
ICARO
Nuestro compromiso es con el futuro.

Back End

Clase 1

¿Qué veremos hoy?

Hoy vamos a comenzar el módulo de **Back-End**, donde veremos en profundidad **Nodejs** y utilizaremos lo aprendido de **JavaScript** integrándolo con **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.

Nodejs

Como recordaremos de las primeras clases, **Nodejs** era el framework de **JavaScript** que nos permitía sacar a **JavaScript** de un navegador y poder correrlo desde nuestra computadora. Ahora utilizaremos este potente framework para primeramente crear un **servidor** y poder responder a rutas básicas sirviendo distintos archivos **HTML**.

Express

Para crear nuestro servidor utilizaremos Express, que es un framework de Nodejs que nos simplificará la tarea y hará nuestro código más sencillo, legible y mantenible.

Express es uno de los frameworks para creación de servidores con más popularidad y ciertamente es el más popular de Nodejs.

A continuación veremos el proceso de instalación y los primeros pasos que daremos para crear nuestro servidor.

Creando nuestra aplicación con Express

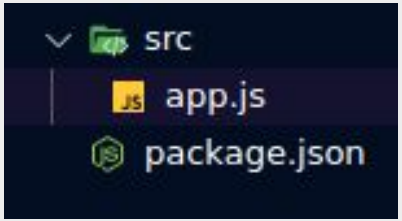
Lo primero será crear una carpeta, donde estará ubicado nuestro proyecto.

Una vez la tengamos creada, utilizando la terminal ingresaremos en la misma y correremos el siguiente comando.

Como hemos visto en el módulo de **Introducción a la Programación**, esta línea nos permitirá inicializar un proyecto de **Nodejs**, para luego crear el entry-point de nuestra aplicación, definir nuestro `package.json` y comenzar a crear las estructuras necesarias para nuestro **servidor**.

Configurando las estructuras iniciales

Como buena práctica, colocaremos nuestro archivo app.js (nuestro entry point) dentro de una carpeta `src`. El nombre **src** significa **source**, y será literalmente la fuente principal de nuestro código, ahí será donde almacenaremos el grueso de nuestro proyecto y desde donde se disparará la magia hacia quien requiera esas funcionalidades.



Podemos aprovechar este paso para instalar un módulo esencial que utilizaremos en breve: **express**.

```
> npm i express
```

Veremos que poco a poco iremos complementando esta carpeta y agregando más y más partes conforme vayamos aprendiendo nuevos conceptos y modificando nuestros proyectos.

Inicializando Git en nuestro proyecto

Estos estadíos iniciales serían un buen momento para inicializar **Git** en nuestro repositorio. De esta manera podremos ir haciendo commits periódicamente e ir guardando de a poco los cambios que vayamos haciendo en nuestro proyecto, y además iremos practicando estas herramientas al tiempo que aseguramos las versiones de nuestro proyecto. Para inicializar **Git** en nuestro proyecto, dispararemos el comando:

```
❯ git init
```

Una vez creado un remoto en algún sitio a tal fin (por ejemplo **GitHub**) podremos agregarlo a nuestro proyecto y ya ir haciendo los primeros commits.

```
❯ git remote add origin git@github.com:userxxxx/repoxxxx.git
```

Inicializando Git en nuestro proyecto

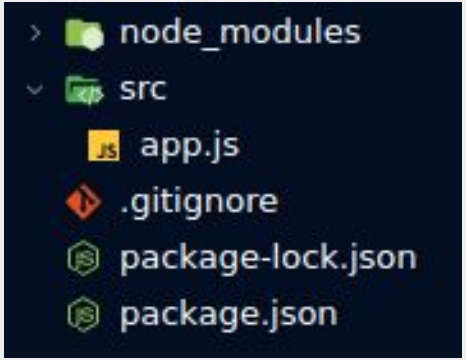
No olvidemos agregar un archivo `.gitignore`, que servirá para indicarle a Git qué archivos o directorios **NO** queremos que trackee y por lo tanto no vaya guardando en las distintas versiones. Uno de los directorios ideales para agregar en nuestro .gitignore sería el ***node_modules*** ya que no sólo es un directorio que sabemos se volverá muy grande en tamaño sino que no es indispensable, ya que con un simple comando podremos instalar estos módulos teniendo bien configurado nuestro package.json. Ya con esto configurado podemos hacer un primer commit para comenzar a versionar el proyecto:

```
> git add .
> git commit -m 'first commit'
```

```
> git push origin master
```

Estructura general de nuestra aplicación

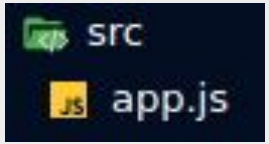
Hasta ahora, deberíamos tener un proyecto con una estructura parecida a la siguiente:



Primeros pasos de nuestro servidor

Ahora que tenemos la estructura básica, podemos comenzar a concentrarnos en el archivo **app.js** que será el corazón de nuestra aplicación.

Este archivo es donde colocaremos, al menos al comienzo, la mayor parte de código y será aquí donde se realizarán casi todas las acciones.



Primeros pasos de nuestro servidor

Ya que tenemos instalado **express**, el primer paso será importarlo utilizando el sistema de módulos de **Nodejs** que ya conocemos.
Luego, inicializaremos nuestra aplicación de **express** ejecutando como método la variable donde guardamos la importación de **express**.
A su vez, esta aplicación que estamos ejecutando la guardaremos en otra variable, **app**, para que sea más fácil acceder a ella, ya que es desde esa variable desde donde se disparará el código específico de nuestra app y la utilizaremos muy a menudo en este archivo.

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

Primeros pasos de nuestro servidor

Ahora lo siguiente será darle vida a nuestra aplicación, para lo cual le diremos a nuestro servidor que deberá estar a la escucha de distintas peticiones en una dirección particular. Como todo esto lo estamos ejecutando en nuestro entorno local, utilizaremos algún puerto de nuestra computadora, por ejemplo, el puerto 3000.

Una buena práctica es guardar el puerto en una constante, para poder reutilizar la variable sin repetir el dato, y además si necesitamos cambiarlo, lo podemos hacer desde un solo lugar.

```
const PORT = 3000
```

Primeros pasos de nuestro servidor

Teniendo listo nuestro puerto, ahora utilizaremos el método listen de nuestra **app**. Este método recibe un primer parámetro que representa al **puerto** a escuchar y un segundo parámetro opcional que es un **callback** que se ejecutará cuando nuestro servidor se levante y esté escuchando en el puerto que le especificamos. Por este mismo motivo lo más común es hacer un `console.log( )` que nos dé justamente esa información.

```
app.listen(PORT, () => console.log('listening on port ', PORT))
```

Disparando nuestro servidor

Ahora sí, ya podemos levantar nuestro servidor para ver si está funcionando correctamente.

Si hicimos todo bien, al disparar desde la consola el comando:

```
❯ node src/app
```

Deberíamos obtener un resultado como el siguiente:

```
listening on port 3000
```

¡Éxito! Ya tenemos nuestro servidor corriendo y escuchando, pero... ¿Y ahora?

Nuestras primeras rutas

Si bien nuestro servidor ya está levantado y escuchando **peticiones**, aún no tenemos ningún lugar desde donde hacerle estas **peticiones** y que éste nos envíe las **respuestas**.

Aún nos queda un paso más para tener lo mínimo e indispensable para considerarlo un servidor funcional: crear nuestra primer **ruta**.

Para ello, invocaremos desde nuestro objeto app su método **GET**, que pertenece al conjunto de métodos **HTTP**, que veremos en detalle más adelante.

Este método **GET** nos permite recibir las **peticiones** de un navegador, por ejemplo, siempre y cuando configuremos correctamente nuestro código.

Nuestras primeras rutas

Dicho esto, entonces, lo que haremos será llamar al método **get** y pasarle dos parámetros:

1. La **ruta** a la cual deberá ingresarse para que se procese la **petición** y se envíe la **respuesta**.

1. Un **callback** que se ejecutará cuando se ingrese a la **ruta** previamente mencionada. Este **callback** recibirá como parámetros dos objetos: un **req** y un **res**, el primero representa al **request**, es decir, a la **petición** que llega del navegador, con todos sus atributos, propiedades y métodos; y el segundo representa al **response**, es decir, a la **respuesta** que podemos darle desde el **servidor**, también con todos sus atributos, métodos y propiedades.
   Dentro del cuerpo del código de este **callback**, irá el código que deberá ejecutarse al acceder a la **ruta**. Lo más básico y sencillo en este paso será utilizar un método que nos provee el parámetro **res** llamado *send*. Este método simplemente responderá a quien realice la **petición** con lo que le coloquemos como parámetro.

Nuestras primeras rutas

Si seguimos bien todo lo mencionado, nuestro código debería verse algo así:

```
app.get('/home', (req, res) => {
  res.send('¡Hola! ¡Bienvenido a mi aplicación!')
})
```

Hasta aquí todo bien, pero… hay un problema. Si intentamos acceder a nuestro puerto local 3000, que sería el que configuramos anteriormente, en la **ruta** especificada (quedaría `localhost:3000/home`, veremos que nuestro código no está funcionando como esperábamos. ¿A qué podrá deberse?

Reiniciando nuestro servidor

Si nuestra respuesta fue que se debe a que hay algún error en el código, lamentablemente no es eso exactamente… El problema es que cuando levantamos nuestro **servidor** el código en ese momento era distinto al que tenemos ahora, por ende, el **servidor** está corriendo con *"código viejo"*, por así decirlo. Debemos entonces cortar la ejecución del mismo con `Ctrl+C` y volver a lanzar el **servidor** con `node src/app`.

Ahora sí, si ingresamos en algún navegador a `localhost:3000/home` deberíamos ver el mensaje que programamos en el **app.js** como **respuesta** a la **petición** a esa ruta.

En la ruta completa `localhost` representa a nuestros puertos locales, con el `:3000` le indicamos específicamente a cuál puerto queremos acceder y finalmente le pasamos la ruta que estamos buscando que sería `/home`, la que creamos en nuestro **app.js**.

Reiniciando nuestro servidor

Entonces, parece que vamos a tener que estar reiniciando el servidor todo el tiempo para aplicar los cambios nuevos que vayamos haciendo en nuestro código. Esto parece bastante engorroso, ¿Podríamos automatizarlo?

La respuesta es un rotundo ¡**Sí**! Para ello utilizaremos un módulo de npm llamado ***nodemon***, que nos permitirá reiniciar el servidor automáticamente cuando detecte que se han producido cambios en el código.

Obviamente necesitamos instalarlo primero, podemos hacerlo de manera local, es decir, en el proyecto que estamos trabajando, o bien de manera global en nuestra computadora para que quede disponible en todo momento y en todo proyecto.

Reiniciando nuestro servidor

Para instalarlo de manera local:

```
npm i -D nodemon
```

Para instalarlo de manera global:

```
npm i -g nodemon
```

Sea como sea que hayamos elegido instalarlo, lo que deberemos hacer es lanzar nuestra aplicación utilizando ahora el comando `nodemon src/app` en vez de `node src/app`:

```
› nodemon src/app
[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/app.js`
listening on port  3000
```

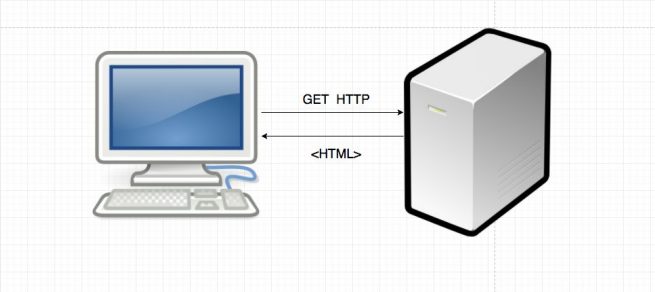
Devolviendo más que un string

Ahora que podemos ir modificando el código y que nuestro **servidor** vaya refrescándose dinámicamente, nos será mucho más fácil ir viendo los cambios que vayamos realizando.

Uno de los primeros cambios que implementaremos será ver qué otras posibilidades tenemos a la hora de responder una **petición**.

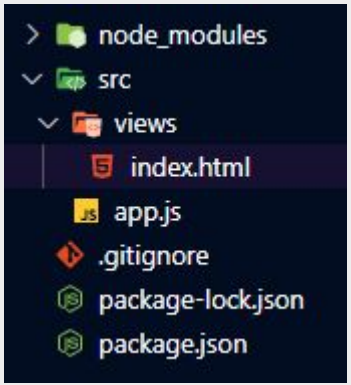
Lo más básico sería que en lugar de responder con un mensaje predefinido, querramos enviar un archivo **HTML**, lo que aquí llamaremos una vista.

Para poder retornar una **vista** necesitaremos actualizar un poco nuestro código.



Retornando una vista

El primer paso será crear una carpeta llamada **views** dentro de nuestra carpeta **src**. Aquí dentro crearemos todos nuestros **HTML**, por ahora simplemente le daremos vida a un `index.html` que será la vista a renderizar.



Luego será fundamental cambiar el método que dispara nuestro parámetro **res** en la **ruta** deseada. Ya no utilizaremos el `res.send( )`, sino que ahora haremos uso del método **sendFile**, quedando entonces `res.sendFile( )`.

```
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'views/index.html'))
})
```

Retornando una vista

Con lo que acabamos de escribir podría surgirnos la pregunta ¿Qué es eso que vemos pasado por parámetro en el **sendFile( )**?

```
sendFile(path.join(__dirname, 'views/index.html'))
```

La respuesta surge de que lo que debe recibir por parámetro es la **ruta** absoluta de donde se encuentre nuestro archivo que deseamos enviar como **respuesta**.

Para ello, utilizaremos el objeto **path** que posee varios métodos para ayudarnos a trabajar con **rutas** (no olvidemos requerirlo previamente y guardarlo en una constante homónima.
Utilizando el método `join( )` del path, podemos indicarle cuáles son las partes de la **rutas** que deseamos unir. La primera parte, el `__dirname`, no es más que una variable de entorno que cuando sea ejecutada reemplazará su valor por la **ruta** absoluta del archivo donde está siendo llamada; luego, pasamos cómo segundo parámetro el final de la **ruta** donde se podrá encontrar nuestro archivo `index.html`.

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!

¡Vamos al código!

Muchas gracias!

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