


Utilizando Express

 gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/utilizando-express.html

Express JS es un librería con un alto nivel de popularidad en la comunidad de Node JS e inclusive este mismo adoptó Express JS dentro de su ecosistema, lo cual le brinda un soporte adicional, pero esto no quiere decir que sea la única o la mejor, aunque existen muchas otras librerías y frameworks para crear REST API, Express JS si es la más sencilla y poderosa, según la definición en la página Web de Express JS: "Con miles de métodos de programa de utilidad HTTP y *middleware* a su disposición, la creación de una API sólida es rápida y sencilla."

Mezclando features

Normalmente cuando un *feature* de la aplicación está listo y probado en su rama de *git* se mezcla con la rama de desarrollo o principal según sea el caso, por lo cual vamos a mezclar nuestro *feature* en la rama y principal y crear otro a partir de esta:

```
git checkout master
git merge feature/feature/01-simple-web-server
```

Puede que existan conflictos al mezclar la rama feature con la master, se asume que el lector tiene los conocimientos básicos para solventar esta posible situación

La mezcla anterior se hizo de manera local ya que estamos trabajando en un flujo individual pero cuando se trabaja en equipo esta se realiza de manera remota mediante *Pull Request* (PR) y luego se actualiza la rama principal local.

Antes de continuar vamos a crear una rama para trabajar:

```
git checkout -b feature/02-using-express
```

Añadiendo Express

Instalamos el módulo de `express` como una dependencia de nuestra aplicación:

```
npm install -S express
```

Esto añadirá una nueva entrada en nuestro archivo `package.json` en la sección de dependencies indicando la versión instalada.

Luego reemplazamos el código existente del archivo `index.js` por el siguiente:

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

const app = express();
const hostname = '127.0.0.1';
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Como lo vemos es muy similar al anterior pero con unas sutiles diferencias:

- Utilizamos la librería de `express` en vez del módulo `http`.
- Creamos una aplicación de `express` (inclusive se pueden crear varias) y la almacenamos en la variable `app`.
- Esta vez solo aceptaremos peticiones en la raíz del proyecto con el verbo GET de HTTP.

Si el servidor se está ejecutando lo detenemos con `CTRL+C`, lo ejecutamos nuevamente con el comando `node index` y abrimos el navegador en la siguiente dirección `http://localhost:3000/`.

Aparentemente vemos el mismo resultado, pero `express` nos ha brindado un poco más de simplicidad en el código y ha incorporado una gran cantidad de funcionalidades, las cuales veremos más adelante.

Antes de finalizar hacemos commit de nuestro trabajo

```
git add --all
git commit -m "Add express JS"
```

Organizando la aplicación

Antes de continuar vamos a organizar nuestra aplicación, crearemos diferentes módulos cada uno con su propósito específico, ya que esto permite separar la responsabilidad de cada uno de ellos y nuestro archivo `index.js` no se convierta en una aplicación monolítica.

Creamos un directorio llamado `server` donde se almacenarán todos los archivos relacionados con el servidor Web así como su nombre lo indica:

```
mkdir server
touch server/index.js
```

Tomaremos como ventaja la convención de Node JS en la nomenclatura de los módulos de la raíz de cada carpeta llamándolos `index.js`

Copiamos el contenido de nuestro archivo `index.js` y lo pegamos en `server/index.js` con las siguientes modificaciones:

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

const app = express();

app.get('/', (req, res) => {
  res.send('Hello World');
});

module.exports = app;
```

Como podemos observar ahora nuestra aplicación de `express` es un módulo que tiene una sola responsabilidad de crear nuestra aplicación y es independiente a la librería que utilizamos.

Creamos el directorio y módulo básico de configuración:

```
mkdir -p server/config
touch server/config/index.js
```

En el archivo `server/config/index.js` es donde centralizamos todas las configuraciones de la aplicación que hasta el momento tenemos, así cualquier cambio será en un solo archivo y no en varios, finalmente lo exportamos como un módulo:

```
const config = {
  server: {
    hostname: '127.0.0.1',
    port: 3000,
  },
};

module.exports = config;
```

De vuelta en nuestro archivo raíz `index.js`, lo reescribimos quitándole las múltiples responsabilidades que tenía y dejándolo solo con la responsabilidad de hacer el "*bootstrapping*" de la aplicación, utilizando los módulos creados posteriormente:

```
const http = require('http');

const app = require('./server');
const config = require('./server/config');

const { hostname, port } = config.server;

const server = http.createServer(app);

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Como podemos ver en el código anterior volvimos a incorporar la librería `http`, esto con el fin de mostrar que express es también compatible con la versión de servidor que crea http en la función `createServer` e inclusive si en el futuro fuera a incorporar un servidor `https`, solo es incluir la librería y duplicar la línea de creación del servidor.

Si visitamos nuevamente el navegador en la dirección `http://localhost:3000` todo debe estar funcionando sin ningún problema.

Antes de finalizar hacemos commit de nuestro trabajo

```
git add .  
git commit -m "App boilerplate"
```

Más información:

[Express JS](#)