Introducció a Express


Express és un framework lleuger i, alhora, flexible i potent per a desenvolupament d'aplicacions web amb Node. En primer lloc, es tracta d'un framework lleuger perquè no ve carregat de sèrie amb tota la funcionalitat que un framework podria tindre, a diferència d'altres frameworks més pesats i autocontinguts com Symfony o Ruby on Rails. Però, a més, es tracta d'un framework flexible i potent perquè permet afegir-li, a través de mòduls Node i de *middleware*, tota la funcionalitat que es requerisca per a cada tipus d'aplicació. D'aquesta manera, podem utilitzar-ho en la seua versió més lleugera per a aplicacions web senzilles, i dotar-li de més elements per a desenvolupar aplicacions molt complexes.

Com veurem, amb Express podem desenvolupar tant servidors típics de contingut estàtic (HTML, CSS i JavaScript), com a serveis REST accessibles des d'un client, i per descomptat, aplicacions que combinen totes dues coses.

Podeu trobar informació actualitzada sobre Express, tutorials i altra informació en la seua pàgina oficial.

1. Descàrrega i instal·lació

La instal·lació d'Express és tan senzilla com la de qualsevol altre mòdul que vulguem incorporar a un projecte Node. Simplement necessitem executar el corresponent comando `npm install` en la carpeta del projecte on vulguem afegir-lo (i, prèviament, el comando `npm init` en el cas que encara no hàgem creat l'arxiu *package.json*):

```
npm install express
```

1.1. Exemple de servidor bàsic amb Express

Crearem un projecte anomenat "*ProvaExpress*" en la carpeta de "*ProjectesNode/Proves*", i instal·larem Express en ell. Després, creem un arxiu anomenat `index.js` amb aquest codi:

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

El codi, com podem veure, és molt senzill. Primer incloem la llibreria, després inicialitzem una instància d'Express (normalment s'emmagatzema en una variable anomenada `app`), i finalment, la posem a escoltar pel port que vulguem. En aquest cas, s'ha triat el port 8080 per a no interferir amb el port per defecte pel qual s'escolten les peticions HTTP, que és el 80. També és habitual trobar exemples de codi en Internet que usen

els ports 3000 o 6000 per a les proves. És indiferent el número de port, sempre que no interferisca amb un altre servei que tinguem en marxa en el sistema.

Per a provar l'exemple des de la nostra màquina local, n'hi ha prou amb posar en marxa l'aplicació Node, obrir un navegador i accedir a la URL:

http://localhost:8080

Si proves d'executar l'aplicació Node des de Visual Studio Code, veuràs que no finalitza. Hem creat un xicotet servidor Express que queda a l'espera de peticions dels clients. No obstant això, encara no està preparat per a respondre a cap d'elles, per la qual cosa donarà un missatge d'error en intentar accedir a qualsevol URL. Això ho solucionarem en els següents apartats.

2. Express com a proveïdor de serveis

Ara que ja sabem què és Express i com incloure-ho en les aplicacions Node, veurem un dels principals usos que se li dona: el de servidor que proporciona serveis REST als clients que ho sol·liciten. Per a això, i com a pas previ, hem de comprendre i assimilar com es processen les rutes en Express, i com s'aïlla el tractament de cadascuna, de manera que el codi resulta molt modular i independent entre rutes.

2.1. Un primer servei bàsic

Partint de l'exemple anterior, afegirem una sèrie de rutes en el nostre servidor principal per a donar suport als serveis associats a aquestes. Una vegada hem inicialitzat l'aplicació (variable `app`), n'hi ha prou amb anar afegint mètodes (`get`, `post`, `put` o `delete`), indicant per a cadascun la ruta o URI que ha d'atendre, i el callback o funció que s'executarà en aqueix cas. Per exemple, per a atendre una ruta anomenada `/benvinguda` per GET, afegiríem aquest mètode:

```
let app = express();

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

...
```

El callback en qüestió rep dos paràmetres sempre: l'objecte que conté la petició (típicament anomenat `req`, abreviatura de *request*), i l'objecte per a emetre la resposta (típicament dit `res`, abreviatura de *response*). Més endavant veurem quines altres coses podem fer amb aquests objectes, però de moment emprarem la resposta per a enviar (`send`) text al client que va sol·licitar el servei, i `req` per a obtindre determinades dades de la petició. Podem tornar a llançar el servidor Express, i provar aquest nou servei accedint a la URL corresponent:

http://localhost:8080/benvinguda

De la mateixa manera, s'afegiran la resta de mètodes per a atendre les diferents opcions de l'aplicació. Per exemple:

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

> **Exercici 1:**
>
> Crea una carpeta anomenada "**ExpressBasic**" en la carpeta d'exercicis "*ProjectesNode/Exercicis*". Instal·la Express en ella, i defineix un servidor bàsic que responga per GET a aquestes dues URIs:
>
> - URI `/data` : el servidor enviarà com a resposta al client la data i hora actuals. Pots utilitzar el tipus `Date` de JavaScript sense més, o també pots "recrear-te" amb la llibreria "moment" vista en sessions anteriors, si vols.
> - URI `/usuari` : el servidor enviarà el login de l'usuari que va entrar al sistema. Necessitaràs emprar la llibreria "os" del nucli de Node, vista en sessions anteriors, per a obtindre aquest usuari.

3. Elements bàsics: aplicació, petició i resposta

Existeixen tres elements bàsics sobre els quals se sustenta el desenvolupament d'aplicacions en Express: l'aplicació en si, l'objecte amb la petició del client, i l'objecte amb la resposta a enviar.

3.1. L'aplicació

L'aplicació és una instància d'un objecte Express, que típicament s'associa a una variable anomenada `app` en el codi:

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

Tota la funcionalitat de l'aplicació (mètodes de resposta a peticions, inclusió de *middleware*, etc) s'assentisca sobre aquest element. Compta amb una sèrie de mètodes útils, que anirem veient en futurs exemples, com són:

- `use(middleware)` : per a incorporar *middleware* al projecte
- `set(propietat, valor)` / `get(propietat)` : per a establir i obtindre determinades propietats relatives al projecte
- `listen(port)` : per a fer que el servidor es quede escoltant per un port determinat.
- ...

3.2. La petició

L'objecte de petició (típicament el trobarem en el codi com `req`) es crea quan un client envia una petició a un servidor Express. Conté diversos mètodes i propietats útils per a accedir a informació continguda en la petició, com:

- `params`: la col·lecció de paràmetres que s'envia amb la petició
- `query`: amb la *query string* enviada en una petició GET (informació darrere de l'interrogant `?` en una URL)
- `body`: amb el cos enviat en una petició POST
- `files`: amb els arxius pujats des d'un formulari en el client
- `get(capçalera)`: un mètode per a obtindre diferents capçaleres de la petició, a partir del seu nom
- `path`: per a obtindre la ruta o URI de la petició
- `url`: per a obtindre la URI juntament amb qualsevol *query string* que hi haja a continuació
- ...

3.3. La resposta

L'objecte resposta es crea juntament amb el de la petició, i es completa des del codi del servidor Express amb la informació que es vaja a enviar al client. Típicament es representa amb la variable o paràmetre `res`, i compte, entre altres, amb aquests mètodes i propietats d'utilitat:

- `estatus(codigo)`: estableix el codi d'estat de la resposta
- `set(capçalera, valor)`: estableix cadascuna de les capçaleres de resposta que es necessiten
- `redirect (estat, url)`: redirigeix a una altra URL, amb el corresponent codi d'estat
- `send([estat], cos)`: envia el contingut indicat, juntament amb el codi d'estat associat (de manera opcional, si no s'envia aquest per separat).
- `json([estat], cos)`: envia contingut JSON específicament, juntament amb el codi d'estat associat (opcional)
- `render(vista, [opcions])`: per a mostrar una determinada vista com a resposta, podent especificar opcions addicionals i un callback de resposta.
- ...