CODER HOUSE

Clase 8. Programación Backend

Router & Multer

OBJETIVOS DE LA CLASE

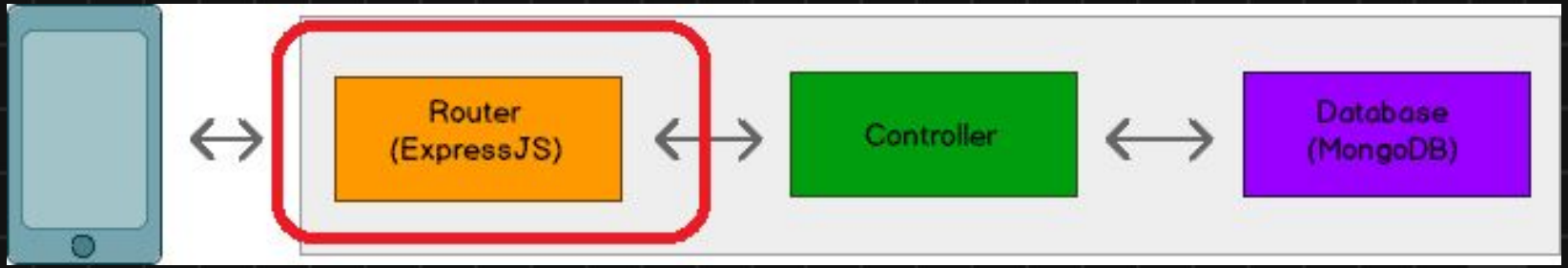
- Configurar el servicio de recursos estáticos en Express
- Creación y uso de capas middlewares.
- Conocer el mecanismo de envío de datos de un formulario al servidor.
- Subir archivos al servidor

CRONOGRAMA DEL CURSO

Clase 7	Clase 8	Clase 9
Express Avanzado	**Router & Multer**	**Motores de plantillas**

CODER HOUSE

Express Router



Ruteo en Express

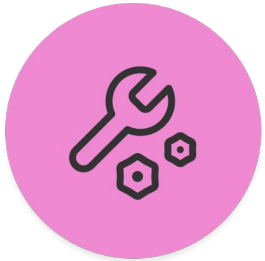
- La clase ***Router*** se usa para crear un nuevo objeto de enrutador, que es una instancia aislada de middleware y rutas. Se utiliza cuando se desea crear un nuevo objeto de enrutador para manejar solicitudes.

- El **Router** de express nos permite crear múltiples "mini aplicaciones" para que se pueda asignar un espacio de nombre al api público, autenticación y otras rutas en sistemas de enrutamiento separados.

Ejemplo Router en Express

```
const express = require('express')
const { Router } = express

const app = express()
const router = Router()

router.get('/recurso, (req, res) => {
   res.send('get ok')
})

router.post('/recurso', (req, res) => {
   res.send('post ok')
})

app.use('/api', router)

app.listen(8080)
```

CODER HOUSE



Express router

Vamos a practicar lo aprendido hasta ahora

Tiempo: 10 minutos

CODER HOUSE

- Crear un servidor que permita manejar una lista de mascotas y personas. Debe poseer dos rutas principales: '/mascotas' y '/personas', las cuales deben incluir métodos para listar y para agregar recursos:

GET: devolverá la lista requerida en formato objeto.

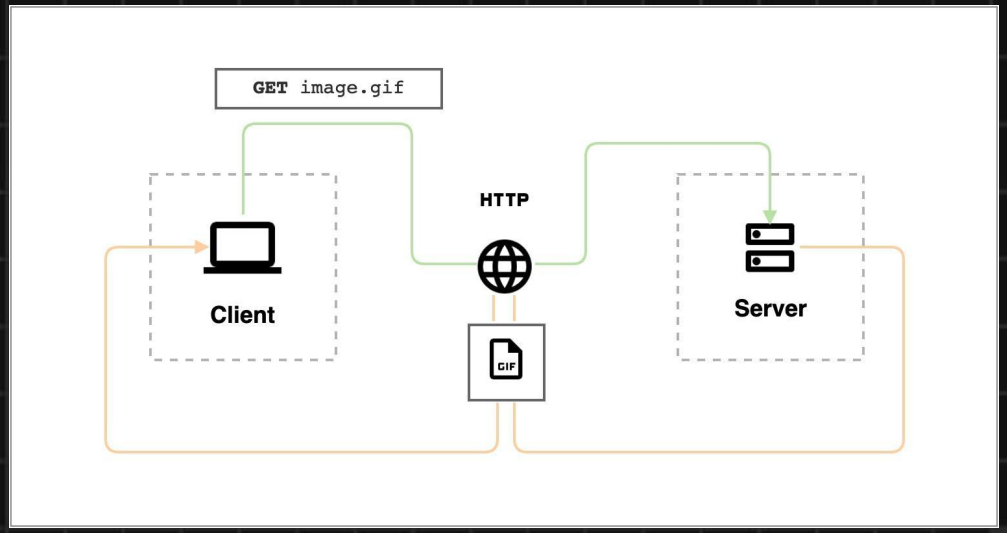
POST: permitirá guardar una persona ó mascota en arrays propios en memoria, con el siguiente formato:

Persona -> { "nombre": ..., "apellido": ..., "edad":... }

Mascota -> { "nombre":..., "raza":..., "edad":... }

- Utilizar el Router de express para definir las rutas base, implementando las subrutas en los métodos correspondientes.
- Probar la funcionalidad con Postman.
- El servidor escuchará peticiones en el puerto 8080 y mostrará en la consola un mensaje de conexión que muestre dicho puerto, junto a los mensajes de error si ocurriesen.

Servicio de archivos estáticos en Express



Fuente: https://expressjs.com/

Introducción

- Para el servicio de archivos estáticos (por ejemplo imágenes, archivos CSS y archivos JavaScript) se utiliza la función de middleware incorporada **express.static**.
- Esta función recibe como parámetro el nombre del directorio que contiene los activos estáticos.
- El siguiente código configura el servicio de imágenes, archivos CSS y archivos JavaScript en un directorio denominado public:

```
app.use(express.static('public'));
```

A continuación podemos cargar los archivos que hay en el directorio **public**:

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

Nota: *Express busca los archivos relativos al directorio estático, por lo que el nombre del directorio estático no forma parte del URL.*

Múltiples directorios

Para utilizar **varios directorios de activos estáticos** se invoca la función de middleware express.static varias veces:

```
app.use(express.static('public'));
app.use(express.static('files'));
```

Nota: *Express busca los archivos en el orden en el que se definen los directorios estáticos con la función de middleware express.static.*

Prefijo virtual

Para crear un **prefijo virtual** (donde el path de acceso no existe realmente en el sistema de archivos) para los archivos servidos por express.static, debemos **especificar un path de acceso de montaje** para el directorio estático:

```
app.use('/static', express.static('public'));
```

CODER HOUSE

Así podemos cargar los archivos que hay en el directorio **public** desde el prefijo **/static.**

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

Path absoluto

El path que se proporciona a la función **express.static** es relativo al directorio desde donde inicia el proceso node.
Por eso si ejecutamos la aplicación Express desde cualquier otro directorio, es más seguro utilizar el path absoluto del directorio al que desea dar servicio:

```
app.use('/static', express.static(__dirname + '/public'));
```

Re Su Men

- Para servir archivos estáticos (Imágenes, CSS, archivos JS, etc.) usaremos la función **express.static** middleware.

- Pasar el nombre del directorio que contiene los recursos a **express.static** para servir los archivos directamente.

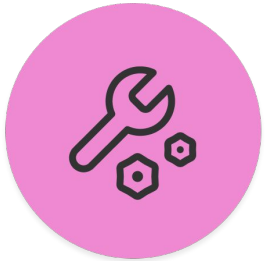
- Podemos usar varios directorios, simplemente llamando a **express.static** varias veces. Express busca archivos en el orden en que establece los directorios con express.static.

Re Su Men

- Podemos crear un prefijo de ruta virtual (uno donde la ruta no existe realmente en el sistema de archivos) con **express.static**, solo tenemos que especificar una ruta de montaje.

- Todas las rutas anteriores han sido relativas al directorio desde donde ejecutó el proceso de node. Por lo tanto, generalmente es más seguro utilizar la ruta absoluta del directorio que desea servir.

- Podemos combinar las opciones de este método: *Ruta Absoluta a Directorio* y *Ruta Virtual*

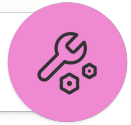


Carpeta public

Tiempo: 10 minutos

CODER HOUSE

Desafío generico



- Partiendo del ejercicio anterior, generar una carpeta pública 'public' en el servidor, la cual tendrá un archivo index.html.
- En ese archivo se encontrarán dos formularios: uno que permita ingresar mascotas y otro personas utilizando el método post
- Probar el ingreso de datos mediante los formularios y con Postman
- Verificar los datos cargados en cada caso.

Capas Middleware

Introducción

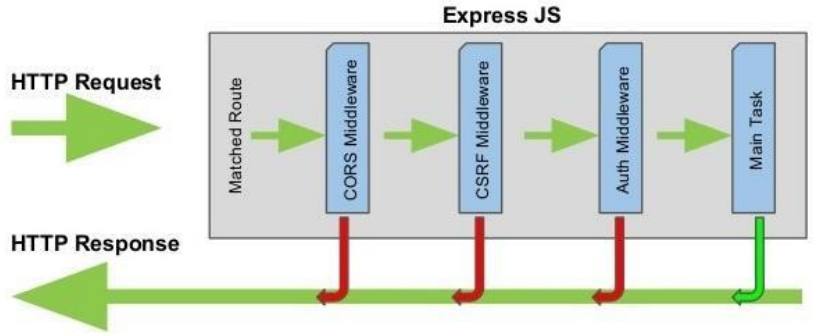
Las funciones de middleware son aquellas que tienen acceso al objeto de solicitud (**req**), al objeto de respuesta (**res**) y a la siguiente función de middleware en el ciclo de solicitud/respuestas de la aplicación (**next**)

Funcionalidad

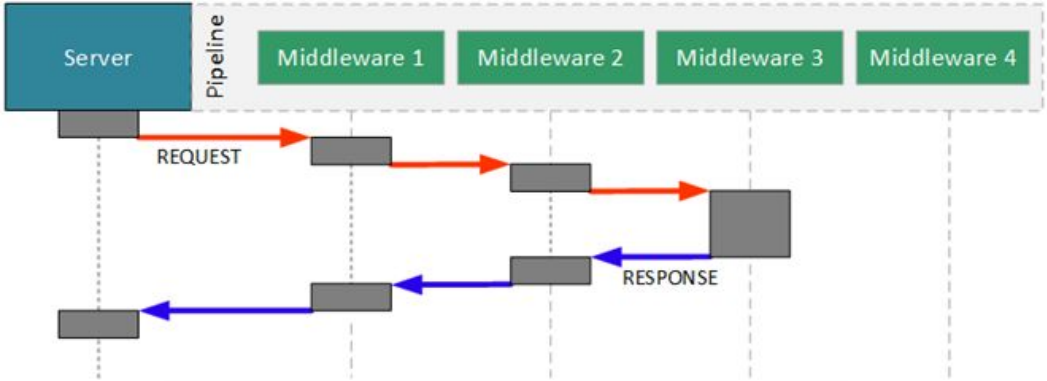
Las funciones de middleware pueden realizar las siguientes tareas:

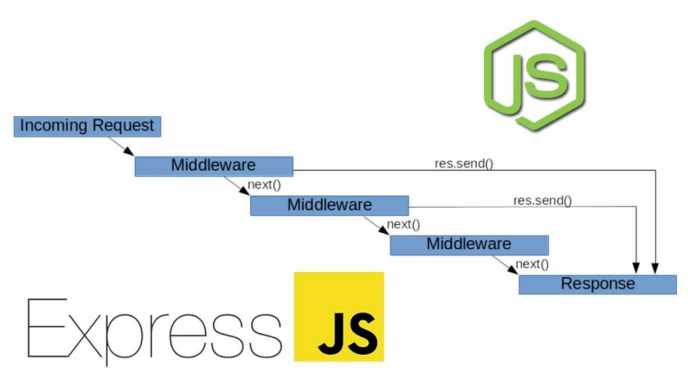
- ➢ Ejecutar cualquier código.
- ➢ Realizar cambios en la solicitud y los objetos de respuesta.
- ➢ Finalizar el ciclo de solicitud/respuestas.
- ➢ Invocar la siguiente función de middleware en la pila.

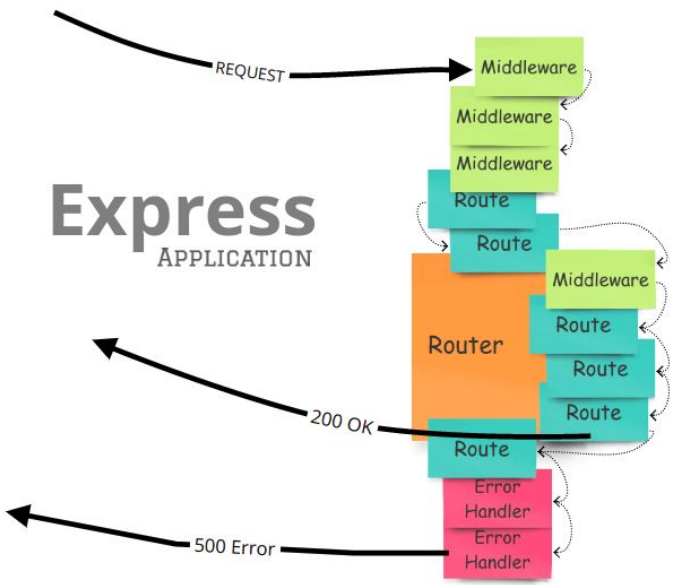
.



Nota: Se debe invocar a next() para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada.







CODER HOUSE

Tipos de middleware

Una aplicación Express puede utilizar los siguientes tipos de middleware:

- **Middleware a nivel de aplicación**
- **Middleware a nivel del Router**
- **Middleware de manejo de errores**
- **Middleware incorporado**
- **Middleware de terceros**

Veamos cada uno de ellos.

Middleware de nivel de aplicación

Este ejemplo muestra una función de middleware **sin** ninguna **vía** de **acceso** de **montaje**. La función se ejecuta cada vez que la aplicación recibe una solicitud.

```
var app = express();

app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
 });
```

CODER HOUSE

Middleware a nivel de ruta

Se pueden **agregar una o múltiples funciones** middlewares en los **procesos** de **atención de las rutas** como se muestra a continuación:

```
function miMiddleware1(req,res,next) {
    req.miAporte1 = 'dato aportado por el middleware 1'
    next()
}
function miMiddleware2(req,res,next) {
    req.miAporte2 = 'dato aportado por el middleware 2'
    next()
}
//Ruta GET con un Middleware
app.get('/miruta1', miMiddleware1, (req, res) => {
    let miAporte1 = req.miAporte1
    res.send({ miAporte1 })
})
//Ruta GET con dos Middleware
app.get('/miruta2', miMiddleware1, miMiddleware2, (req, res) => {
    let { miAporte1, miAporte2 } = req
    res.send({ miAporte1, miAporte2 })
})
```

Middleware a nivel del Router

El middleware de nivel de router funciona de la misma manera que el middleware de nivel de aplicación, excepto que está **enlazado a una instancia de express.Router()**.

```
var app = express();
var router = express.Router();

// funcion middleware sin via de acceso de montaje. El codigo es
ejecutado por cada peticion al router
router.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
 });
```

Middleware de manejo de errores

Estas funciones se definen de la misma forma que otras funciones de middleware, excepto que llevan cuatro argumentos en lugar de tres, específicamente con la firma (err, req, res, next):

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something broke!');
 });
```

Middleware incorporado

La única función de middleware incorporado en Express es express.static. Esta función es responsable del servicio de archivos estáticos:

```
app.use(express.static('public', options));
```

- **express.static(root, [options])**
  - El **argumento root** especifica el directorio raíz desde el que se realiza el servicio de activos estáticos.
  - El **objeto options** opcional puede tener las siguientes propiedades: dotfiles, etag, extensions, index, lastModified, maxAge, redirect, setHeaders

Middleware de terceros

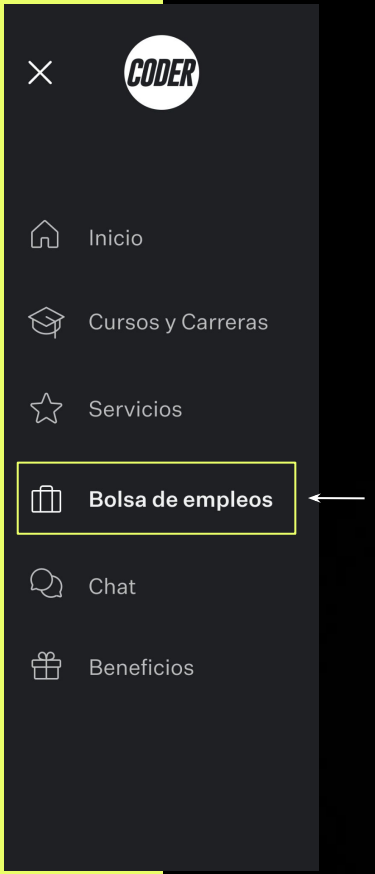
Podemos **instalar y utilizar middlewares de terceros** para añadir funcionalidad a nuestra aplicación. El uso puede ser **a nivel de aplicación o** a nivel de **Router**. Por ejemplo, instalamos y usamos la función de middleware de análisis de cookies cookie-parser.

```
$ npm install cookie-parser
```

```
var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');

// load the cookie-parsing middleware
 app.use(cookieParser());
```

BREAK

¡5/10 MINUTOS Y VOLVEMOS!



Nuevo

¡Lanzamos la Bolsa de Empleos!

Un espacio para seguir **potenciando tu carrera** y que tengas más **oportunidades de inserción laboral**.

Podrás encontrar la **Bolsa de Empleos** en el menú izquierdo de la plataforma.

Te invitamos a conocerla y ¡postularte a tu futuro trabajo!

Conócela

CODERHOUSE

Subir archivos: Multer

CODER HOUSE

¿Qué es Multer?

- Cuando un cliente web sube un archivo a un servidor, generalmente lo envía a través de un formulario y se codifica como *multipart/form-data*.
- **Multer** hace que sea fácil manipular este *multipart/form-data* cuando tus usuarios suben archivos.

  - ➢ **Multer** es un middleware para Express.
  - ➢ Un middleware es una pieza de software que conecta diferentes aplicaciones o componentes de software.
  - ➢ En Express, un middleware procesa y transforma las peticiones entrantes en el servidor.
  - ➢ **Multer** actúa como un ayudante al cargar archivos.

Multer
Ejemplo práctico de uso

CODER HOUSE

Configuración del proyecto

1. Creamos un archivo server.js. En el mismo, inicializamos todos los módulos. Crearemos una aplicación Express y crearemos un servidor para conectarse a los navegadores.

```
//npm install express express multer

// call all the required packages
const express = require('express')
const multer = require('multer');
app.use(express.urlencoded({extended: true}))

//CREATE EXPRESS APP
const app = express();

//ROUTES WILL GO HERE
app.get('/', function(req, res) {
    res.json({ message: 'WELCOME' });
});

app.listen(3000, () => console.log('Server started on port 3000'));
```

Crear el código del cliente

2. Creamos un archivo *index.html*
3. Este archivo contendrá los diferentes formularios que utilizaremos para cargar nuestros diferentes tipos de archivos.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>MY APP</title>
</head>
<body>
    <!--  SINGLE FILE -->
    <form action="/uploadfile" enctype="multipart/form-data" method="POST">
        <input type="file" name="myFile" />
        <input type="submit" value="Upload a file" />
    </form>
    <!-- MULTIPLE FILES -->
    <form action="/uploadmultiple" enctype="multipart/form-data" method="POST">
        Select images: <input type="file" name="myFiles" multiple>
        <input type="submit" value="Upload your files" />
    </form>
</body>
</html>
```

Servir el código del cliente

4. Modificamos server.js escribiendo una ruta GET que muestre el archivo index.html en lugar del mensaje "WELCOME" ("BIENVENIDO", en español).

```
// ROUTES
app.get('/',function(req,res){
    res.sendFile(__dirname + '/index.html');
});
```

Almacenamiento con Multer

5. Multer ofrece la opción de almacenar archivos en el disco. Definimos una ubicación de almacenamiento para nuestros archivos.
6. Configuramos Multer con esas opciones.

```
//server.js
// SET STORAGE
var storage = multer.diskStorage({
    destination: function (req, file, cb) {
      cb(null, 'uploads')
    },
    filename: function (req, file, cb) {
      cb(null, file.fieldname + '-' + Date.now())
    }
})
var upload = multer({ storage: storage })
```

CODER HOUSE

Manejo de carga de archivos

CODER HOUSE

Subiendo un solo archivo

En el archivo index.html, definimos un atributo de acción que realiza una petición POST. Ahora necesitamos crear un punto final en la aplicación Express. Abrimos el archivo server.js y agregamos el siguiente código:

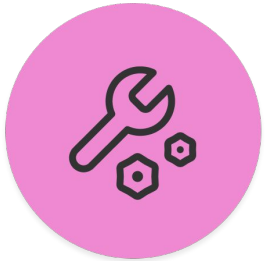
```
//Tener en cuenta que el name (nombre, en español) del campo archivo debe
//ser el mismo que el argumento myFile pasado a la función upload.single.
app.post('/uploadfile', upload.single('myFile'), (req, res, next) => {
    const file = req.file
    if (!file) {
      const error = new Error('Please upload a file')
      error.httpStatusCode = 400
      return next(error)
    }
    res.send(file)
})
```

CODER HOUSE

Subiendo múltiples archivos

Cargar varios archivos con Multer es similar a cargar un solo archivo, pero con algunos cambios.

```
//Uploading multiple files
app.post('/uploadmultiple', upload.array('myFiles', 12), (req, res, next) => {
    const files = req.files
    if (!files) {
      const error = new Error('Please choose files')
      error.httpStatusCode = 400
      return next(error)
    }
    res.send(files)
})
```



Express y Multer

Tiempo: 10 minutos

CODER HOUSE

- Crear un servidor que permita elegir y subir un archivo utilizando un formulario servido desde su espacio público.
- Dicho archivo se almacenará en una carpeta propia del servidor llamada 'uploads'.
- El nombre del archivo guardado se formará con el nombre original anteponiéndole un timestamp (Date.now()) seguido con un guión. Ej: 1610894554093-clase1.zip
- Utilizar express y multer en un proyecto de servidor que escuche en el puerto 8080.

API RESTful

API RESTful

Formato: link a un repositorio en Github con el proyecto cargado.
Sugerencia: no incluir los node_modules

Desafío entregable

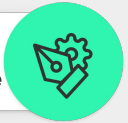
>> Consigna: Realizar un proyecto de servidor basado en node.js y express que ofrezca una API RESTful de productos. En detalle, que incorpore las siguientes rutas:

- GET '/api/productos' -> devuelve todos los productos.
- GET '/api/productos/:id' -> devuelve un producto según su id.
- POST '/api/productos' -> recibe y agrega un producto, y lo devuelve con su id asignado.
- PUT '/api/productos/:id' -> recibe y actualiza un producto según su id.
- DELETE '/api/productos/:id' -> elimina un producto según su id.

API RESTful

Formato: link a un repositorio en Github con el proyecto cargado.
Sugerencia: no incluir los node_modules

Desafío entregable



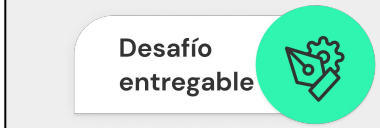
- Cada producto estará representado por un objeto con el siguiente formato:

```
{
    title: (nombre del producto),
    price: (precio),
    thumbnail: (url al logo o foto del producto)
}
```

- Cada ítem almacenado dispondrá de un id numérico proporcionado por el backend, comenzando en 1, y que se irá incrementando a medida de que se incorporen productos. Ese id será utilizado para identificar un producto que va a ser listado en forma individual.

API RESTful

Formato: link a un repositorio en Github con el proyecto cargado.
Sugerencia: no incluir los node_modules



- Para el caso de que un producto no exista, se devolverá el objeto:
  { error : 'producto no encontrado' }
- Implementar la API en una clase separada, utilizando un array como soporte de persistencia en memoria.
- Incorporar el Router de express en la url base '/api/productos' y configurar todas las subrutas en base a este.
- Crear un espacio público de servidor que contenga un documento index.html con un formulario de ingreso de productos con los datos apropiados.
- El servidor debe estar basado en express y debe implementar los mensajes de conexión al puerto 8080 y en caso de error, representar la descripción del mismo.
- Las respuestas del servidor serán en formato JSON. La funcionalidad será probada a través de Postman y del formulario de ingreso.

CODER HOUSE

¿PREGUNTAS? 

¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

-Express Router

- express.static

- Capas Middleware

- Uso de Multer

OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN

CODER HOUSE