Esta clase va a ser

- grabada

Clase 08. DESARROLLO AVANZADO DE BACKEND

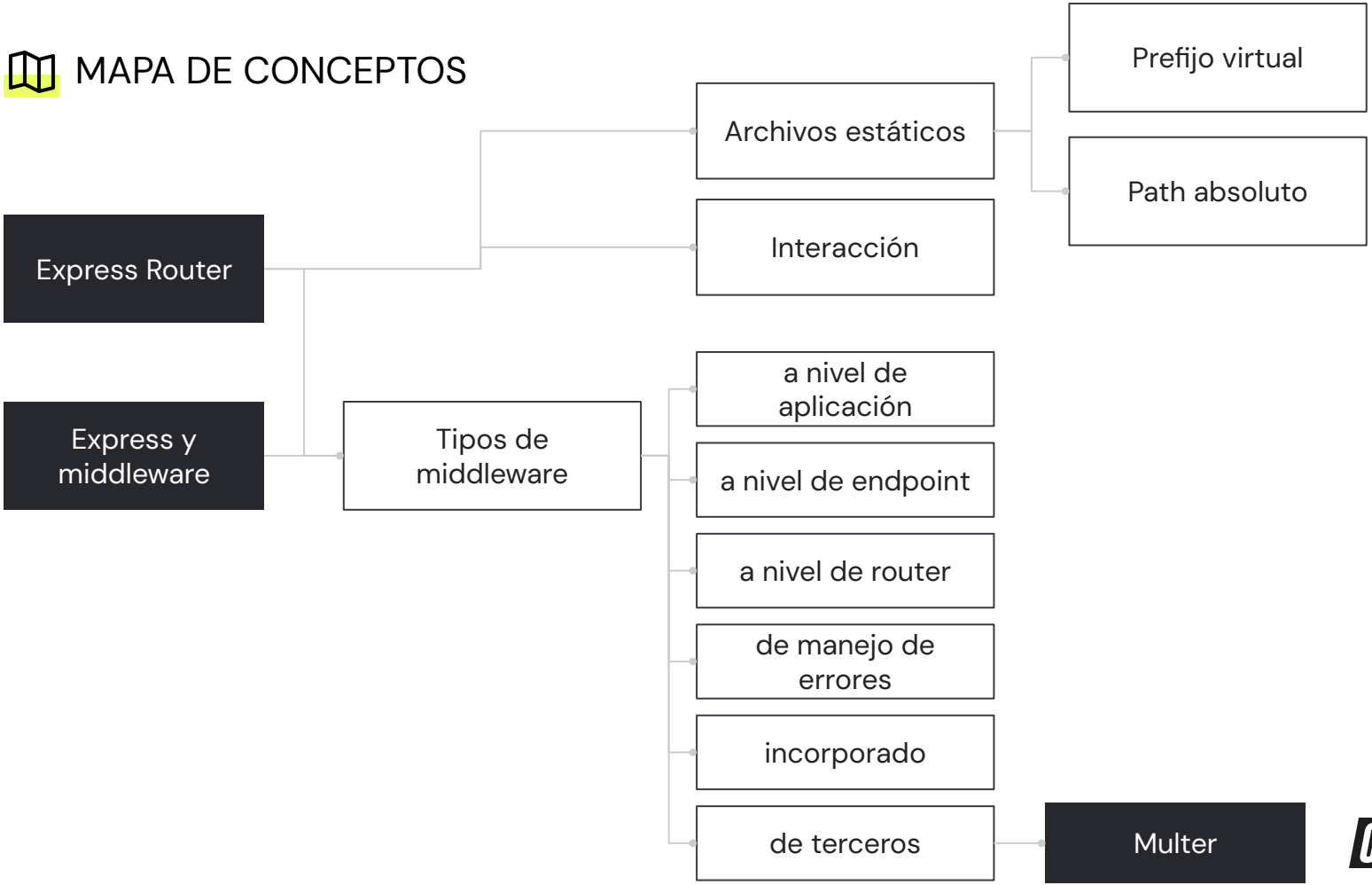
Router y Multer

CODERHOUSE

Objetivos de la clase

- Conocer la implementación Router en Express
- Implementar un sistema de recursos estáticos
- Comprender el uso de Middlewares
- Manejar Multer para carga de archivos.

MAPA DE CONCEPTOS

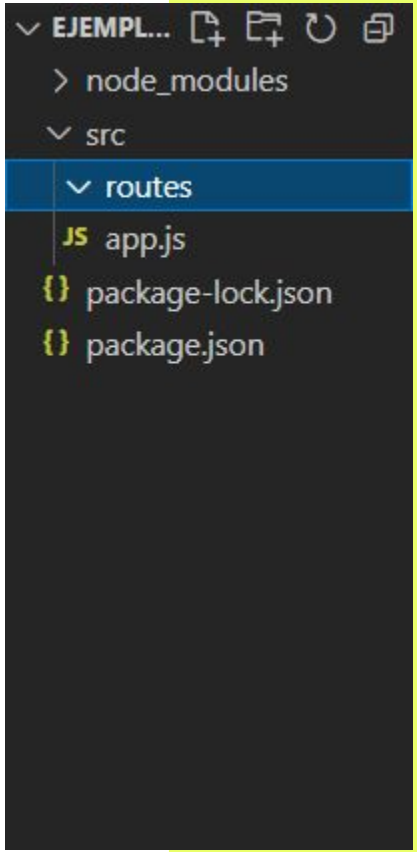


Express Router

CODERHOUSE

¿Cómo aplicar un router?

Ahora, agregaremos una carpeta “routes” donde vivirán nuestros diferentes routers (Nota que app.js se queda fuera de routes, pero sigue dentro de src).



CODERHOUSE

HANDS ON LAB

Express Router

¿Cómo lo hacemos? **Se crearán dos routers: users y pets.**

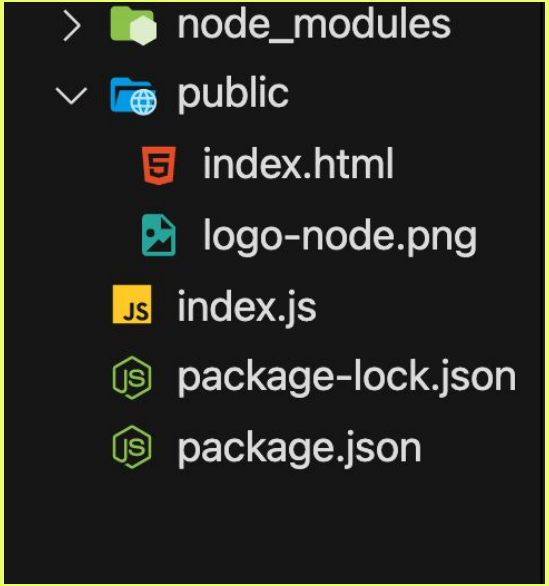
- ✓ El router de users debe tener la ruta principal /api/users
- ✓ El router de pets debe tener la ruta principal /api/pets
- ✓ Ambos deben tener, de manera interna, un array para almacenarlos.
- ✓ Ambos deben contar con un método get en su ruta raíz para poder obtener el arreglo.
- ✓ Ambos deben contar con un método POST en su ruta raíz para poder agregar un usuario o mascota según sea el router.
- ✓ Conectar los routers al archivo app.js para tener listo el apuntador al router.
- ✓ Probar funcionalidad con Postman.

Servicio de archivos estáticos con Express

Archivos estáticos

¿Cómo funciona?

- ✓ Nuestro servidor tiene la posibilidad de alojar recursos que pueden ser visibles para el cliente de manera directa.
- ✓ Podemos configurar una carpeta para que el usuario pueda acceder y ver dichos recursos de manera directa sólo con acceder a la ruta donde se encuentra ubicada.
- ✓ En este curso y en proyectos profesionales podrás encontrar estos archivos en la carpeta “public”, haciendo referencia como dice el nombre, a recursos públicos de fácil acceso para el cliente.



ACTIVIDAD EN CLASE

Carpeta public

Partiendo del ejemplo anterior, recrear la estructura con un index.html para poder visualizarse en la ruta raíz.

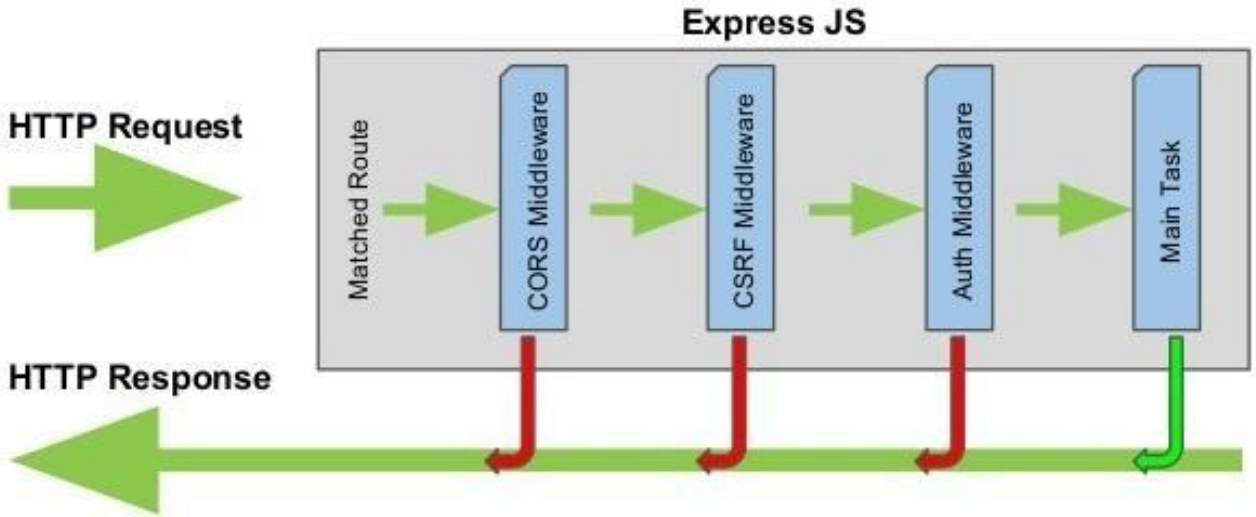
- ✓ En este archivo deberá haber un formulario donde podremos ingresar una mascota a partir del método POST. Dicho POST conectará al endpoint raíz del router pets
- ✓ Configurar el router pets para que pueda recibir el json por parte del formulario (recordar express.json() y express.urlencoded({extended:true}))
- ✓ Verificar con POSTMAN que la información llegue al servidor y se guarde correctamente.

Break

¡10 minutos y volvemos!

Express y middlewares

Flujo de múltiples middlewares a través de una petición



Este “main Task” sería el endpoint con tu servicio”.

CODERHOUSE

¡Importante!

Como lo ves en el diagrama anterior, los middlewares se ejecutan EN ORDEN, eso quiere decir que, si algún middleware depende de que se haya realizado otra operación ejecutada por un middleware previo, los coloquemos en cascada según prioridad.

Tipos de Middlewares

Middleware de nivel de aplicación

```
const app = express();

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

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.

Middleware de nivel de endpoint

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

```
function mid1(req, res, next) {
    req.dato1 = 'un dato'
    next()
}

function mid2(req, res, next) {
    req.dato2 = 'otro dato'
    next()
}
```

```
app.get('/ruta1', mid1, (req, res) => {
    res.json({
        dato1: req.dato1
    })
})
app.get('/ruta2', mid1, mid2, (req, res) => {
    res.json({
        dato1: req.dato1,
        dato2: req.dato2
    })
})
```

Middleware de 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().

```
const app = express();
const 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());
```

¿Qué es multer?

Multer es un **middleware de terceros**, pensado para poder realizar carga de archivos al servidor.

En ocasiones el cliente necesitará subir una imagen, un vídeo o un archivo, según sea nuestra aplicación, ello nos lleva a configurar nuestro servidor para soportar estos archivos y poder almacenarlos en donde nosotros le indiquemos.

Al ser de terceros, necesitaremos instalarlo para poder utilizarlo.

Instalación y configuración de MULTER

¡Importante!

Cuando subimos un archivo (imagen, vídeo, etc), estamos hablando de un flujo de datos. lo cual no puede plasmarse en un JSON. Cuando enviamos información a un endpoint donde sabemos que utilizamos MULTER, debemos enviarlo como **FormData**, no como JSON.

ACTIVIDAD EN CLASE

Express + multer

Basado en el formulario para ingresar una mascota al sistema:

- ✓ Configurar el formulario para añadir un campo input type=”file” name “file” para que la mascota a agregar pueda tener una “imagen representativa”.
- ✓ El nombre del archivo guardado se formará con el nombre original anteponiéndole un timestamp (Date.now()) seguido con un guión. Ej: 1610894554093-clase1.zip.
- ✓ Corroborar que la imagen se guarde correctamente. Guardar la ruta del archivo guardado en un campo “thumbnail”.

Primera entrega de tu Proyecto final

Se desarrollará un servidor que contenga los endpoints y servicios necesarios para gestionar los productos y carritos de compra en el e-commerce

CODERHOUSE

Primera entrega

Se debe entregar

Desarrollar el servidor basado en Node.JS y express, que escuche en el puerto 8080 y disponga de dos grupos de rutas: /products y /carts. Dichos endpoints estarán implementados con el router de express, con las siguientes especificaciones:

Para el manejo de productos, el cual tendrá su router en /api/products/ , configurar las siguientes rutas:

- ✓ La ruta raíz GET / deberá listar todos los productos de la base. (Incluyendo la limitación ?limit del desafío anterior
- ✓ La ruta GET /:pid deberá traer sólo el producto con el id proporcionado

Primera entrega

✓ La ruta raíz POST / deberá agregar un nuevo producto con los campos:

- id: Number/String (A tu elección, el id NO se manda desde body, se autogenera como lo hemos visto desde los primeros entregables, asegurando que NUNCA se repetirán los ids en el archivo.
- title:String,
- description:String
- code:String
- price:Number
- status:Boolean
- stock:Number
- category:String
- thumbnails:Array de Strings que contenga las rutas donde están almacenadas las imágenes referentes a dicho producto

- Status es true por defecto.
- Todos los campos son obligatorios, a excepción de thumbnails

Primera entrega

- ✓ La ruta PUT /:pid deberá tomar un producto y actualizarlo por los campos enviados desde body. NUNCA se debe actualizar o eliminar el id al momento de hacer dicha actualización.
- ✓ La ruta DELETE /:pid deberá eliminar el producto con el pid indicado.

Para el carrito, el cual tendrá su router en /api/carts/, configurar dos rutas:

- ✓ La ruta raíz POST / deberá crear un nuevo carrito con la siguiente estructura:
  - Id:Number/String (A tu elección, de igual manera como con los productos, debes asegurar que nunca se dupliquen los ids y que este se autogenere).
  - products: Array que contendrá objetos que representen cada producto

Primera entrega

- ✓ La ruta GET /:cid deberá listar los productos que pertenezcan al carrito con el parámetro cid proporcionados.
- ✓ La ruta POST /:cid/product/:pid deberá agregar el producto al arreglo “products” del carrito seleccionado, agregándose como un objeto bajo el siguiente formato:
  - product: SÓLO DEBE CONTENER EL ID DEL PRODUCTO (Es crucial que no agregues el producto completo)
  - quantity: debe contener el número de ejemplares de dicho producto. El producto, de momento, **se agregará de uno en uno.**

Además, si un **producto ya existente** intenta agregarse al producto, **incrementar el campo quantity** de dicho producto.

Primera entrega

La persistencia de la información se implementará utilizando el file system, donde los archivos “productos.json” y “carrito.json”, respaldan la información.
No es necesario realizar ninguna implementación visual, todo el flujo se puede realizar por Postman o por el cliente de tu preferencia.

Formato

- ✓ Link al repositorio de Github con el proyecto completo, sin la carpeta de Node_modules.

Sugerencias

- ✓ No olvides **app.use(express.json())**
- ✓ No es necesario implementar multer
- ✓ Link al video donde se explica.

¿Preguntas?

Opina y valora esta clase

Muchas gracias.