

Agencia de
Aprendizaje
a lo largo
de la vida

Desarrollo Fullstack



Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 28

Node JS

- ▶ Express Router
- ▶ Body Parser
- ▶ Method Override
- ▶ Middlewares

Clase 29

Node JS

- ▶ Error 404
- ▶ Controladores
- ▶ ENV

Clase 30

Base de Datos

- ▶ Introducción
- ▶ MySQL + Motor BBDD
- ▶ Tipos de Datos
- ▶ Diagramas ER

NODE JS

Controllers



**Antes de comenzar con el tema de hoy,
veamos cómo crear un middleware que
maneje las rutas que no existen de
nuestra app.**

Error 404

Para manejar este error vamos a crear un **middleware** en nuestro archivo `app.js`

```
(COLOCAR LUEGO DE LAS RUTAS Y ANTES DEL .listen())

// Middleware para manejar el error 404
app.use((req, res, next) => {
  res.status(404).send('Recurso no encontrado');
});
```

Este código revisa el estado de la petición y si no encuentra el recurso devuelve un **código 404**. Luego en `.send()` podemos enviar lo que deseamos, desde un **texto** hasta una **vista HTML**.

Controladores

Son una de las **capas de MVC** (modelo-vista-controlador) y ayuda a separar nuestra aplicación en capas.

Hasta ahora la respuesta a un **ENDPOINT** se encontraba dentro de la ruta:



```
router.get('/home', (req, res) => res.send("Página de Home"));
```

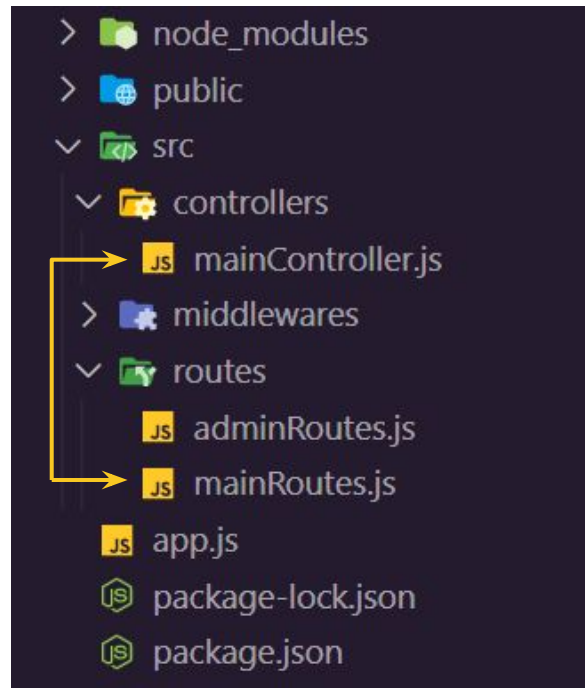
Pero, ¿qué pasa cuando tengo muchas rutas y sus respuestas son más complejas?

Controladores

Lo mejor en estos casos es separar el código para que sea más **escalable** y **legible**.

mainController.js

```
module.exports = {  
  home: (req, res) => res.send("Página de Home"),  
  contact: (req, res) => res.send("Página de Contacto"),  
  about: (req, res) => res.send("Página Sobre Nosotros")  
}
```



Controladores

Una vez que **tenemos nuestros controladores** los usamos en el archivo de rutas donde antes colocábamos la **lógica de respuesta**.

mainRoutes.js

```
const mainController = require('../controllers/mainController.js');  
/* MAIN ROUTES */  
router.get('/home', mainController.home);  
router.get('/contact', mainController.contact);  
router.get('/about', mainController.about);  
  
module.exports = router;
```

De esta manera
logramos dividir la
lógica de nuestra
aplicación.



Veámoslo en la práctica...



Datos protegidos

En una aplicación a veces necesitamos utilizar **valores** o **constantes** que **NO** se expongan en nuestro código.

Para ello se utiliza algo conocido como **.ENV**.



.env


Comencemos instalando la dependencia **dotenv**:

```
npm install dotenv
```

Luego **creamos un archivo** en la raíz del proyecto, llamado **.env** y escribimos un valor que deseamos que sea “secreto” como el puerto a utilizar en la APP.

.env

```
PORT=3000
```



```
▼ NODE_SERVER_EXPRESS  
  > node_modules  
  > public  
  > src  
  .env  
  app.js  
  package-lock.json  
  package.json
```

.env

Ahora podemos utilizar esa variable en nuestro archivo **app.js**

```
/* Requerimos la dependencia*/  
require('dotenv').config();  
/* Leemos la constante*/  
const PORT = process.env.PORT;
```

De esta manera veremos que seguimos utilizando el mismo **puerto** pero ahora lo **lee desde un archivo oculto**.

Servidor corriendo en http://localhost:3000

Este enfoque es muy útil por ejemplo para guardar las credenciales de acceso a una base de datos que no deben ser expuestas.

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