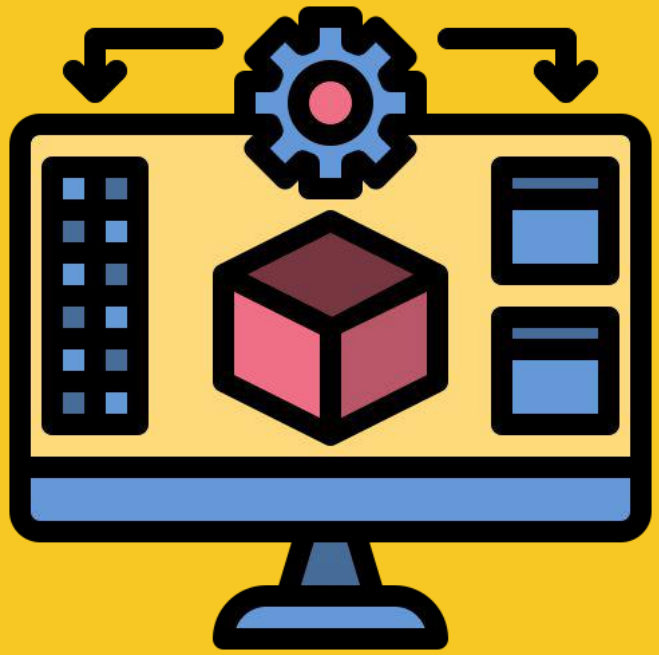
Agencia de Aprendizaje a lo largo de la vida

Desarrollo Fullstack

BA

ARQUITECTURA

Patrones



¿Qué es un Patrón de Arquitectura?

Un patrón de arquitectura es una **solución probada y documentada** a un **problema recurrente en el desarrollo de software**.

Estos patrones son utilizados para **resolver problemas comunes de diseño** y permiten a los desarrolladores construir software **escalable y robusto**.

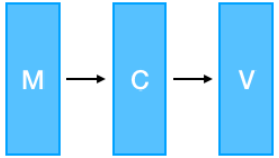
¿Qué es un Patrón de Arquitectura?

Los patrones de arquitectura son guías o plantillas que ofrecen **soluciones a problemas específicos,** y han evolucionado a lo largo del tiempo a medida que la industria del software ha enfrentado desafíos recurrentes.

Estos patrones pueden abordar cuestiones relacionadas con la organización del código, la distribución de responsabilidades entre componentes, la escalabilidad, la seguridad, la eficiencia, entre otros.

¿Qué es la escalabilidad y robustez?

- Cuando un software es escalable y robusto, significa que puede crecer con las necesidades del negocio y que garantiza su disponibilidad y funcionamiento correcto, incluso en situaciones difíciles o inesperadas.
- Es capaz de manejar cargas de trabajo crecientes y mantenerse operativo bajo diversas condiciones, lo que asegura un rendimiento óptimo y una experiencia de usuario satisfactoria en todo momento.
- La escalabilidad se refiere a la capacidad de un software para crecer y adaptarse a una mayor demanda, mientras que la robustez se refiere a su capacidad para mantenerse en funcionamiento de manera confiable y resistir situaciones adversas.



Tipos de Patrones

- **Capas**

Dividen el software en **capas**, cada una con una responsabilidad específica. Ejemplos de patrones de capas son: **MVC** (Modelo-Vista-Controlador) y **MVP** (Modelo-Vista-Presentador).

- **Basados en eventos**

Se centran en el intercambio de mensajes o eventos entre componentes. Ejemplos de patrones de eventos son: **Publicar-Suscribir**, **Observer** y **Reactor**.

- **Basados en servicios:**

Se enfocan en la creación de servicios reutilizables. Por ejemplo: **SOA** (Arquitectura Orientada a Servicios) y **REST** (Representational State Transfer).

- **Basados en microservicios**

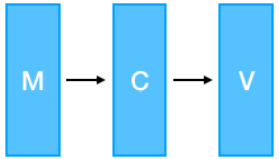
Se enfocan en la creación de una arquitectura compuesta por servicios independientes que trabajan juntos para realizar una tarea. Algunos ejemplos son: **Microservicios** y **Arquitectura Hexagonal**.

¿Qué es MVC?

Es una arquitectura de software que propone la **división** de responsabilidades de una aplicación en **3 capas diferentes**.

¿Qué plantea MVC?

- Este patrón de arquitectura lo que plantea es un scaffolding (andamiaje) o una estructura de carpetas, a través de la cual el programador va a moldear la aplicación.
- Dentro de cada carpeta se alojará el código de la aplicación.
- Este enfoque arquitectónico propone una estructura de carpetas para organizar el código de la aplicación de manera coherente y mantener una separación clara de responsabilidades entre las diferentes partes del sistema.



Principios

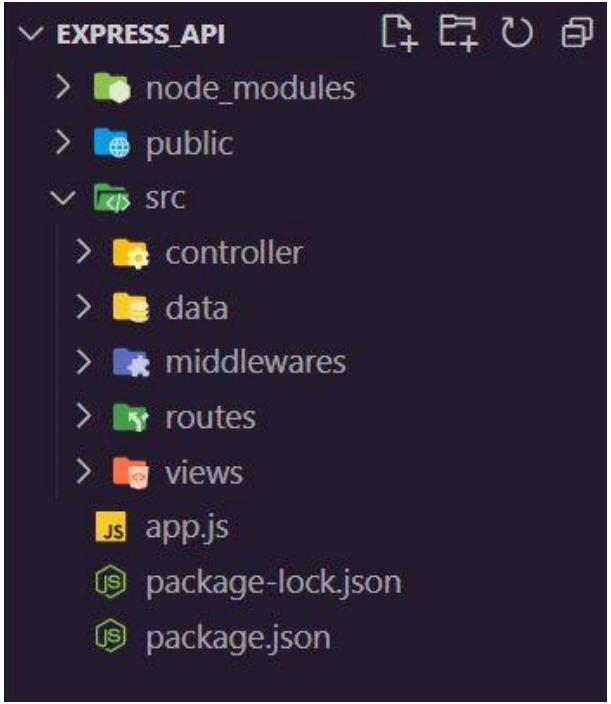
MVC es útil en sistemas donde se requiere el uso de interfaces de usuario, aunque en la práctica el mismo patrón de arquitectura **se puede utilizar para distintos tipos de aplicaciones**.

Ayuda a **crear softwares más robustos**, donde se potencie la facilidad de mantenimiento, [illegible] y la separación de conceptos

Estructura MVC

Una estructura **MVC** básica está compuesta por al menos 3 carpetas y un entry point.

En la imagen **podemos observar** el archivo **app.js** cómo entry point y las carpetas **data** (**m**odelo), **views** (**v**ista), **controller** (**c**ontrolador).
Además *se pueden tener otras carpetas* las rutas, estilos y middlewares que **veremos más adelante**.



Modelo

Capa que trabaja con los datos.

Contiene mecanismos para acceder a la información y también para actualizar su estado.

En los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

En **Node** se suele utilizar ***Sequelize***, un programa ORM que facilita el trabajo con BBDD relacionales.

Vista

Es la capa visible de nuestra aplicación.

Es el código ***HTML***, ***CSS***, ***Javascript*** necesario para renderizar y mostrar los datos e información a nuestros usuarios.

En ocasiones se suelen utilizar ***Template Engines*** o diversos frameworks como ***React*** para adoptar flexibilidad en el desarrollo.

Normalmente es la capa Front End de los proyectos y su comunicación hacia las fuentes de datos se realiza a través de los ***controladores***.

Controlador

Capa que sirve de enlace entre las vistas y los modelos

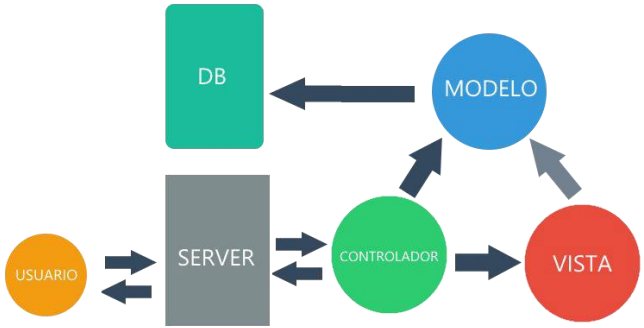
Su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida.

Normalmente contienen la lógica de nuestra aplicación junto con las condiciones o reglas de negocio.

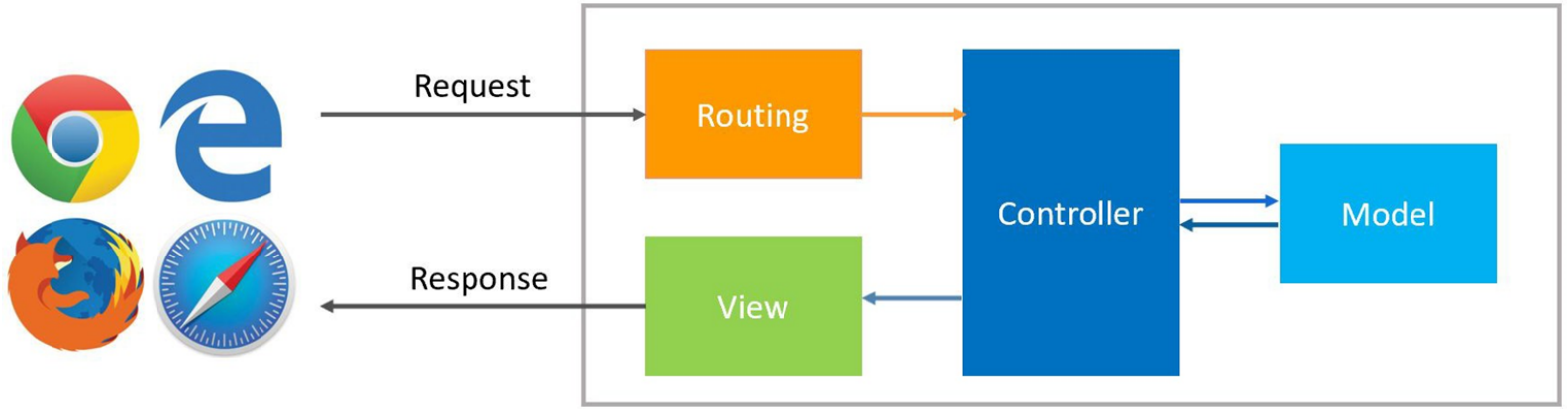
Es **invocado por nuestras rutas**, **solicita datos al modelo** y los **envía a la vista para ser renderizados**.

Recorrido MVC

1. El **cliente realiza una solicitud** a nuestro servidor.
2. El **router invoca un controlador**.
3. El **controlador solicita información al modelo** y este a la **base de datos**, **devuelve al controlador** y **retorna los datos a la vista**.
4. La vista **crea un archivo estático** y se envía al cliente. **El cliente recibe los archivos y renderiza** la aplicación.



Otro ejemplo...

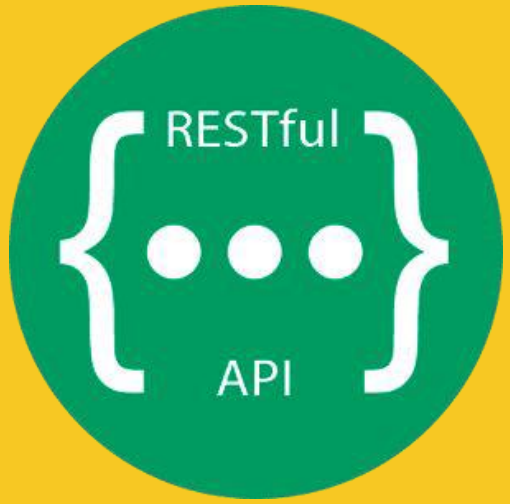


¿Qué es una API Rest?

Las **API interactúan** con **sistemas o PC 's** de manera que el sistema **comprenda la solicitud y la cumpla**.

REST no es un protocolo ni un estándar, sino **un conjunto de límites** de arquitectura. Es un tipo específico de API que usa el protocolo HTTP para la comunicación.

Es decir la información **se entrega por medio de HTTP** en uno de estos formatos: **JSON** (JavaScript Object Notation), **HTML, XLT, Python, PHP o texto sin formato.**



Analogía del restaurante

- Imagina que estás en un restaurante y deseas pedir comida. Para hacerlo, no necesitas ir a la cocina y preparar la comida tú mismo; en su lugar, le das tu pedido al mesero y él se encarga de llevarlo a la cocina, comunicarse con el chef y traerte la comida que has solicitado.
- En este caso, el mesero actúa como una interfaz entre tú (el cliente) y la cocina (donde se prepara la comida).

Analogía del restaurante

- De manera similar, en el mundo de la programación, una API (Interfaz de Programación de Aplicaciones) es como el mesero en el restaurante.
- Es una interfaz que permite que diferentes aplicaciones se comuniquen entre sí y compartan datos o funcionalidades, de manera que una aplicación puede solicitar información o realizar acciones en otra aplicación sin necesidad de conocer los detalles internos de cómo se implementa esa funcionalidad.

Analogía del restaurante

- Una API REST es un tipo específico de API que sigue un conjunto de **principios y restricciones para facilitar la comunicación** (o transferencia) entre sistemas distribuidos a través del protocolo HTTP.
- El término "REST" significa Representational State Transfer.
- **La API REST, en particular,** se puede comparar con el menú en ese restaurante: proporciona una estructura estandarizada y predefinida que permite realizar acciones específicas (como obtener información, agregar datos, actualizar o eliminar recursos) sobre los elementos del "menú" (o recursos) de manera consistente y predecible.

Principios REST

- **REST** se enfoca en exponer recursos (hacer accesibles y disponibles ciertos datos o funcionalidades) a través de URLs y utilizar los verbos **HTTP** (**GET**, **POST**, **PUT**, **DELETE**) para manipularlos.
- Se basa en la utilización de los **verbos HTTP** para realizar operaciones sobre los recursos, y en la utilización de los formatos **JSON** o **XML** para representar la información.
- Además propone un conjunto de restricciones arquitectónicas, como la interfaz uniforme, el estado sin sesión, la cacheabilidad, la visibilidad y la escalabilidad, que permiten construir sistemas web flexibles y escalables.

¿Qué necesito para crear una API Rest?

Definir los recursos:

Identificar los recursos que se van a exponer en la API RESTful, como entidades de negocio o funciones específicas.

Definir la estructura de la URL

Utilizar URLs descriptivas para cada recurso, evitando utilizar verbos o adjetivos en la URL y separando los elementos con barras diagonales.

Utilizar los verbos HTTP

GET, POST, PUT, DELETE, etc. En base a la intención para manipular los recursos. Cómo usar GET para obtener un recurso y POST para crear uno nuevo.

¿Qué necesito para crear una API Rest?

Utilizar los códigos de estado HTTP

Utilizar los códigos de estado HTTP para comunicar el resultado de la operación, como 200 para una solicitud exitosa o 404 para un recurso no encontrado.

Utilizar el formato de datos correcto

Utilizar el formato de datos adecuado para cada operación, como XML o JSON, y especificar el tipo de contenido en la cabecera HTTP.

Utilizar la documentación

Documentar la API RESTful para que los consumidores puedan entender cómo utilizarla y qué recursos están disponibles.

Utilizar la autenticación y la autorización

Proteger la API RESTful mediante la autenticación y la autorización de los usuarios que acceden a los recursos.

MVC vs REST

Característica	MVC	REST
Enfoque	Separación de preocupaciones y organización en tres componentes claramente definidos: **Modelo**, **Vista** y **Controlador**.	Exposición de recursos y utilización de verbos **HTTP** para manipularlos.
Tipo de aplicación	Principalmente aplicaciones web, aunque también puede utilizarse en otros tipos de aplicaciones.	
Manejo de solicitudes	A través del Controlador, que actúa como intermediario entre la Vista y el Modelo.	A través de los verbos HTTP: GET, POST, PUT y DELETE.
Representación de la información	Utiliza un conjunto de estructuras de datos para representar los datos en la aplicación.	Utiliza formatos como JSON o XML para representar los datos.
Escalabilidad	Escalabilidad limitada debido a la organización de la aplicación en tres componentes.	Altamente escalable debido a la utilización de los verbos HTTP y la exposición de recursos.
Reutilización de código	Permite la reutilización de código a través de la separación clara de las responsabilidades.	Permite la reutilización de código a través de la exposición de recursos y la utilización de los verbos HTTP.

Si bien ambos patrones tienen sus principios y diferencias, es posible utilizarlos juntos.