<codoacodo/>

Agencia de
Aprendizaje
a lo largo
de la vida

Desarrollo Fullstack

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 22

Introducción a Backend

- Relación Cliente/Servidor
- Protocolo HTTP
- Status Codes

Clase 23

Patrones de Arquitectura

- ¿Qué son?
- Tipos de patrones
- MVC
- REST

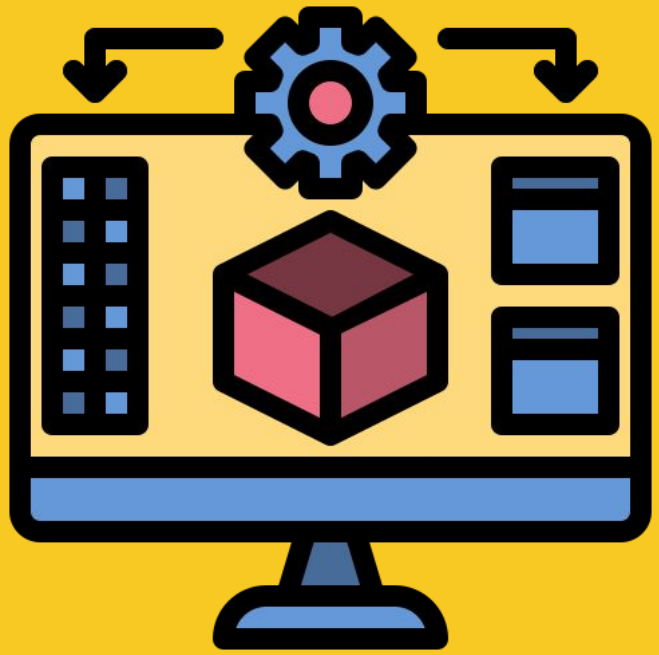
Clase 24

Node JS

- ¿Qué es?
- ¿Cómo funciona?
- Módulos

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.

Tipos de Patrones

Algunos de ellos son:

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:

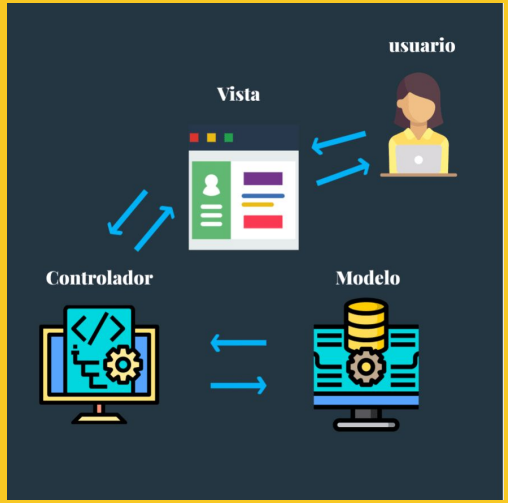
Se enfocan en la creación de servicios reutilizables. Por ejemplo: Arquitectura **SOA** (Orientada a Servicios) y **REST** (Representational State Transfer).

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.



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**.

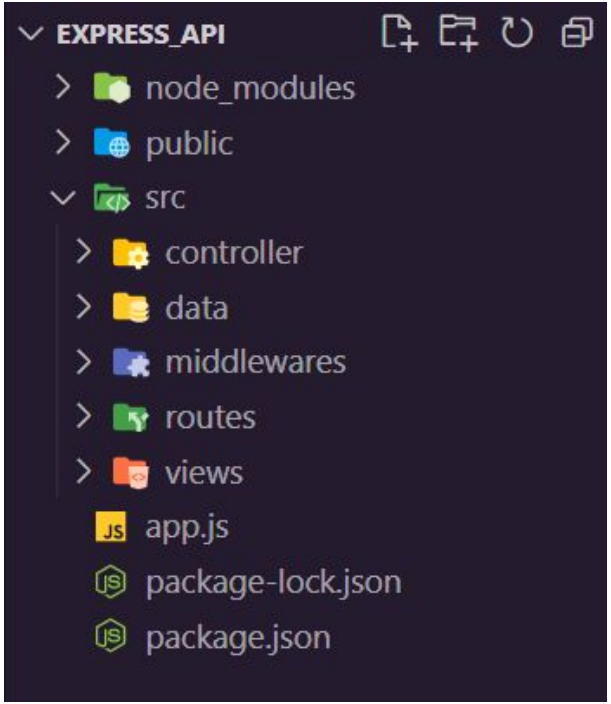
Ayuda a **crear softwares más robustos**, donde se potencie la facilidad de mantenimiento, **reutilización del código** y la separación de conceptos.

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* para 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

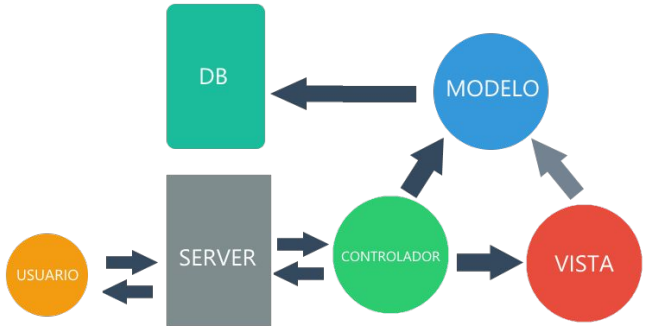
Su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida.

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

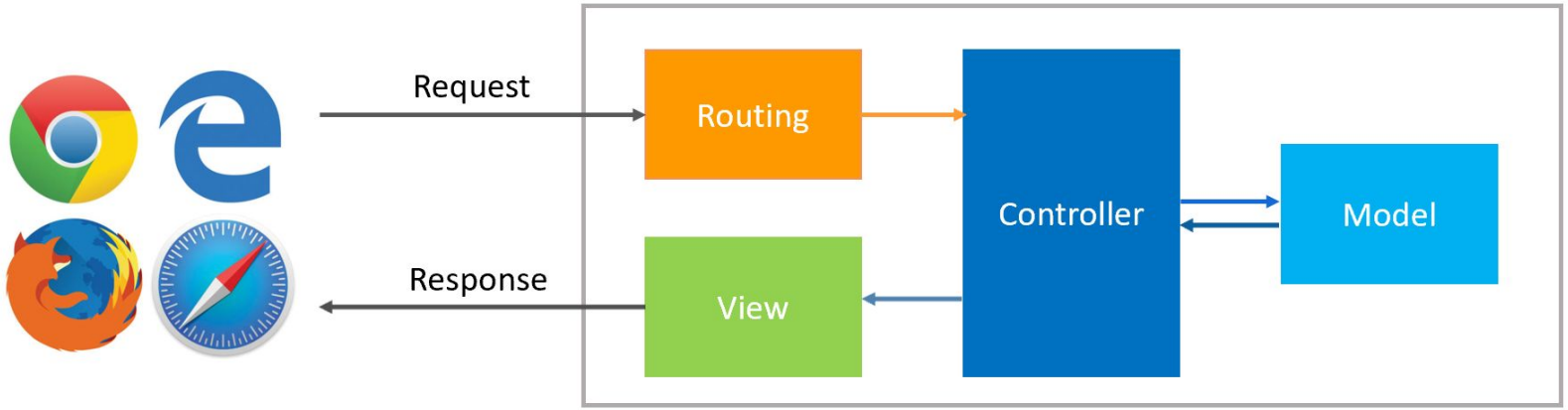
Es invocado por nuestras rutas, solicita datos al modelo y los envía a la vista para ser renderizados.

Recorrido MVC

- El usuario realiza una solicitud a nuestro servidor.
- El router invoca un controlador.
- El controlador solicita información al modelo y este a la base de datos, devuelve al controlador y retorna los datos a la vista.
- 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.

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***.

Principios REST

REST se enfoca en exponer recursos 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.

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.

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 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 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.

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.

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.

Todo en el Aula Virtual.

Gracias