
ICARO
Nuestro compromiso es con el futuro.

Clase 2

¿Qué veremos hoy?

Hoy vamos a continuar con el módulo de **Back-End**, donde pondremos en práctica lo visto sobre **Nodejs**, **JavaScript** y **Express** para crear un servidor y poder programar nuestro **Back-End** !!.

En este módulo crearemos nuestro primer **servidor** que podrá responder a distintas peticiones de clientes y ofrecer una respuesta adecuada procesando los datos necesarios.

Además vamos a ver un poco más en detalle sobre vistas; veremos a su vez una introducción al motor **EJS** y como usarlo para seguir mejorando nuestra velocidad a la hora de crear vistas y templates, y algo super importante, como reutilizar y pasar datos a ellos.

¿Cuál será el resultado final de nuestro ejercicio?

Como parte de este trabajo práctico, tendremos tres rutas a definir:

1. Ruta principal.
2. Ruta `productos`. Se mostrará una lista con distintos productos en el navegador
3. Ruta `productos/1`. Se mostrará una lista de un solo elemento!. En este punto a ***tener en cuenta***, seran valor pre definidos (definidos en la **vista html**).

Se espera obtener

Al iniciar nuestro servidor, el mismo lo hará en el puerto 3000. Si vamos a nuestro navegador e ingresamos a la url `http://localhost:3000` obtendremos :

Se espera obtener

Se puede definir **rutas** adicionales? Claro que si!, para probarlo, definamos una **ruta**, que nos muestre una lista de productos. Para esto, similar a lo anterior, si ingresamos la url `http://localhost:3000/productos` obtendremos:

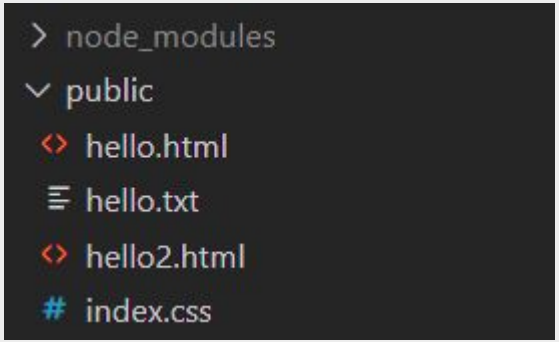
Se espera obtener

Ahora, como último paso, definamos una **ruta** con un contenido diferente. Esta ultima, tendrá al final un número indicando un ítem particular. No nos preocupemos por ese ítem por ahora, ya veremos como hacerlo de forma dinámica en breve, por ahora la **ruta** será: `http://localhost:3000/productos/1`.

Archivos estáticos y carpeta public

Que consideramos como archivos estáticos? Estos pueden ser logos, archivos de estilo (CSS), o bien archivos de extension .js (JavaScript), entre otros. Estos se agregan a plantillas HTML, para dar así más funcionalidad a nuestra vista. Pero, donde se pueden almacenar y cómo pueden ser ubicados?, Estos archivos pueden almacenarse en una carpeta denominada **public**.

I



Archivos estáticos y carpeta public

Ahora, ya tenemos definida nuestra carpeta y nuestros archivos. Consideremos el siguiente escenario; supongamos que solo definimos una ruta y queremos posteriormente acceder al archivo **hello.html** usando la ruta http://localhost:3000/public/hello.html, que pasaria?

```
const express = require('express')
const app = express()

app.get('/', (req, res) => {
    res.send("Hola mundo")
})


app.listen(3000, () => console.log('server port 3000'))
```

Archivos estáticos y carpeta public

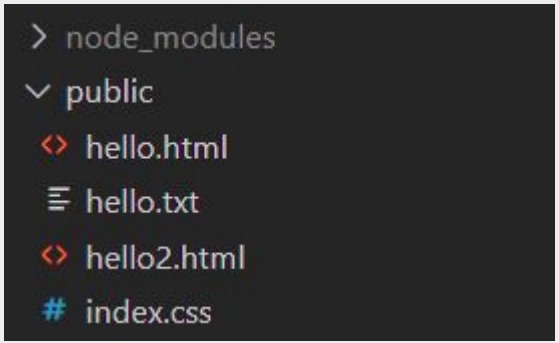
Como respuesta a lo anterior, como podemos observar, solo hemos definido una ruta, la ruta principal; pero como le decimos a nuestro servidor, que queremos que al ingresar http://localhost:3000/hello.html busque directamente a nuestro archivo hello.html, dentro del directorio **public**?

Para ello express proporciona una función llamada **static**, esto permitirá indicarle a express cuál será el directorio de archivos estáticos de nuestra aplicación:

```
express.static(root, [options])
```

Definiendo directorio ‘public’

Si hacemos uso del método `express.static(root, [options])` entonces nuestro codigo quedaria de la siguiente manera:




```
const express = require('express')
const path = require('path');
const app = express()

app.get('/', (req, res) => {
    res.send("Hola mundo")
})

// aca hacemos uso del metodo static
app.use(express.static(path.join(__dirname, 'public')))


app.listen(3000, () => console.log('server port 3000'))
```

En este caso accedemos ingresando al navegador http://localhost:3000/hello.html

Definiendo directorio “public”

Quiere esto decir que no se puede modificar e indicar una url distinta? No, es posible, por ejemplo

```
const express = require('express')
const path = require('path');
const app = express()

app.get('/', (req, res) => {
    res.send("Hola mundo")
})

// aca hacemos uso del metodo static. Observar que modificamos y usamos una url distinta (miurl)
app.use('/miurl', express.static(__dirname + '/public'));


app.listen(3000, () => console.log('server port 3000'))
```

Asi, nuestra url cambiaria a http://localhost:3000/miurl/hello.html, obteniendo el mismo resultado.

Motor EJS

EJS es un lenguaje de plantillas muy simple que puede ayudarnos a generar rápidamente páginas **HTML** utilizando código **JavaScript** ordinario. De esta forma, podemos incluir logica, entre otras ventajas. Para poder usarlo deberemos crear archivos de extensión ejs (.ejs). Veamos a continuación como instalarlo y comenzar a utilizarlo!.

EJS. Instalación y uso

Para poder hacer uso de **EJS** haremos uso de npm. Recordemos que este era nuestro gestor de dependencias y para poder llevar a cabo la instalación de ejs, ingresamos en consola:

```
$ npm install ejs
```

Una vez instalado el paquete, podremos incluirlo de la siguiente manera:

```
const express = require('express')
const app = express()

app.set('view engine', 'ejs')
```

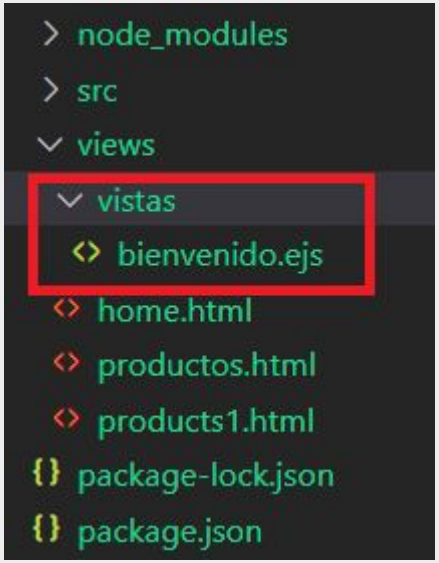
Poniendo en práctica lo visto

Ahora que hemos visto cómo iniciar un **servidor** con **Nodejs** y **express** es momento de ponerlo en práctica y despejar todas las dudas sobre estas cuestiones.

En la clase que viene, serán ustedes quienes creen sus propios **servidores** y comiencen a disponibilizar **rutas**, **archivos**, disparar código de **Back-End**, etc.

¡Muy pronto comenzaremos a interactuar con los **datos** y procesarlos, escuchando **requests / peticiones** y preparando las **responses / respuestas** correspondientemente!

Definiendo nuestra ruta usando EJS




```
const express = require('express')
const app = express()

app.set('view engine', 'ejs')

app.get('/', (req, res) => {
    res.render('vistas/bienvenido')
})


app.listen(3000, () => console.log('server port ', 3000))
```

Funcion ‘render’

Las vistas **EJS** básicamente consisten en código HTML mezclado con tags **EJS**. Las plantillas **EJS** residiran en la carpeta **app/views** y tendran la extension .ejs. Cuando utilicemos el método **res.render()** el engine **EJS** buscará la plantilla en la carpeta views, y si la encuentra renderizará la salida HTML. Pero esto no finaliza sólo en este punto, **que pasa si queremos pasar información a nuestras vistas**?

Pasando información a nuestros EJS!

```
const express = require('express')
const app = express()

app.set('view engine', 'ejs')

// Declaramos un objeto, que tendra nombre y apellido
const usuario = {
    nombre: 'nombre',
    apellido: 'apellido',
}

// En este punto, ademas de especificar la vista, pasamos el objeto a nuestra vista
app.get('/', (req, res) => {
    res.render('vistas/bienvenido', {
        usuario: usuario
    })
})


app.listen(3000, () => console.log('server port ', 3000))
```

Pasando información a nuestros EJS!

Como podemos observar, el método **render**, esta en este caso recibiendo un parámetro extra, que representa a la información que queremos finalmente mostrar en nuestra vista. El último paso, es una vez enviada la información, es mostrarla en nuestro **EJS**, como se muestra a continuación:

```
<h1>Hola, soy <%= usuario.nombre  %></h1>
```

Recordemos que ‘usuario’ es el nombre del objeto que se pasó como parámetro a la función **render**.

Reusando vistas con EJS. Introducción a ‘Parciales’

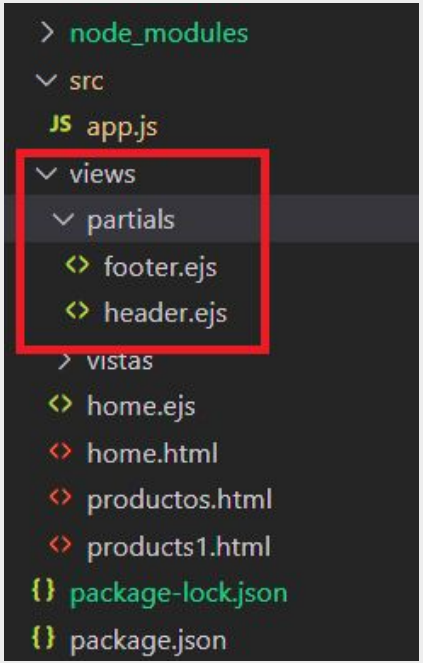
Igual que muchas aplicaciones que creamos, hay mucho código que se reutiliza. A esos códigos los llamamos parciales y en nuestro caso definiremos dos archivos que usaremos en todo nuestro sitio: **header.ejs**, y **footer.ejs**.

Sintaxis para incluir un parcial de EJS

Debemos usar <%- **include**('**RELATIVE/PATH/TO/FILE**') %> para integrar un parcial de EJS en otro archivo.

El resultado final se muestra a continuacion!.

Reusando vistas con EJS. Introducción a ‘Parciales’




```
const express = require('express')
const app = express()

app.set('view engine', 'ejs')

// Accedemos al empezar la aplicacion, al home.ejs
app.get('/', (req, res) => {
    res.render('home')
})


app.listen(3000, () => console.log('server port ', 3000))
```

Poniendo en práctica lo visto

¡Muy pronto comenzaremos a interactuar con nuestras vistas **EJS**, pasar datos a las mismas y procesarlos. También empezaremos a reutlizar nuestras vistas!. Poco a poco iremos haciendo nuestras vistas más complejas.

Muchas gracias!

ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar