Aplicación Backend

Que son y cómo utilizar los paquetes npm, peticiones HTTP y un servidor SQL para nuestra aplicación web.

Aplicación Backend

¿Qué es y para qué se usa?

Aplicación Backend

Una **aplicación backend** es la parte de una página web o app que trabaja "detrás de escena". No la ves directamente, pero hace que todo funcione.

El backend se encarga de procesar datos, guardar y buscar información en bases de datos, manejar usuarios, controlar accesos y enviar los datos correctos al frontend

Cuando haces algo en una app o página, como iniciar sesión o buscar algo, esa acción va al backend, que hace el "trabajo duro" y envía el resultado para que lo veas en el frontend.

Nuestra aplicación será hecha en NODE utilizando varios de sus paquetes.

¿Qué es un paquete de NODE?

Un **paquete de Node** es un conjunto de funciones ya hechas que alguien creó y compartió para que otros puedan usarlas en sus proyectos. Es como un "bloque" de código que puedes añadir a tu proyecto para hacer tareas específicas sin tener que escribir todo desde cero.

Cuando instalas un paquete con **npm** (Node Package Manager), lo descargas y lo añades a tu proyecto. Estos paquetes sirven para hacer cosas que necesitas, como conectarte a una base de datos, o crear un servidor web de forma más rápida y sencilla.

¿Dónde se guardan los paquetes de Node?

En Node, los paquetes que instalas se guardan en una carpeta llamada **node_modules** dentro de tu proyecto.

Cada vez que clonas un nuevo repositorio, la carpeta **node_modules** no se incluye (para no hacer el repositorio muy pesado), pero en el proyecto queda un archivo llamado **package.json** que lista todos los paquetes necesarios.

Por eso cuando se clone o descargue un repositorio de **github**, se debe correr el comando **npm i** (o npm install) para que las dependencias de paquetes sean descargadas.

Instalación de SweetAlert

Vamos a probar instalando y utilizando un paquete de Node en nuestro proyecto.

CommonJS vs ES6

Las distintas formas de importar y exportar

¿Qué son commonJs y ES6?

Estos dos son sistemas de ***módulos*** en JavaScript.

Un sistema de módulos permite organizar el código dividiéndolo en archivos más pequeños y fáciles de entender.

En lugar de tener todo en un solo archivo, se puede ***exportar e importar*** partes del código en archivos separados, haciéndolo más fácil de manejar y reutilizar.

¿Cuáles son sus diferencias?

COMMONJS

Es el sistema de módulos que utiliza Js por defecto

Utiliza **require**() y **module**.**exports**

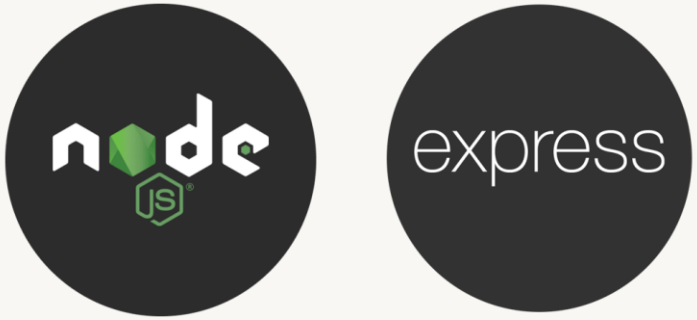
ES6 (TYPE MODULE)

Es un sistema de módulos más nuevo y estándar

Utiliza **import** y **export**

Se debe configurar poniendo *“type”: “module”* en package.json

Servidores con Express



¿Qué es Express?

Express es un paquete de Node que facilita la creación de servidores web. Con Express, puedes hacer que tu aplicación responda a peticiones como abrir páginas, enviar datos, recibir información de formularios, y mucho más, de forma rápida y organizada.

Cuando usamos Express en nuestro código, creamos una instancia de app. Esta instancia es el "cerebro" o controlador principal del servidor. Con app, podemos decirle a Express cómo debe responder a diferentes rutas (urls) y métodos (GET, POST, etc.).

Métodos del Objeto App

Cómo usar los métodos de rutas y el método use para nuestro servidor

Métodos de Rutas

En Express, los métodos como app.get, app.post, app.put, y app.delete definen rutas y cómo el servidor debe responder a diferentes tipos de ***solicitudes HTTP***. Cada uno de estos métodos recibe una ***función flecha*** con dos parámetros: req (request) y res (response).

```
// Ejemplo de una ruta hecha sin router
app.get('/home', async (req, res) => {
    res.json( { message: 'Llegaste a home!' } );
});
```

¿Qué hace la función flecha?

Esta es el ***controlador*** de la ruta, que define qué hace el servidor cuando recibe una solicitud en una dirección específica. La función flecha es anónima y *se ejecuta cada vez que alguien hace una solicitud a esa ruta*. Dentro de esta función es donde se usan req y res.

```
// Ejemplo de una función flecha (el handler) pasada por parámetro
const homeHandler = async (req, res) => {
    res.json( {message: 'Llegaste a home!'} )
}

app.get('/home', homeHandler)
```

¿Qué es el parámetro req?

Es el objeto que representa la petición que el cliente le envía al servidor. req contiene toda la información sobre lo que el cliente pide. Como es un objeto, tiene muchas “claves” útiles, como:

req.query: Datos extra que el cliente pasa en la URL (https://www.google.com/search**?q=gmail**).

req.body: Información enviada en el cuerpo de la solicitud en un objeto, útil en formularios. A diferencia de los otros dos métodos, la información no se ve en el url.

req.params: Valores de la ruta (localhost:4000/page/**:pagNum**).

¿Qué es el parámetro res?

Es el objeto que representa la respuesta que el servidor le envía de vuelta al cliente. Con res, decides qué información enviar de regreso, como:

res.send(): Para enviar texto o HTML.

res.json(): Para enviar datos en formato JSON, útil en APIs (generalmente usamos este).

res.status(): Para definir el código de estado HTTP, como 200 (OK) o 404 (No encontrado).

¿Para qué se usan los métodos de rutas?

APP.GET RESPONDE CUANDO ALGUIEN QUIERE "PEDIR" O VER DATOS, COMO AL ABRIR UNA PÁGINA.

APP.POST ES PARA "ENVIAR" DATOS AL SERVIDOR, COMO AL ENVIAR UN FORMULARIO O LOGEARSE.

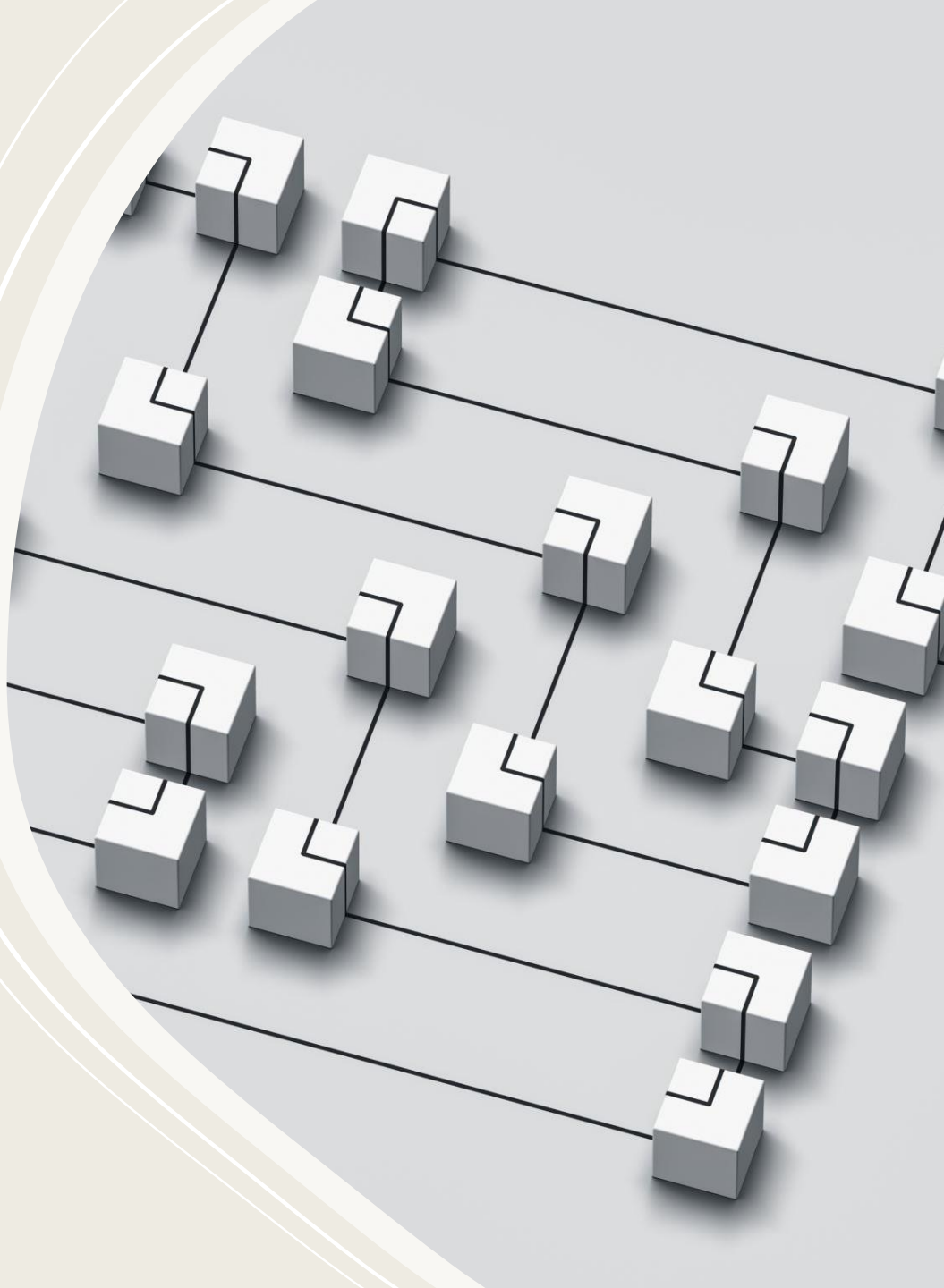
APP.PUT Y **APP.DELETE** SIRVEN PARA ACTUALIZAR O ELIMINAR INFORMACIÓN EN EL SERVIDOR.

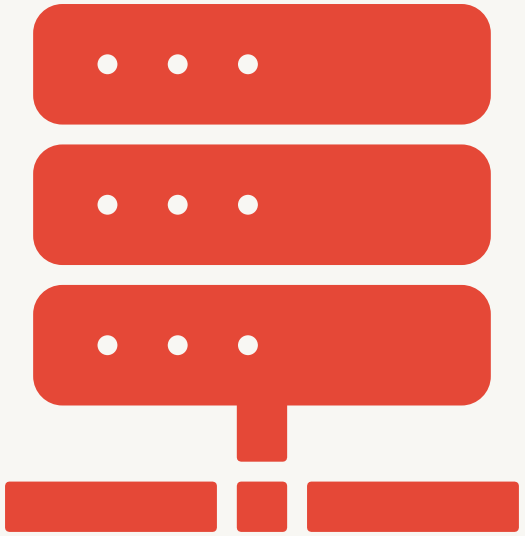
Intervalo para Crear Rutas

¿Qué es un router?

Un router en Node.js es como un mini app que nos ayuda a gestionar rutas específicas de forma **modular**.

En lugar de poner todas las rutas en un solo archivo, se pueden crear **routers** para cada sección y mantener el código organizado.





Método app.use

app.use: Este método sirve para "conectar" ***middlewares*** y ***routers*** al servidor.

Un middleware es código que se ejecuta antes de las rutas.

Cuando usas app.use con routers, estás diciendo a Express que todas las rutas dentro de ese router deben responder desde un camino específico.

Intervalo para crear un Router