Uso de *middleware*

El término *middleware* ha sido mencionado algunas veces con anterioridad, pero aún no habíamos explicado de qué se trata. En términos generales, podríamos definir *middleware* como un software que se ejecuta en medio de otros dos. En el ámbito en que lo estamos tratando, un **middleware** es una función que gestiona peticiones y respuestas HTTP, de forma que puede manipularlas y pasarlas al siguiente *middleware* para que las siga procesando, o bien terminar el proceso y enviar por sí misma una respuesta al cliente.

1. El método *use*

Existen diferentes *middlewares* disponibles en los módulos de Node, tales como el procesador `express.json()` que hemos empleado en ejemplos previos para procesar el cuerpo de peticiones POST y poder acceder a su contenido de forma más cómoda. El middleware se aplica sobre una aplicación a través del método `use` de la misma. Por eso, cuando hacíamos:

```
app.use(express.json());
```

... estábamos activando el *middleware* correspondiente de Express para procesar los cuerpos de las peticiones y extraer de ellos la información JSON que contengan, dejándola preparada para ser accedida.

2. Definir nuestro propio *middleware*

Además de este y otros ejemplos de *middleware* hecho por terceras partes y que podemos incorporar a nuestros proyectos, también podemos definir nuestra propia función de *middleware*, y mediante el método `use` de la aplicación, incluirla en la cadena de procesamiento de la petición y la respuesta. Por ejemplo, el siguiente middleware saca por consola la dirección IP del cliente que hace la petición:

```
app.use((req, res, next) => {
    console.log("Petición desde", req.ip);
    next();
});
```

Como vemos, el *middleware* no es más que una función que acepta tres parámetros: la petición, la respuesta, y una referencia al siguiente *middleware* que debe ser llamado (`next`). Cualquier *middleware* puede finalizar la cadena enviando algo en la respuesta al cliente, pero si no lo hace, debe obligatoriamente llamar al siguiente eslabón (`next`).

Podemos emplear diversas llamadas a `use` para cargar distintos *middlewares*, y el orden en que hagamos estas llamadas es importante, porque marcará el orden en que se ejecutarán dichos *middlewares*.

```
app.use(express.json());
app.use(function(...) { ... });
app.use(...);
```

3. Flujo básico de una petición y respuesta

Teniendo en cuenta todo lo visto anteriormente, el flujo elemental de una petición cliente que llega a un servidor Express es el siguiente:

Existen algunos middlewares realmente útiles, como el ya citado `express.json()`, o el servidor de contenido estático que veremos más adelante, o el enrutador (*router*), que típicamente es el último eslabón de la cadena, y se emplea para configurar diferentes rutas de petición disponibles, y la respuesta para cada una de ellas. Veremos cómo enlazarlos a continuación.

4. Añadir middleware a enrutadores concretos

Cuando definimos enrutadores independientes en archivos separados podemos añadir middleware por separado también a cada enrutador, empleando el método `use` del propio enrutador. Por ejemplo, podemos añadir un middleware en el archivo `routes/contactos.js` que muestre por consola la fecha actual. Lo haremos en un proyecto llamado *ContactosREST_v2_Middleware*, copia de un ejemplo previo llamado *ContactosREST_v2*:

```
let router = express.Router();

router.use((req, res, next) => {
    console.log(new Date().toString());
    next();
});

...
```

Ejercicio 1:

Crea una copia del proyecto *LibrosREST_v2* llamada **LibrosREST_v2_Middleware**. Vamos a definir dos middlewares propios:

- Uno lo definiremos únicamente para el enrutador de libros (`routes/libros.js`), de forma que cada vez que accedamos a un servicio de ese enrutador, se mostrará por consola la fecha, método (GET, POST, etc) y URI solicitada. Para obtener estos dos últimos datos, deberás utilizar las propiedades `req.method` y `req.url` del objeto de la petición (`req`). También puedes usar `req.baseUrl` para obtener la URL base, en el caso de que utilices enrutadores con URL base, como en este ejercicio.
- El otro será global (en `index.js` o `app.js` , dependiendo de cómo hayas llamado al programa principal), y deberá enviar por JSON un mensaje de "En mantenimiento", sin permitir acceder a ningún servicio (es decir, que no llame al método `next`). Siempre que este middleware esté activo, la aplicación no funcionará (sólo enviará "En mantenimiento" para cada solicitud que le llegue). Cuando se desactive/comente, la aplicación funcionará con normalidad.