**Hola Mundo de Express**

Ahora que sabemos un poco más sobre que es Express, introduciremos un pequeño ejemplo de una aplicación muy básica en Express.

*//(1) Requerimos o importamos 'express' a nuestra aplicación.*

**const** express = require('express');

*//(2) Instanciamos 'express' en una variable llamada 'app' por convención.*

**const** app = express();

*//(3) Le indicamos a 'express' un verbo HTTP 'get' y una ruta.*

app.get('/', (req, res) **=>** {

res.send('¡Hola mundo!');

});

*//(4) Le indicamos a 'express' que escuche posibles peticiones en el puerto 3000 con el host 127.0.0.1 .*

**const** port = 3000;

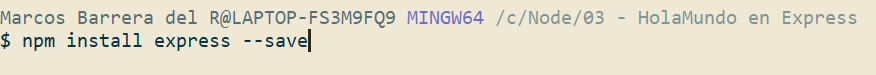
**const** host = '127.0.0.1';

app.listen(port, host, () **=>** {

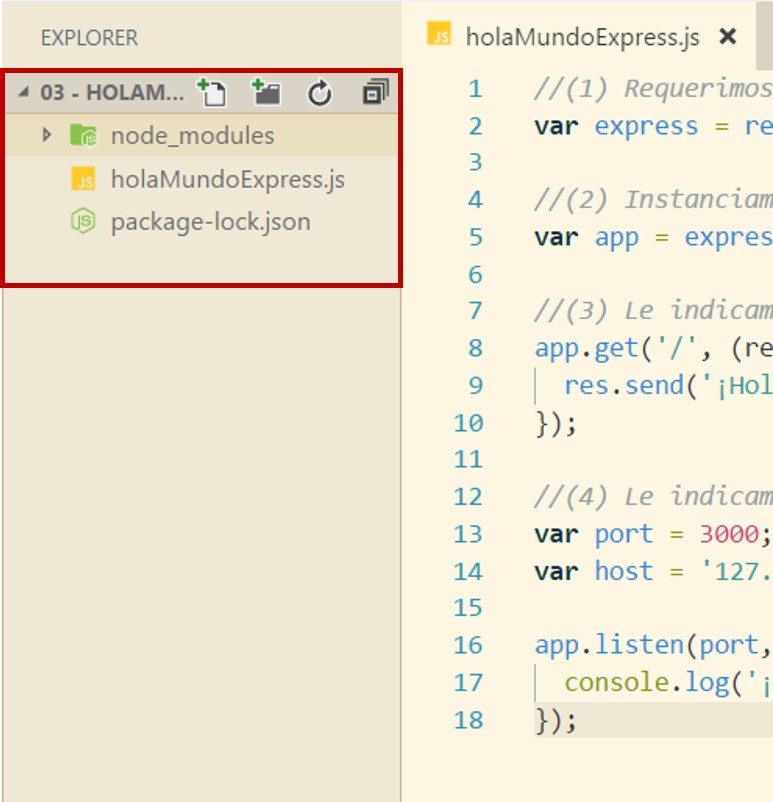
console.log('¡Escuchando en puerto 3000!');

});

Como podemos ver, en el punto (1), requerimos el módulo de *express* para poder usarlo en nuestra aplicación. Como todavía no lo hemos instalado, es necesario correr el comando de “*npm install express --save*” desde la carpeta en la que se encuentre nuestro archivo JavaScript, para así poder utilizarlo correctamente.



Una vez ejecutado el comando de *npm install*, se nos creará nuestra carpeta *node\_modules* y un archivo *package.json (*componentes que ya explicamos en la sección de NPM*).* Y nuestro directorio quedaría tal que así:



Ahora, continuando con la explicación del código del punto (2):

**const** app = express();

Podemos ver cómo se utiliza una variable llamada “*app*” para instanciar *express* en ella y aunque el nombre puede ser cambiado, se recomienda por convención siempre llamarla así.

En el punto (3):

app.get('/', (req, res) **=>** {

res.send('¡Hola mundo!');

});

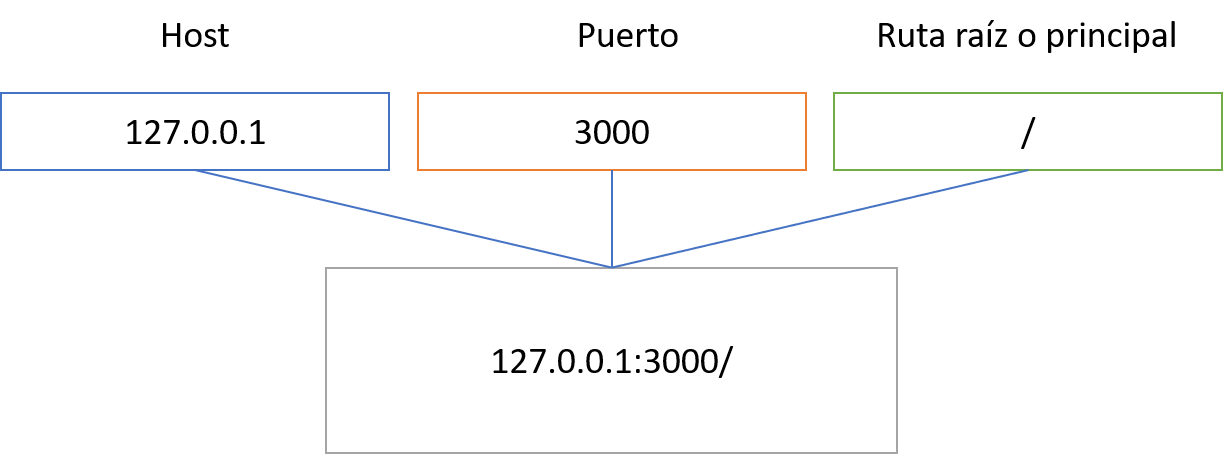
Observamos cómo la variable “*app*” instanciada anteriormente contiene un método llamado *“get”.* Si recordamos los distintos verbos del protocolo HTTP, entre ellos, había uno llamado GET. Esta función de “*app*” nos permite indicarle a *express* que cuando un cliente envié una petición GET al servidor y *express* la reciba, dicha petición tendrá que ir encaminada a *“app.get(…)”.* Este patrón de diseño es muy poderoso y nos permite modularizar nuestro código basándonos en el tipo de petición (GET, POST, PUT, etc.) que se reciba en *express.*

Al igual que existe “*app.get(…)*”, existe una función en “*app*” para cada uno de los verbos HTTP. Es decir, también existe *app.post(…), app.put(…), app.delete(…),* etcétera.

Si vemos en el punto (3), el primero parámetro de “*app.get(…)”* es una *string* con valor de ‘/’. Esto le indica a *express* que cada vez que una petición llegue al servidor con una **ruta** de ‘/’, la encamine a esta porción de código.

Pude ser un poco confuso qué es una **ruta**, pero veámoslo de la siguiente manera:

Supongamos que tenemos nuestro servidor de *Express* montado de manera local en nuestra computadora, eso quiere decir que nuestro *host* será “*127.0.0.1*” o lo que es lo mismo, “*localhost*”, ahora imaginemos que *Express* está escuchando en el puerto *3000.* Esto quiere decir que todas las peticiones que lleguen a nuestro servidor usarán la siguiente estructura:



Si analizamos la estructura anterior, podemos ver que para que una petición pueda llegar a nuestro servidor, tiene que especificar un *Host* (en nuestro caso *127.0.0.1* porque es local), un Puerto (en nuestro caso, el puerto *3000*)y la Ruta.

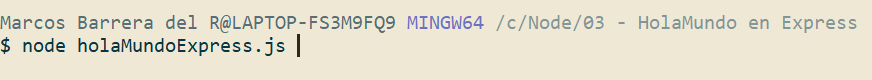
Si regresamos a nuestro ejemplo:

app.get('/', (req, res) **=>** {

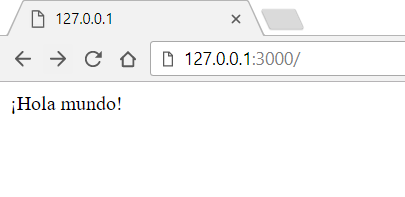
res.send('¡Hola mundo!');

})

Podemos ver que le estamos indicando a *Express* que a todas las peticiones que lleguen a la Ruta Principal con el verbo HTTP de tipo GET, les responda con una *string* ‘¡Hola mundo!’. Ahora, si corremos el archivo con el siguiente comando:



Después nos dirigimos a nuestro navegador, e ingresamos [127.0.0.1:3000/](http://127.0.0.1:3000/), veremos lo siguiente:



Al ingresar el host, el puerto y dar Enter, hicimos que nuestro navegador hiciera una petición HTTP de tipo GET a nuestro servidor de *Express* y como en nuestro archivo holaMundoExpress.js, le especificamos a *Express* que, si recibía una petición de tipo GET en la ruta principal, entonces tendría que responder con una *string* de ‘¡Hola Mundo!’ al cliente (nuestro navegador).

El segundo parámetro de *“app.get(…)”* es una *callback,* concepto que ya explicamos al inicio de la sección de Node. *Express* inyectará en esta *callback* los valores del objeto *req (*la request o petición*)* y *res (*la respuesta o response*).* Lo que está marcado en color rojo, es el segundo parámetro, es decir, nuestra *callback* que será llamada asíncronamente por *Node y Express* posteriormente, cuando una petición (*request*)llegue al servidor y concuerde con el verbo HTTP y la ruta de ‘*/’*.

app.get('/', (req, res) **=>** {

res.send('¡Hola mundo!');

})

Si no recuerdas cómo funciona Node y su naturaleza asíncrona, puedes volver a leer las secciones de “***Hola Mundo en Node***”, “***Programación Asíncrona***”y“***Programación Asíncrona en Node.js***” para refrescar un poco la memoria.

Una vez que llegue una petición y concuerde el verbo HTTP y la ruta, *Express* usará la función “*send*”del objeto *res (*es decir, el objeto *response,* que en español es petición*)* para regresarle el valor ‘¡Hola Mundo!’ al cliente.

Por último, en el punto (4)

**const** port = 3000;

**const** host = '127.0.0.1';

app.listen(port, host, () **=>** {

console.log('¡Escuchando en puerto 3000!');

});

Utilizamos la función de *“listen”,* que le indica a *Express* que esté esperando (o escuchando) peticiones en el puerto *“3000” (*primer parámetro*)* del host *“127.0.0.1”* (segundo parámetro) y finalmente le indicamos pasamos una *callback* sin parámetro propios, como tercer parámetro. Esta *callback* será ejecutada cuando se inicie el servidor e imprimirá la *string* ‘¡Escuchando en puerto 3000!’ en la consola.

