Express

Express se autodescribe como:

Framework web minimalista, rápido y sin opiniones para Node.js.

Express.js, a veces también llamado «Express», es un [framework de backend](https://kinsta.com/es/blog/bibliotecas-javascript/) Node.js minimalista, rápido y similar a Sinatra, que proporciona características y herramientas robustas para desarrollar aplicaciones de backend escalables. Te ofrece el sistema de enrutamiento y características simplificadas para ampliar el framework con componentes y partes más potentes en función de los casos de uso de tu aplicación.

##### CARACTERISTICAS:

PERMITE ROUTING

ES ENFOCADO EN ALTO RENDIMIENTO

NOS PERMITE DESARROLLAR APLICACIONESDE NODE.JS RAPIDDAMENTE CON CODIGO MAS CONCISO

### CONCEPTOS BASICOS:

#### CRUD:

Es un acrónimo que representa las operaciones básicas que se realizan en el tratamiento de información.

A la izquierda cada el significado de cada letra de crud- a la derecha su equivalente como verbo HTTP.

Create = POST

Read = GET

Update =PUT

Delete = DELETE

#### API:

Aplication programming interface:

Es una interfaz intermediaria entre dos software que permita que uno se pueda nutrir del otro, es decir pretende brindar ciertos beneficios a otro software.

##### API REST:

REST: Representational State Transfer

:

REST es un modelo para usar APIs con ciertas características.

Una API REST trabaja con solicitudes HTTP y operaciones de CRUD

##### RESTful API:

Se le llama RESTfulAPI a las APIs que usan el modelo REST y que ya están implementadas funcionando en la web.

##### IMPORTACION DE EXPRESS:

*const* expressImportName = require('express');

##### USO DE EXPRESS:

*const* appName = expressImportName ();

##### ROUTING:

Se le conoce como routing en express, a el tratamiento de las diferentes funcionalidades que podemos realizar entre un cliente y un servidor.

La sintaxis general del routing es:

app.METHOD(PATH, HANDLER)

app = nuestra aplicación:

METHOD =un método HTTP:

PATH : la ruta URL

HANDLER: como vamos a tratar la información (lleva dos parámetros: request y response.)

EJEMPLO DE SINTAXIS:

appName.get('/URL', ()*=>*{CODIGO})

appName.get('/URL', functionName)

ejemplo :

*const* expressImport = require("express");

*const* app = expressImport();

app.get("/", (*req*, *res*) *=>* {*res*.send('Primer server 💛');});

*const* PUERTO = 3000;

app.listen(PUERTO,(*req*,*res*)*=>*{console.log(`Servidor escuchando en puerto ${PUERTO}`);});

por consola:



En el navegador:



## request y response:

**request** y **response** son dos objetos que se reciben como parámetros cuando se trabaja con peticiones cliente servidor.

se usan en la construcción de diferentes métodos de express:

### REQUEST:

hace referencia a información o una consulta que podríamos recibir del usuario

esta compuesta por:

endpoint

headers

body

#### REQUEST PARAMS:

permiten generalizar parte de una URL, y lograr una declaracion mas dinamica de las URL

SINTAXIS:

appName.get("/url:ParameterName ", (*req*, *res*) *=>* {});

EJEMPLO 1:

*const* expressImport = require("express");

*const* app = expressImport();

app.get('/carpeta1/:loQueSea',(*req*,*res*)*=>*{

  console.log(*req*.params);

  console.log(*req*.params.loQueSea);

*res*.send('hola');

})

*const* PUERTO = 3000;

app.listen(PUERTO, () *=>* {  console.log(`Servidor escuchando en puerto ${PUERTO}`)});

cuando ingresemos a una URL como esta:



En el navegador veremos:



Por consola veremos:



EJEMPLO 2:

*const* expressImport = require("express");

*const* app = expressImport();

app.get('/inicio/:a/:b',(*req*,*res*)*=>*{

  console.log(*req*.params.a);

  console.log(*req*.params.b);

*res*.send(`${*req*.params.a} y ${*req*.params.b}`);

*const* valorA = *req*.params.a;

  console.log(typeof(valorA));

  console.log(typeof(*req*.params.b));

})

*const* PUERTO = 3000;

app.listen(PUERTO, () *=>* {  console.log(`Servidor escuchando en puerto ${PUERTO}`)});

cuando ingresemos a una URL como esta:



En el navegador veremos:



Por consola veremos:



EJEMPLO 3:

app.get('/:algo/:a/:b',(req,res)=>{

  const {a, b} *=* req.params;

  console.log(a);

  console.log(b);

  console.log(*typeof*(a));

  console.log(*typeof*(b));

  res.send(` suma de ${a} + ${b} = ${parseInt(a)*+*parseInt(b)}`);

  console.log(*typeof*(a));

  console.log(*typeof*(b));

});

cuando ingresemos a una URL como esta:



En el navegador veremos:



Por consola veremos:



EJEMPLO 4:

*const* expressImport = require("express");

*const* app = expressImport();

*const* { gods } = require("./dioses");

app.get('/:cultura', (*req*, *res*) *=>* {

*const* cult = *req*.params.cultura;

  if (cult === "griegos") {*res*.send(gods.griegos)}

  else if (cult === "egipcios") {*res*.send(gods.egipcios)}

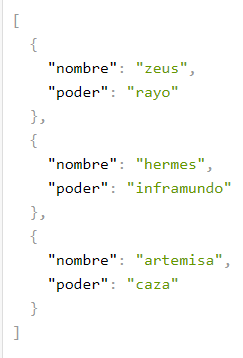
  else{*res*.status(404).send(`La cultura ${cult} no ha sido encontrada`)}

});

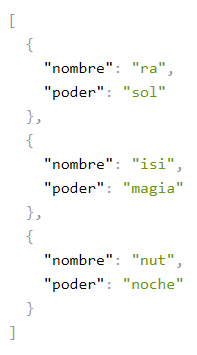
*const* PUERTO = 3000;

app.listen(PUERTO,(*req*,*res*)*=>*{console.log(`Servidor escuchando en puerto ${PUERTO}`);});













#### REQUEST QUERY:

Para acceder a los parámetros querys ,lo hacemos mediante la propiedad query del objeto req.

SINTAXIS:

*req*.query



Consultar la query entera.

app.get('/', (*req*, *res*) *=>* {console.log(*req*.query)});

*const* PUERTO = 3000;

app.listen(PUERTO,(*req*,*res*)*=>*{console.log(`Servidor escuchando en puerto ${PUERTO}`);});

si ingresamos una url como esta:



Esto veriamos por consola:



Si ingresamos una url como esta:



Veriamos esto por consola:



¡!notese que a diferencia del ejemplo anterior la clave se repetirá en ambas querys y asi mismo se agruparon los valores en la misma clave.

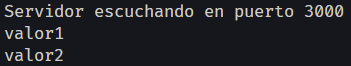
Consultar los valores en las claves de los parámetros query:

app.get('/', (*req*, *res*) *=>* {

  console.log(*req*.query.clave1);

  console.log(*req*.query.clave2);

});



Probando condicionales con querys.

app.get("/productos", (req, res) => {

*if* (req.query.consulta *===* "precio") {

    res.send("Enseguida le informamos el valor del producto");

  } *else* {

    res.send("Lo siento esa informacion no esta disponible");

  }

});









#### Request body:

MIDDLEWARE:

app.use(exp.json()); ///transforma todo a Json

POST:

app.post('/url',(req,res)=>{

  console.log(req.body);

  res.send("holis");

});S

si enviamos una solicitud post a esta url:



Con este contenido en el body como cuerpo



Por consola veríamos



En en navegador veríamos :



\*Si no hubiésemos usado el middleware el objeto req.body se vería como undefined\*

#### Otros métodos del objeto request:

##### url:

// +Permite acceder a al URL utilizada

app.get("/camino1/camino2", (req, res) => {

  console.log(` La ruta visitada es : ${req.url}`);

});



##### method:

// +Permite acceder al metodo HTTP utilizado

app.get("/camino1/camino2", (req, res) => {

  console.log(` El metodo usado es : ${req.method}`);

});



### RESPONSE:

hace referencia a la respuesta que puede enviarse al usuario.

Diferencias entre:

Res.send

Res.json

Pero la principal diferencia entre res.json y res.send aparece cuando tienes que pasar objetos que no son como respuesta. res.json también convertirá objetos que no sean (por ejemplo, nulo, indefinido, etc.) que en realidad no son un JSON válido, mientras que res.send no los convertirá.

Res.end

Podemos usar res.end() si queremos finalizar la respuesta sin proporcionar ningún dato. Esto podría ser útil para una página 404, por ejemplo:

Pero, ¿y si queremos enviar algunos datos y finalizar la respuesta? La respuesta es simple, res.send() (y recuerde, res.json()) ambos nos permiten enviar algunos datos y también finalizan la respuesta, por lo que no es necesario llamar explícitamente a res.end().

## all method:

//# ALL METHOD:

//+ Permite generara una respuesta para todos los metodos HTPP en el URL indicado

app.all('/',(req,res)=>{res.send('Buenas')});

si accedemos a la URL indicada:



Obtendremos esta respuesta.



Y siempre obtendremos la misma respuesta independientemente del método HTTP que utilizemos:

…GET-PUT-PATCH-DELETE-POST…

## MIDDLEWARES:

UN MIDDLEWARE ES UNA INSTANCIA INTERMEDIARIA ENTRE DOS PROCESOS.

En este caso lo podemos entender como una acción que se lleva a cabo en medio de peticiones cliente/servidor.

Es decir: interponiéndose antes de que el servidor obtenga la información del cliente

o tambien

Interponiéndose antes de que el servidor mande la respuesta al cliente.

Las funciones middewares se ejecutan:

**DESPUES** de recibir una solicitud.

**ANTES** de enviar una respuesta

Tienen acceso al objeto de la solicitid, al objeto de la respuesta a **next** que es una funcion que se llama para ejecutar el proximo middleware

Para hacer uso de middlewares express tiene una propiedad llamada USE

### USE:

cuando creemos nuestra aplicación de express, o mejor dicho instancia de express, podremos modificar su uso antes o después de tener que procesarla en el futuro…

para eso existe una propiedad de nuestra instancia de express llamada **use.**

Con esta podemos modificar que pasa antes o después de que se realice un intercambio entre el cliente y el servidor.

SINTAXIS:

app.use('/URL',(req,res,next)=>{CODE});

el tercer parámetro next se usa cuando queremos indicar que puede continuar con la siguiente instrucción, de lo contrario el servidor quedara esperando infinitamente.

Ejemplo 1:

//# importar express:

const expressImport *=* require("express");

//# Usar express:

const app *=* expressImport();

app.use('/',(req,res,next)=>{

  console.log('La peticion paso por aqui') /// primero muestra esto.

  next() /// ahora si, se puede pasar a la siguiente instrucción.

});

app.get('/',(req,res)=>{res.send('Buenas')}); ///siguiente instrucción

const PUERTO *=* 3000;

app.listen(PUERTO, () => {

  console.log(`Servidor escuchando en puerto ${PUERTO}`);

});

al visitar la URL indicada:



Veriamos primero esto por consola



Y luego esto en el navegador.



Ejemplo 2.1:

Esto va a pasar con todas las sub rutas que nosotros utilizemos después de la ruta donde inidcamos el middleware.

//# importar express:

const expressImport *=* require("express");

//# Usar express:

const app *=* expressImport();

app.use('/inicio',(req,res,next)=>{

  console.log('La peticion paso por aqui previamente')

  next()

});

app.get('/',(req,res)=>{res.send('Buenas')});

const PUERTO *=* 3000;

app.listen(PUERTO, () => {

  console.log(`Servidor escuchando en puerto ${PUERTO}`);

});















Pero si nosotros accedemos a una URL previa:







No veremos el mensaje puesto que estos empiezan a parir de el url indicado es decir:

“/inicio”

Ejemplo 2.2:

Por otro lado en un ejemplo como este:

//# importar express:

const expressImport *=* require("express");

//# Usar express:

const app *=* expressImport();

app.use('/:cualquiercosa',(req,res,next)=>{

  console.log('La peticion paso por aqui')

  next()

});

app.get('/',(req,res)=>{res.send('que tal')});

const PUERTO *=* 3000;

app.listen(PUERTO, () => {

  console.log(`Servidor escuchando en puerto ${PUERTO}`);

});

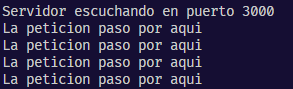
Pasaría absolutamente con todas las rutas después de la principal











## SET:

La propiedad set permite que nosotros ingresemos valores de configuración para usar en nuestros aplicaciones de express.

De esta forma nosotros podemos asegurar que tenemos todas las variables deseadas guardadas en la misma propiedad.

Algunas de estas “claves” son palabras reservadas.

SINTAXIS:

app.set('NAME', VALUE);

EJEMPLO 1:

const exp *=* require("express");

const app *=* exp();

app.set('PUERTO', 3000);

app.set('Server', 'Servidor con express');

console.log(`${app.get('Server')} en puerto ${app.get('PUERTO')}`);



EJEMPLO 2:

app.set('PUERTO', 3000);

app.set('puerto', '3000');

app.set('Server', 'Super Server');

const PUERTO\_NUMBER *=* app.get('PUERTO');

const puerto\_string *=* app.get('puerto');

const servidor\_name *=* app.get('Server');

console.log(PUERTO\_NUMBER);

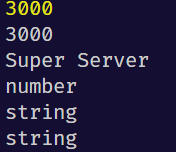
console.log(puerto\_string);

console.log(servidor\_name);

console.log(*typeof*(PUERTO\_NUMBER));

console.log(*typeof*(puerto\_string));

console.log(*typeof*(servidor\_name));



EJEMPLO 3:

//-configuraciones manuales con set:

app.set("PUERTO", 3000);

app.set("Server", "Server super Express");

//-configuraciones reservadas con set:

app.set("case sensitive routing", true);

app.listen(

  app.get("PUERTO"),

  console.log(`${app.get("Server")} - port: ${app.get("PUERTO")}`)

);

app.get("/inicio", (req, res) => {

  res.send("Funciona");

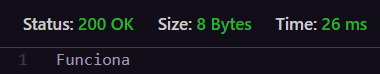
});

En esta caso estamos usando una configuración reservada de la propiedad use

La misma nos permite tener URL sensitivas a mayúsculas o minúsculas

Algo que por defecto no pasa:





Pero si cambiamos tan solo un carácter, ya no funcionaria.





### PUERTOS DINAMICOS:

En caso de que nuestra aplicación este en un servicio externo de hosting generalmente el puerto se va a asignar automáticamente de forma dinamica y no va a tener un valor fijo, para tomar este puerto que se suele asignar de forma dinámica se utiliza el siguiente método :

process.env.PORT

este método permite recibir el valor del puerto que debe estar de forma dinámica,

de todas formas nosotros podemos tambien asignar un puerto fijo como alternativa si es que no existe este valor previo.

*Opción 1:*

*const* PUERTODINAMICO = process.env.PORT;

*Opción 2:*

*const* PUERTODINAMICO = process.env.PORT || 3000;

## rutas URL:

Supongamos que queremos establecer diferentes respuestas para cada path de nuestra URL.

*//# Creando mas rutas:*

*const* { gods } = require("./dioses");

app.get("/api/", (*req*, *res*) *=>* {*res*.send("Bienvenido a la api de dioses griegos y egipcios 👿😈");});

app.get("/api/dioses", (*req*, *res*) *=>* {*res*.send(gods)});

app.get("/api/dioses/griegos", (*req*, *res*) *=>* {*res*.send(gods.griegos)});

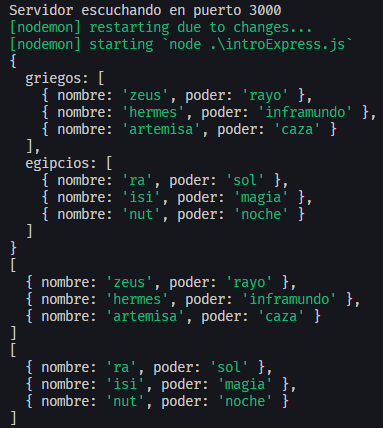
app.get("/api/dioses/egipcios", (*req*, *res*) *=>* {*res*.send(gods.egipcios)});

console.log(gods);

console.log(gods.griegos);

console.log(gods.egipcios);

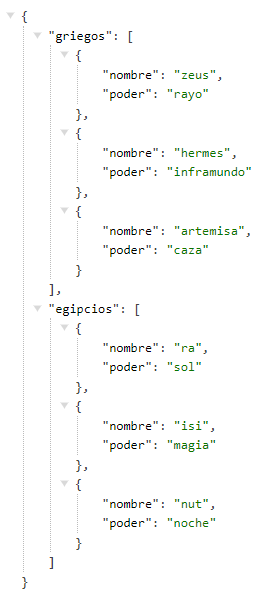
vista por consola:



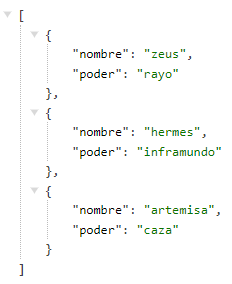
Vista por navegador:



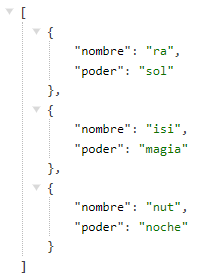












## Routers:

Permiten reusar parte de una URL.

SINTAXIS:

Para importar:

variableName.Router();

para usar:

app.use('/URL,variableNameReference);

variableNameReference.get('/',(*req*,*res*)*=>*{ *CODIGO* });

***EJEMPLO:***

*const* expressImport = require("express");

*const* app = expressImport();

*const* { productos} = require("./datos");

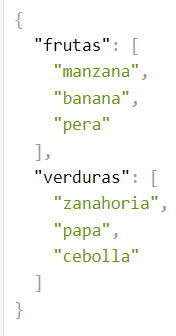
*const* routerNew = expressImport.Router();

app.use('/productos',routerNew);

ejemplo 1:

routerNew.get('/',(*req*,*res*)*=>*{*res*.send(productos)});





ejemplo 2:

routerNew.get('/:algo',(*req*,*res*)*=>*{

  prod = *req*.params.algo

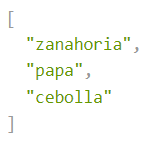
  if (prod ==='frutas') { *res*.send(productos.frutas)}

  else if (prod ==='verduras') { *res*.send(productos.verduras)}

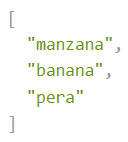
  else { *res*.send(`el producto "${prod}" no se ha encontrado`)}

});













*const* PUERTO = 3000;

app.listen(PUERTO,(*req*,*res*)*=>*{console.log(`Servidor escuchando en puerto ${PUERTO}`)});

## Roouters en distintos archivos:

##### Archivo principal:

*const* expressImport = require("express");

*const* app = expressImport();

*const* {personas} = require("./datos.js");

*const* routerNombres = require("./informacion/nombresRouter.js");

*const* routerApellidos = require("./informacion/apellidosRouter.js");

app.use('/datos/nombres',routerNombres);

app.use('/datos/apellidos',routerApellidos);

*// Routing*

app.get('/', (*req*, *res*) *=>* {*res*.send('Bienvenido al servidor de nombres y apellidos ♥.')});

app.get('/datos', (*req*, *res*) *=>* {*res*.send((personas));});

*const* PUERTO = 3000;

app.listen(PUERTO,()*=>*{console.log(`Servidor escuchando en puerto ${PUERTO}`)});

##### Archivo 1:

*const* expressImport = require("express");

*const* app = expressImport();

*//$ sintaxis 1:*

*const* { personas} = require("../datos.js");

*//$ sintaxis 2:*

*// const { nombres} = require("../datos.js").personas;*

*const* routerNombres = expressImport.Router();

*// routerNombres.get('/',(req,res)=>{res.send(personas.nombres)})*

routerNombres.get('/', (*req*, *res*) *=>* {

*res*.send(JSON.stringify(personas.nombres));

});

*//$ sintaxis 1:*

module.exports = routerNombres;

*//$ sintaxis 2:*

*// module.exports.routerNombres = routerNew;*

##### Archivo 2:

*const* expressImport = require("express");

*const* app = expressImport();

*const* { personas} = require("../datos.js");

*const* routerApellidos = expressImport.Router();

routerApellidos.get('/',(*req*,*res*)*=>*{*res*.send(personas.apellidos)})

*// module.exports.routerApellidos = routerNew;*

module.exports = routerApellidos;

##### archivo 3 :

datosPersonales = {

  'nombres': ['Jorje','Eustaquio','Rigoberta'],

  'apellidos':['Valdez' ,'Galindez', 'Jalapeño']

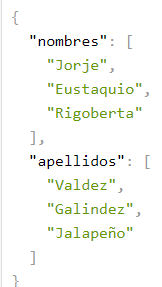
};

module.exports.personas = datosPersonales;

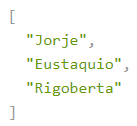




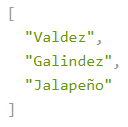












##### TIP:

Si nosotros exportamos una variable como propiedad del objeto exports

Esta misma se tiene que indicar entre llaves.

De lo contrario va sin llaves

Ejemplo:

Si exportamos asi:

module.exports.variableName = objectName;

se tiene que importar asi:

*const* { variableName } = require("./URL");

pero si exportamos asi:

module.exports = objectName;

se tiene que importar asi:

*const* variableName = require("./URL");

## POST PUT PATCH DELETE:

app.post('/datos',(*rea*,*res*)*=>*{res.send('Agregando datos')});

app.put('/datos',(*rea*,*res*)*=>*{res.send('Actualizando datos con put')});

app.patch('/datos',(*rea*,*res*)*=>*{res.send('Actualizando datos con path')});

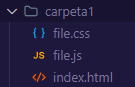
app.delete('/datos',(*rea*,*res*)*=>*{res.send('borrando datos')});

## STATIC:

La función static permite importar todos los archivos de una carpeta.

Cuenta como una función middleware ya que influye en el funcionamiento de nuestro servidor

Supongamos una carpeta que contiene 3 archivos:





<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta http-equiv="X-UA-Compatible" content="IE=edge">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>HTML DOCUMENT</title>

  <link rel="stylesheet" href="file.css">

  <script src="file.js"> </script>

</head>

<body>

  <H1>SUPER TITTLE</H1>

</body>

</html>



console.log('soy el archivo JavaScript de la carpeta  "carpeta1" ');



body{

  background-color: aquamarine;

  color: burlywood;

}

Si quisiéramos hacer uso de cada uno de estos con la función use estaríamos haciendo más líneas de código de las necesarias puesto que existe una función que permite importar toda una carpeta de archivos estáticos.

SINTAXIS:

express.static('./URL', [options])

EJEMPLO 1:

app.use(expressImport.static('./carpeta1')); /// importa todo de la "carpeta1"

si visitamos esta url:





Y por consola



Podemos ver que se ejecuto el html el css y el JS

EJEMPLO 2:

Si visitamos esta URL:

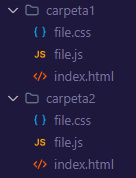


Veriamos lo mismo,





EJEMPLO 3:



Notese que aunque los archivos se llaman igual su ruta es diferente y por eso no se superponen

app.use(('/archivos1'),expressImport.static('./carpeta1'));

app.use(('/archivos2'),expressImport.static('./carpeta2'));







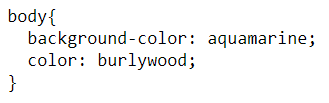






ALGO A TENER EN CUENTA ES QUE EL METODO STATIC CONLLEVA CIERTA EXPOSICION DE LAS CARPETAS PUBLICAS QUE USEMOS:

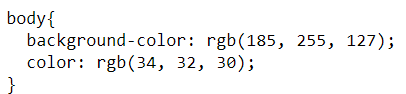
















COMO SE PUEDE OBSERVAR SE PUEDE ACCEDER A LOS ARCHIVOS CONOCIENDO SU NOMBRE Y LA URL.

También es necesario tener en cuenta el orden donde se coloca este middleware, puesto que este influye en el comportamiento de la aplicación.

### PATH:

**En caso de que tengamos todo dentro de un carpeta contenedora que no sea la carpeta raíz necesitaremos importar el módulo path de express**



##### Dirname:

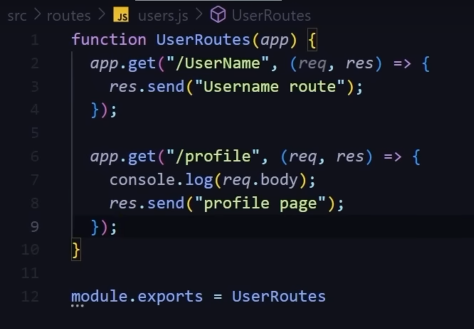
la constante “\_\_dirname”

contiene la URL exacta donde nosotros estamos ejecutando NODE

console.log(\_\_dirname);

“C:\Users\Documents\EXPRESS\codigos express\subCarpeta"

De esta forma nosotros podemos concatenar el nombre de un archivo a esta URL.







### REST API CON EXPRESS:

para hacer una simulacion de una api rest con express debemos al menos poder realizar las operaciones basicas de crud, obstante tenemos que poder:

crear elementos desde cero

leer o buscar elementos

actualizar o modificar elementos

y poder borrar elementos.

Supongamos que vamos a tener un catálogo de compras de productos de supermercado,

Todos los productos tienen que tener un **ID**entificador único

Y además:

Nombre:

Precio:

Fecha de vencimiento

Marca

Stock:

EJEMPLO:



//. import:

const exp *=* require("express");

const app *=* exp();

//. set port:

app.listen(4000);

const catalogo *=* require("./catalogo.js");

//. Middleware:

app.use(exp.json()); ///transforma todo a json

//\_ get:

// //- obtener todos los productos

app.get("/catalogo", (req, res) => {

  res.json(catalogo);

});

// //- obtener un solo producto

app.get("/catalogo/:id", (req, res) => {

  const idSearch *=* catalogo.productos.find((e) => e.id *==* req.params.id); ///Comparando IDentificador

  //+ En caso de  error o exito:

*if* (*!*idSearch) *return* res.status(404).send("producto no encontrado");

  res.json(idSearch);

});

// //\_ post:

//- ingresar productos

app.post("/catalogo", (req, res) => {

  const nuevoProducto *=* { *...*req.body, id: catalogo.productos.length *+* 1 };

  catalogo.productos.push(nuevoProducto);

  res.send(nuevoProducto);

  console.log(req.body);

});

// //\_ put:

// //- actualizar productos

app.put("/catalogo/:id", (req, res) => {

  const newData *=* req.body;

  //+ Encontrando el producto:

  const idSearch *=* catalogo.productos.find((e) => e.id *==* req.params.id);

  //+ Informando busqueda:

*if* (*!*idSearch) *return* res.status(404).send("producto no encontrado");

  //+ Renovando La lista

  //$ metodo 1:

  //+REEMPLAZA TODO EL CONTENIDO

  const nuevoProducto *=* {

    id: parseInt(req.params.id),

*...*req.body

  };

  catalogo.productos[idSearch] *=* nuevoProducto;

  res.send(catalogo.productos[idSearch]);

  //$ metodo 2:

  //+REEMPLAZA SOLO LAS PROPIEDADES DEL BODY Y MANTIENE EL RESTO

  const nuevoProducto *=* catalogo.productos.map(p =>( p.id *==* req.params.id) *?* {*...*p,*...*newData} *:* p );

  catalogo.productos *=* nuevoProducto;

  //+respuesta final:

  res.send("actualizando productos");

});

//\_ delete:

//- borrar productos

app.delete("/catalogo/:id", (req, res) => {

  //+ Encontrando el producto:

  const idSearch *=* catalogo.productos.find((e) => e.id *==* req.params.id);

  //+ Informando busqueda:

*if* (*!*idSearch) *return* res.status(404).send("producto no encontrado");

  //+ Eliminando producto

  const nuevoArreglo *=* catalogo.productos.filter((a) => a.id *!==* idSearch.id);

  //+ Renovando La lista

  catalogo.productos *=* nuevoArreglo;

  res.send(nuevoArreglo);

});



let inventario *=* {

  productos: [

    {

      id: 1,

      nombre: "galletitas de chocolate",

      precio: "180$",

      vencimiento: "02/2025",

      marca: "OREO",

      stock: "15 unidades",

    },

    {

      id: 2,

      nombre: "Tapas de tarta",

      precio: "170$",

      vencimiento: "08/2024",

      marca: "LA CLARITA",

      stock: "7 unidades",

    },

    {

      id: 3,

      nombre: "Pan Lactal con semillas",

      precio: "170$",

      vencimiento: "07/2023",

      marca: "BIEMBOENO",

      stock: "10 unidades",

    },

    {

      id: 4,

      nombre: "Mermelada de maracuya",

      precio: "237$",

      vencimiento: "04/2028",

      marca: "MERMEX",

      stock: "22 unidades",

    },

    {

      id: 5,

      nombre: "Queso Roquefort",

      precio: "400$",

      vencimiento: "02/2026",

      marca: "HONGO AZUL",

      stock: "3 unidades",

    },

    {

      id: 6,

      nombre: "Papel Higienico",

      precio: "34$",

      vencimiento: "No Disponible",

      marca: "ASS CLEAN",

      stock: "45 unidades",

    },

    {

      id: 7,

      nombre: "Mentitas de arandano",

      precio: "160$",

      vencimiento: "07/2026",

      marca: "MENTIS",

      stock: "77 unidades",

    },

  ],

};

module.exports *=* inventario;