Ús de *middleware*


El terme *middleware* ha sigut esmentat algunes vegades amb anterioritat, però encara no havíem explicat de què es tracta. En termes generals, podríem definir *middleware* com un programari que s'executa enmig d'altres dos. En l'àmbit en què l'estem tractant, un **middleware** és una funció que gestiona peticions i respostes HTTP, de manera que pot manipular-les i passar-les al següent *middleware* perquè les continue processant, o bé acabar el procés i enviar per si mateixa una resposta al client.

1. El mètode *use*

Existeixen diferents *middlewares* disponibles en els mòduls de Node, com ara el processador `express.json()` que hem emprat en exemples previs per a processar el cos de peticions POST i poder accedir al seu contingut de forma més còmoda. El middleware s'aplica sobre una aplicació a través del mètode `use` d'aquesta. Per això, quan féiem:

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

... estàvem activant el *middleware* corresponent d'Express per a processar els cossos de les peticions i extraure d'ells la informació JSON que continguen, deixant-la preparada per a ser accedida.

2. Definir el nostre propi *middleware*

A més d'aquest i altres exemples de *middleware* fet per terceres parts i que podem incorporar als nostres projectes, també podem definir la nostra pròpia funció de *middleware*, i mitjançant el mètode `use` de l'aplicació, incloure-la en la cadena de processament de la petició i la resposta. Per exemple, el següent middleware saca per consola l'adreça IP del client que fa la petició:

```
app.use((req, res, next) => {
    console.log("Petició des de", req.ip);
    next();
});
```

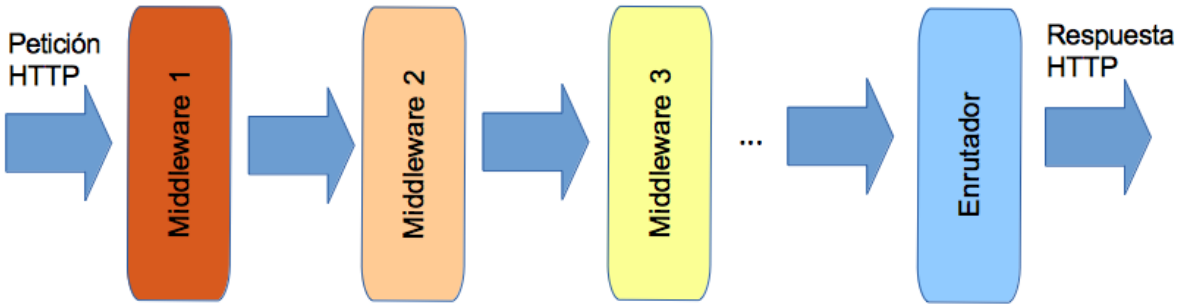
Com veiem, el *middleware* no és més que una funció que accepta tres paràmetres: la petició, la resposta, i una referència al següent *middleware* que ha de ser anomenat (`next`). Qualsevol *middleware* pot finalitzar la cadena enviant alguna cosa en la resposta al client, però si no ho fa, deu obligatòriament cridar a la següent baula (`next`).

Podem emprar diverses anomenades a `use` per a carregar diferents *middlewares*, i l'ordre en què fem aquestes crides és important, perquè marcarà l'ordre en què s'executaran dits *middlewares*.

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

3. Flux bàsic d'una petició i resposta

Tenint en compte tot el vist anteriorment, el flux elemental d'una petició client que arriba a un servidor Express és el següent:



Existeixen alguns middlewares realment útils, com el ja citat `express.json()`, o el servidor de contingut estàtic que veurem més endavant, o l'encaminador (*router*), que típicament és l'última baula de la cadena, i s'empra per a configurar diferents rutes de petició disponibles, i la resposta per a cadascuna d'elles. Veurem com enllaçar-los a continuació.

4. Afegir middleware a encaminadors concrets

Quan definim encaminadors independents en arxius separats podem afegir middleware per separat també a cada encaminador, emprant el mètode `use` del propi encaminador. Per exemple, podem afegir un middleware en l'arxiu `routes/contactes.js` que mostre per consola la data actual. Ho farem en un projecte anomenat *ContactesREST_v2_Middleware*, còpia d'un exemple previ anomenat *ContactesREST_v2*:

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

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

...
```

Exercici 1:

Crea una còpia del projecte *LlibresREST_v2* anomenada **LlibresREST_v2_Middleware**. Definirem dos middlewares propis:

- Un ho definirem únicament per a l'encaminador de llibres (`routes/llibres.js`), de manera que cada vegada que accedim a un servei d'aqueix encaminador, es mostrarà per consola la data, mètode (GET, POST, etc) i URI sol·licitada. Per a obtindre aquestes dues últimes dades, hauràs d'utilitzar les propietats `req.method` i `req.url` de l'objecte de la petició (`req`). També pots usar `req.baseUrl` per a obtindre la URL base, en el cas que utilitzes encaminadors amb URL base, com en aquest exercici.
- L'altre serà global (en `index.js` o `app.js` , depenent de com hages anomenat al programa principal), i haurà d'enviar per JSON un missatge de "En manteniment", sense permetre accedir a cap servei (és a dir, que no cride al mètode `next`). Sempre que aquest middleware estiga actiu, l'aplicació no funcionarà (només enviarà "En manteniment" per a cada sol·licitud que li arribe). Quan es desactive/comente, l'aplicació funcionarà amb normalitat.