Ejercicio 3: express 1

Hemos visto ya que podemos crear módulos independientes e intercambiables, que podemos llevarnos a otros proyectos y aprovechar las funciones que hemos creado muchas veces. No solo podemos copiar y pegar el documento .js en nueva carpeta directamente, sino que también podemos registrarnos en la web de npm y subir ahí nuestros módulos para que otros también se los puedan importar y usar las funciones que hemos creado.

Pero los demás no saben qué funciones tenemos en nuestro módulo y para qué sirven, así que cuando lo subamos a npm deberíamos documentarlo de manera que el resto de la comunidad entienda lo que hace y los interesados se lo puedan descargar.

En este ejercicio vamos a importar (requerir) un módulo hecho por otro, subido al repositorio npm, y descargable con el instalador npm. En la carpeta hay un solo fichero .js llamado "servidor": si echas un vistazo al principio del código verás que todo empieza requiriendo un módulo que se llama "express". Express es el servidor más conocido y usado con Node: en realidad viene a ser como el documento con funciones que hemos creado nosotros antes, que nos ahorran trabajo porque es código que ya está programado y podemos usar simplemente llamándolas desde cualquier otro punto de la aplicación.

Aquí no hará falta entrar a explicar Express en detalle y todo lo que necesitas saber está en este mismo tutorial, pero si quieres investigarlo más a fondo aquí tienes la página de express en npm:

https://www.npmjs.com/package/express

Por lo que nos toca a nosotros lo único que vamos a hacer es, sabiendo que nuestro código necesita express para funcionar, importárnoslo con npm. Pero antes que eso tenemos otro sencillo paso que hacer.

Imagínate que no necesitamos importar solamente express, sino varios (quizá muchos) módulos de terceros. También es posible (de hecho en la práctica de este tutorial es así) hay requerimientos de módulos que no se hacen desde el fichero .js principal, sino desde otros escondidos en subcarpetas. ¿Cómo sabemos qué módulos tenemos que traer? Aquí entra a trabajar una de las "magias" de npm.

En línea de comandos (entrando a esta carpeta) ejecuta lo siguiente:

```
npm init --yes
```

Con esto verás que se te ha creado un fichero llamado "package.json" que contiene información sobre tu aplicación: el nombre, la versión y demás. No entraremos ahora en más detalles, simplemente fíjate en que (de momento) no hay ninguna referencia a express.

Ahora vamos a instalar el módulo. Otra vez desde la consola escribe:

```
npm install express --save
```

Con esto comprobarás que npm empieza a bajarse el paquete, y al terminar verás tres cosas nuevas: primero, una carpeta llamada node_modules que antes no estaba ahí (esto no necesitas tocarlo para trabajar); segundo, un fichero package-lock.json (tampoco tienes que tocarlo); y tercero, en el archivo package.json se ha creado un apartado de “dependencies” que ahora sí hace mención a express con su correspondiente versión (esto sí que te interesa).

Esa carpeta node_modules es donde se guardan los paquetes de terceros que tienes en tu proyecto (y en package-lock va información relacionada, tú olvídate de esto), y al importarlos con la opción --save le decimos a npm que guarde un registro de los paquetes que tenemos instalados. ¿En qué nos ayuda esto?

Elimina la carpeta node_modules y el archivo package-lock.js. Con esto has eliminado completamente el módulo de express que te habías traído, así que si intentaras arrancar tu aplicación ahora recibirías un error porque en tu código se requiere express y no está.

Pero fíjate en que en package.json sigue estando la mención a express en el apartado de dependencias; solo tenemos express pero en un proyecto más grande podríamos tener una lista muy larga de módulos.

En un caso así no haría falta instalarlos todos uno por uno. En línea de comandos escribe:

```
npm install
```

Y solo con esto se te volverá a crear mágicamente la carpeta node_modules y el package-lock.json, y en la carpeta se importarán automáticamente todos los paquetes (en este caso solo express) que tengamos en la lista de dependencias de package.json.

Esto te ayuda en muchas cosas: por ejemplo si quieres compartir tu código (como he hecho yo con este paquete-tutorial) no hace falta que metas en tus ficheros compartidos la carpeta node_modules (que ocupa bastante por cierto); la eliminas y de paso también el fichero package-lock.json, compartes el código sin ellos, y el receptor simplemente ejecuta “npm install” para tener el proyecto completo. A partir de ahora, siempre que alguien (incluido yo ahora con los ejercicios siguientes) te pase un proyecto de Node, lo primero que tienes que hacer es ver si hay dependencias de módulos de terceros que instalar.

Empezando con express

Ahora sí estamos listos para entrar en lo que es el ejercicio de verdad. Ejecuta desde línea de comandos:

```
node server.js
```

Y aparecerá en la consola un mensaje avisándonos de que el servidor está escuchando por el puerto 3000. Abre cualquier navegador y vete a esta dirección:

```
localhost: 3000  // “localhost” sustituye a la dirección ip de tu propio ordenador
```

Y aparecerá el texto que devuelve el servidor. Ojo, es un texto porque para el ejemplo es lo más fácil pero podría ser otra cosa: xml, json, una página en html, una imagen… Para ver un ejemplo, si vamos a

localhost:3000/ejemplojson

Encontraremos lo que devuelve en este caso en formato json. Mientras el servidor express esté ejecutándose esas direcciones estarán disponibles en el navegador, cuando cortes la ejecución del programa desaparecerán.

En el código de server.js puedes ver cómo funciona express. Solo para que te fijes en algo importante: la llamada a una ruta lo que hace es llamar a una función concreta asociada a esa ruta a la que se llama, y al ejecutarse esa función simplemente el código que contiene se pone en marcha. Tómate un momento para entender esto, porque es una clave importante y una vez lo tengas claro todo lo demás de express se te hará muchísimo más fácil.