TEMA 3 EXPRESS.JS

Parte I – Introducción a Express

¿Qué es Express?

- Es un framework ligero, flexible y potente.
 - Ligero: No viene cargado con toda la funcionalidad (Como Symphony o Ruby)
 - Flexible y potente: Permite añadir funcionalidad através de los ya conocidos módulos de Node.js y de middleware.
- Podemos crear aplicaciones:
 - Servidores de contenido estático (HTML, CSS y JavaScript)
 - Aplicaciones accesibles desde Cliente
 - Combinar ambas cosas
- □ Página oficial de Express: http://expressjs.com/

Instalación

- □ Tenemos dos formas de hacerlo:
 - Instalar Express para un proyecto concreto (Instalándolo con npm como un módulo como siempre) npm install express.
 - Crea carpeta "PruebaExpress" y en el index.js escribe:

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

Generador de proyectos Express, usando "express-generator". Se instalará Express de forma global con una infraestructura global para todos los proyectos.

npm install express-generator -g

Ejemplo Proyecto Generado

- Nos colocamos en la carpeta de Pruebas y ejecutamos: express PruebaExpressGenerator
- Nos metemos en la carpeta generada y ejecutamos: npm install. Con lo que instalaremos todos los módulos y dependencias que se necesitan para la infraestructura de proyecto pre-generada
- Para lanzar el proyecto usamos el siguiente comando: set DEBUG=PruebaExpressGenerator: * & npm start (En caso de Mac o Linux) : DEBUG=PruebaExpressGenerator: * npm start
- Vamos a un navegador y ponemos la url: http://localhost:3000

Elementos básicos de EXPRESS

- Existen 3 elementos fundamentales en aplicaciones Express:
 - La aplicación en si.
 - El objeto con la petición del Cliente
 - El objeto con la respuesta del Servidor

La aplicación

□ Es una instancia de un objeto EXPRESS, que se suele asociar a una variable llamada "app":
const express = require('express'):

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

- TODA la funcionalidad se asienta sobre esta variable.
- ☐ Cuanta con una serie de <u>métodos</u>:
 - use(middleware): Para incorporar middleware al proyecto.
 - set(propiedad, valor) / get(propiedad): Establece y obtiene determinadas propiedades relativas al proyecto.
 - listen(puerto): hace que el servidor se quede escuchando por un puerto determinado. Opcionalmente, se puede definir un callback a ejecutar cuando llegue una petición
 - render(vista, [opciones], callback): Muestra una determinada vista estática como respuesta, pudiendo especificar opciones adicionales y un callback de respuesta
 - Etc, los iremos viendo.

La petición

- Normalmente será la variable req o request.
- ☐ Se crea cuando un cliente envía una petición al Server Express.
- □ Contiene propiedades y métodos:
 - params: la colección de parámetros que se envía con la petición
 - query: con la query string enviada en una petición GET
 - body: con el cuerpo enviado en una petición POST
 - files: con los archivos subidos desde un formulario en el cliente
 - get(cabecera): un método para obtener distintas cabeceras de la petición, a partir de su nombre
 - path: para obtener la ruta o URI de la petición
 - url: para obtener la URI junto con cualquier query string que haya a continuación
 - **.**..

La respuesta

- □ Variable res o response
- Se crea junto con la petición y lo completa el Servidor Express con lo que se quiera reenviar al Cliente.
- □ También tiene propiedades y métodos:
 - status(codigo): establece el código de estado de la respuesta
 - set(cabecera, valor): establece cada una de las cabeceras de respuesta que se necesiten
 - redirect (estado, url): redirige a otra URL, con el correspondiente código de estado
 - send([estado], cuerpo): envía el contenido indicado, junto con el código de estado asociado (de forma opcional, si no se envía éste por separado).
 - json([estado], cuerpo): envía contenido JSON específicamente, junto con el código de estado asociado (opcional)
 - ...

El middleware

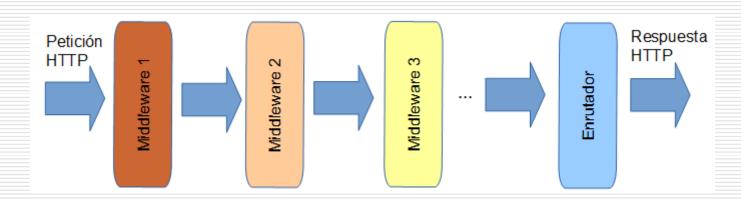
- Es una función Javascript que gestiona peticiones y respuestas HTTP, de forma que puede manipularlas y pasarlas al siguiente *middleware* para que las siga procesando, o bien terminar el proceso y enviar por sí misma una respuesta al cliente.
- Acepta tres parámetros: la petición, la respuesta, y una referencia al siguiente middleware que debe ser llamado. El tercer parámetro puede ser una respuesta al Cliente lo que querrá decir que se finaliza, en caso contrario obligatoriamente debe ser NEXT.
- ☐ Tenemos varios tipos de middle en Node:
 - Body-parser: Para procesar el cuerpo de peticiones POST y poder acceder a su contenido de forma más cómoda
 - Definir nuestro propio middleware y con 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();
});
```

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(bodyParser);
app.use(function(...) { ... });
app.use(...);
```

Flujo básico de una petición y respuesta



Existen algunos middlewares realmente útiles, como el ya citado body-parser, 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 ejemplos de su uso más adelante.