Introducción a Express

Express es un framework ligero y, a la vez, flexible y potente para desarrollo de aplicaciones web con Node. En primer lugar, se trata de un framework ligero porque no viene cargado de serie con toda la funcionalidad que un framework podría tener, a diferencia de otros frameworks más pesados y autocontenidos como Symfony o Ruby on Rails. Pero, además, se trata de un framework flexible y potente porque permite añadirle, a través de módulos Node y de *middleware*, toda la funcionalidad que se requiera para cada tipo de aplicación. De este modo, podemos utilizarlo en su versión más ligera para aplicaciones web sencillas, y dotarle de más elementos para desarrollar aplicaciones muy complejas.

Como veremos, con Express podemos desarrollar tanto servidores típicos de contenido estático (HTML, CSS y Javascript), como servicios REST accesibles desde un cliente, y por supuesto, aplicaciones que combinen ambas cosas.

Podéis encontrar información actualizada sobre Express, tutoriales y demás información en su página oficial.

1. Descarga e instalación

La instalación de Express es tan sencilla como la de cualquier otro módulo que queramos incorporar a un proyecto Node. Simplemente necesitamos ejecutar el correspondiente comando `npm install` en la carpeta del proyecto donde queramos añadirlo (y, previamente, el comando `npm init` en el caso de que aún no hayamos creado el archivo *package.json*):

```
npm install express
```

1.1. Ejemplo de servidor básico con Express

Vamos a crear un proyecto llamado "*PruebaExpress*" en la carpeta de "*ProyectosNode/Pruebas*", y a instalar Express en él. Después, creamos un archivo llamado `index.js` con este código:

```
const express = require('express');
let app = express();
app.listen(8080);
```

El código, como podemos ver, es muy sencillo. Primero incluimos la librería, después inicializamos una instancia de Express (normalmente se almacena en una variable llamada `app`), y finalmente, la ponemos a escuchar por el puerto que queramos. En este caso, se ha escogido el puerto 8080 para no interferir con el puerto por defecto por el que se escuchan las peticiones HTTP, que es el 80. También es habitual encontrar

ejemplos de código en Internet que usan los puertos 3000 o 6000 para las pruebas. Es indiferente el número de puerto, siempre que no interfiera con otro servicio que tengamos en marcha en el sistema.

Para probar el ejemplo desde nuestra máquina local, basta con poner en marcha la aplicación Node, abrir un navegador y acceder a la URL:

http://localhost:8080

Si pruebas a ejecutar la aplicación Node desde Visual Studio Code, verás que no finaliza. Hemos creado un pequeño servidor Express que queda a la espera de peticiones de los clientes. Sin embargo, aún no está preparado para responder a ninguna de ellas, por lo que dará un mensaje de error al intentar acceder a cualquier URL. Esto lo solucionaremos en los siguientes apartados.

2. Express como proveedor de servicios

Ahora que ya sabemos qué es Express y cómo incluirlo en las aplicaciones Node, veremos uno de los principales usos que se le da: el de servidor que proporciona servicios REST a los clientes que lo soliciten. Para ello, y como paso previo, debemos comprender y asimilar cómo se procesan las rutas en Express, y cómo se aísla el tratamiento de cada una, de forma que el código resulta muy modular e independiente entre rutas.

2.1. Un primer servicio básico

Partiendo del ejemplo anterior, vamos a añadir una serie de rutas en nuestro servidor principal para dar soporte a los servicios asociados a las mismas. Una vez hemos inicializado la aplicación (variable `app`), basta con ir añadiendo métodos (`get`, `post`, `put` o `delete`), indicando para cada uno la ruta o URI que debe atender, y el callback o función que se ejecutará en ese caso. Por ejemplo, para atender a una ruta llamada `/bienvenida` por GET, añadiríamos este método:

```
let app = express();

app.get('/bienvenida', (req, res) => {
    res.send('Hola, bienvenido/a');
});

...
```

El callback en cuestión recibe dos parámetros siempre: el objeto que contiene la petición (típicamente llamado `req`, abreviatura de *request*), y el objeto para emitir la respuesta (típicamente llamado `res`, abreviatura de *response*). Más adelante veremos qué otras cosas podemos hacer con estos objetos, pero de momento emplearemos la respuesta para enviar (`send`) texto al cliente que solicitó el servicio, y `req` para obtener determinados datos de la petición. Podemos volver a lanzar el servidor Express, y probar este nuevo servicio accediendo a la URL correspondiente:

http://localhost:8080/bienvenida

Del mismo modo, se añadirán el resto de métodos para atender las distintas opciones de la aplicación. Por ejemplo:

```
app.delete('/comentarios', (req, res) => { ...
```

> **Ejercicio 1:**
>
> Crea una carpeta llamada "**ExpressBasico**" en la carpeta de ejercicios "*ProyectosNode/Ejercicios*". Instala Express en ella, y define un servidor básico que responda por GET a estas dos URIs:
>
> - URI `/fecha` : el servidor enviará como respuesta al cliente la fecha y hora actuales. Puedes utilizar el tipo `Date` de JavaScript sin más, o también puedes "recrearte" con la librería "moment" vista en sesiones anteriores, si quieres.
> - URI `/usuario` : el servidor enviará el login del usuario que entró al sistema. Necesitarás emplear la librería "os" del núcleo de Node, vista en sesiones anteriores, para obtener dicho usuario.

3. Elementos básicos: aplicación, petición y respuesta

Existen tres elementos básicos sobre los que se sustenta el desarrollo de aplicaciones en Express: la aplicación en sí, el objeto con la petición del cliente, y el objeto con la respuesta a enviar.

3.1. La aplicación

La aplicación es una instancia de un objeto Express, que típicamente se asocia a una variable llamada `app` en el código:

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

Toda la funcionalidad de la aplicación (métodos de respuesta a peticiones, inclusión de *middleware*, etc) se asienta sobre este elemento. Cuenta con una serie de métodos útiles, que iremos viendo en futuros ejemplos, como son:

- `use(middleware)` : para incorporar *middleware* al proyecto
- `set(propiedad, valor)` / `get(propiedad)` : para establecer y obtener determinadas propiedades relativas al proyecto
- `listen(puerto)` : para hacer que el servidor se quede escuchando por un puerto determinado.
- ...

3.2. La petición

El objeto de petición (típicamente lo encontraremos en el código como `req`) se crea cuando un cliente envía una petición a un servidor Express. Contiene varios métodos y propiedades útiles para acceder a información contenida en la petición, como:

- `params`: la colección de parámetros que se envía con la petición
- `query`: con la *query string* enviada en una petición GET (información detrás del interrogante `?` en una URL)
- `body`: con el cuerpo enviado en una petición POST
- `files`: con los archivos subidos desde un formulario en el cliente
- `get(cabecera)`: un método para obtener distintas cabeceras de la petición, a partir de su nombre
- `path`: para obtener la ruta o URI de la petición
- `url`: para obtener la URI junto con cualquier *query string* que haya a continuación
- ...

3.3. La respuesta

El objeto respuesta se crea junto con el de la petición, y se completa desde el código del servidor Express con la información que se vaya a enviar al cliente. Típicamente se representa con la variable o parámetro `res`, y cuenta, entre otros, con estos métodos y propiedades de utilidad:

- `status(codigo)`: establece el código de estado de la respuesta
- `set(cabecera, valor)`: establece cada una de las cabeceras de respuesta que se necesiten
- `redirect (estado, url)`: redirige a otra URL, con el correspondiente código de estado
- `send([estado], cuerpo)`: envía el contenido indicado, junto con el código de estado asociado (de forma opcional, si no se envía éste por separado).
- `json([estado], cuerpo)`: envía contenido JSON específicamente, junto con el código de estado asociado (opcional)
- `render(vista, [opciones])`: para mostrar una determinada vista como respuesta, pudiendo especificar opciones adicionales y un callback de respuesta.
- ...