APLICACIONES EXPRESS

Tiempo estimado: 10m

Una aplicación (*application*) no es más que un programa diseñado para hacer algo. Las aplicaciones webs (*web applications*) tienen como función servir recursos, documentos, archivos, contenido, etc. a través de una red como, por ejemplo, Internet. Las aplicaciones Express (*Express applications*) son aplicaciones webs servidoras desarrolladas mediante el *framework* Express.

La lección comienza introduciendo el concepto de aplicación Express. Cuáles son sus componentes. Y los archivos más importantes de la aplicación. A continuación, presentamos cómo configurar la aplicación mediante el uso de opciones. Finalizamos introduciendo el concepto de controlador de petición, pieza angular de las aplicaciones Express.

Al finalizar la lección, el estudiante sabrá:

- Qué es una aplicación Express.
- Cuáles son los archivos claves de una aplicación Express.
- Cómo configurar opciones de aplicación.
- Qué son los controladores de petición.

INTRODUCCIÓN

Tal como vimos en la primera lección, Express es un *framework* para el desarrollo de aplicaciones webs bajo la plataforma Node. Las aplicaciones escritas usando este *framework* se conocen formalmente como aplicaciones Express (*Express application*). Son servidoras, escuchan en un determinado puerto, se ejecutan bajo Node y sirven recursos mediante el protocolo HTTP.

La lógica de Express se encuentra en el módulo express descargable mediante NPM. Debe incorporarse a las dependencias del proyecto. Recordemos, propiedad dependencies del archivo package.json.

Los componentes más importantes de una aplicación Express son:

- Los controladores de petición, los cuales se encargan de procesar parcial o totalmente las peticiones recibidas por la aplicación.
- La pila de *middleware* que contiene el flujo de procesamiento que debe seguir toda petición HTTP recibida por la aplicación. Mediante este flujo, por un lado, se procesa las peticiones y, por otro lado, se escribe la respuesta a remitir al cliente.
- El encaminador, componente que se encarga de atender determinadas peticiones HTTP según su *path* o ruta. Mediante este elemento, podemos fijar controladores que se ejecuten sólo con determinadas peticiones.
- El motor de plantillas que facilita la generación de contenido dinámico.

ESTRUCTURA DE DIRECTORIOS DE UNA APLICACIÓN EXPRESS

Toda aplicación Express presenta una estructura de directorios en la que se almacena los distintos archivos que la forman. Express no tiene definida ninguna estructura especial, pero se recomienda utilizar una estructura similar en todos los proyectos de la organización, evitando así que cada una parezca de un padre y una madre.

Para su creación, se puede utilizar el generador oficial de Express, express-generator, o el de Justo.js, justo-generator-express. La función de un generador no es más que crear una estructura inicial para la aplicación. Esto puede hacerse con ambos generadores. El de Justo.js, además, añade comandos adicionales para crear ciertos componentes de la aplicación como encaminadores, rutas, vistas, etc.

Archivo app.js

Una vez creada la estructura de directorios de la aplicación, lo siguiente es crear la aplicación propiamente dicha. El *script* app.js, ubicado en la raíz del proyecto, es el archivo elegido por convenio para contener la lógica de arranque de la aplicación.

Este archivo tiene básicamente los siguientes objetivos:

- Crear la aplicación Express.
- Configurar la aplicación.
- Enlazar la aplicación a una dirección de IP y puerto.

Creación de la aplicación

Para crear la aplicación, no hay más que invocar la función express(), obtenida al importar el módulo express:

```
//imports
import express from "express";
…

//creación de la aplicación
const app = express();
```

Configuración de la aplicación

Una vez creada la aplicación, lo siguiente es configurarla. Generalmente, tal como veremos a lo largo del curso, consiste en indicar el motor de plantillas, los componentes de *middleware* a usar para procesar las peticiones HTTP, el registro de eventos, etc.

He aquí un ejemplo ilustrativo, que ayudará a ir abriendo boca:

```
//config
app.set("env", process.env.NODE_ENV || "development");
app.set("config", require(`./config/${app.get("env")}`));
app.set("host", process.env.HOST || app.get("config").host);
app.set("port", process.env.PORT || app.get("config").port);

//config template engine
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "hbs");
app.set("view cache", app.get("env") == "production");
app.engine("hbs", hbs.__express);
hbs.registerPartials(path.join(__dirname, "views/partials/"));

//middleware
app.use(require("helmet")());
app.use(require("express-session")({
  name: "sinfo",
  cookie: {maxAge: "60000"},
  secret: "1468654010998",
  resave: false,
  saveUninitialized: false,
  genid: function(req) { return Date.now().toString(); }
}));
app.use(require("morgan")("combined"));
app.use(require("serve-favicon")(path.join(__dirname, "public", "/images/favicon.png"), {
  maxAge: 60000
}));
app.use(require("serve-static")(path.join(__dirname, "public"), {
  maxAge: 60000
}));
app.use(require("cookie-parser")("1468654005796"));
app.use(require("body-parser").urlencoded({
  extended: true
}));
```

Solicitud de socket de red

Como paso final, hay que poner la aplicación Express a escuchar en un determinado puerto y dirección IP:

```
http.createServer(app).listen(app.get("port"), app.get("host"), function() {
  console.log("Listening...");
});
```

Opciones de aplicación

Las opciones de aplicación (*application settings*) son parámetros a través de los cuales configurar determinados aspectos de la aplicación, los cuales iremos presentando a lo largo del curso. Las opciones se configuran mediante los métodos set(), enable() y disable() del objeto aplicación. Mediante set(), fijamos el valor de una opción:

```
set(name, value)
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre de la opción.
value	object	Valor a asignar a la opción.

Mediante enable() y disable(), fijamos valores de opciones booleanas. Con enable(), fijamos true; y con disable(), false:

```
enable(name)
disable(name)
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre de la opción.

He aquí unos ejemplos ilustrativos:

```
app.set("views", path.join(__dirname, "views"));
app.enable("trust proxy");
```

Para consultar los valores de las opciones, se utiliza los métodos get(), enabled() y disabled(). El primero devuelve el valor de la opción; el segundo si el valor de la opción es true; y el tercero, si es false:

```
get(name) : object
enabled(name) : boolean
disabled(name) : boolean
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre de la opción.

Opción de modo de entorno

El modo de entorno (*environment mode*) indica el entorno de ejecución de la aplicación. De cara a Express, básicamente, se distinguen dos entornos: el de producción y el de desarrollo. El entorno de desarrollo (*development environment*) es aquel en el que se escribe o implementa la aplicación. Generalmente, nuestra máquina. En cambio, el entorno de producción (*production environment*) es aquel en el que se encuentra la aplicación para su uso por parte de los usuarios.

Atendiendo al entorno de ejecución de la aplicación, se suele configurar unas cosas u otras. Por ejemplo, cuando estamos en desarrollo, cuanta más información dispongamos de los errores que se producen, mejor. En cambio, en producción, se suele limitar la información de depuración porque puede ralentizar el sistema y dar información a los usuarios que no les interesa.

Mediante la opción env, se informa a Express bajo qué entorno de ejecución se encuentra la aplicación: development o production. De manera predeterminada, su valor lo toma de la variable de entorno NODE_ENV, o sea, del valor devuelto por process.env.NODE_ENV. Si no existe esta variable o no tiene valor, automáticamente se pondrá a development. Pero podemos fijar su valor explícitamente mediante el método set(), tal como acabamos de ver hace unos instantes.

Es muy común que las aplicaciones presenten configuraciones específicas según cada entorno. En estos casos, se usa proposiciones similares a:

```
if (app.get("env") == "development") {
  //configuración específica del entorno de desarrollo
```



```
} else if (app.get("env") == "production") {
  //configuración específica del entorno de producción
}
```

Veamos un ejemplo ilustrativo:

```
if (app.get("env") == "development") {
  app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.render("error", {
      message: err.message,
      error: err
    });
  });
}
```

Opción x-powered-by

De manera predeterminada, Express añade automáticamente el campo de cabecera X-Powered-By: Express a las respuestas HTTP remitidas por la aplicación. Por cuestiones de seguridad, no se recomienda adjuntar esta cabecera, porque proporciona información sobre el tipo de aplicación que se encuentra detrás. Algunos usuarios podrían utilizar esa información para explotar agujeros o problemas de seguridad del *framework*.

Por ello, se recomienda configurar la aplicación para que este campo de cabecera no se añada, mediante la opción booleana x-powered-by. Para desactivarla, no hay más que usar el método disable():

```
app.disable("x-powered-by");
```

Opciones host y port

Mediante las opciones host y port, especificamos la interfaz de red y el puerto en el que debe escuchar la aplicación Express. Actualmente, estas opciones no las usa internamente Express. Pero se recomienda crearla durante la fase de configuración y usarla en el momento de atarla a una dirección de IP y puerto. Así pues, tendremos algo como:

```
//crear aplicación
app = express();

//configurar aplicación
app.set(…);
app.set(…);
app.set("config", require(`./config/${app.get("env")}`));
app.set("host", process.env.HOST || app.get("config").host);
app.set("port", process.env.PORT || app.get("config").port);

//…

//atar aplicación a IP y puerto
http.createServer(app).listen(app.get("port"), app.get("host"), function() {
  …
});
```

Las opciones no definidas por Express se conocen formalmente como opciones personalizadas (*custom settings*). Son un buen lugar donde almacenar información útil de la aplicación.

Controladores de petición

Un controlador de petición (*request handler*) es una función que procesa total o parcialmente una petición HTTP. Se registran principalmente en la pila de *middleware* o en los encaminadores, componentes que veremos más adelante en este curso.

La signatura de estos controladores es:

```
function(req, res)
function(req, res, next)
```

Parámetro	Tipo de datos	Descripción
req	Request	Solicitud HTTP en procesamiento.

res	Response	Respuesta HTTP que se está generando.
next	function	Función que debe invocar el controlador para indicarle a Express que ejecute el siguiente componente de la pila de procesamiento: next([error]).

Cuando Express invoca la función, le pasará como argumentos: la solicitud HTTP, que podrá accederse mediante el parámetro req; la respuesta HTTP, que se puede acceder mediante el parámetro res; y una función especial, next(), a través de la cual la función indica al motor si debe seguir invocando los restantes controladores registrados en la aplicación.

Este último parámetro, también conocido como función de continuación de flujo, tiene dos sobrecargas:

```
function next()
function next(error)
```

Cuando no le pasamos ningún argumento, estamos indicando que el controlador ha finalizado correctamente y hay que pasar al siguiente. En cambio, si le pasamos un argumento, éste se comportará como error y le indica al motor que ejecute la función de control de errores. Más adelante en el curso, veremos más acerca de este tipo de funciones. Por ahora, asumamos que todo va siempre bien.

Es posible, para determinadas solicitudes HTTP, que no sea necesario que el mensaje recibido del cliente tenga que atravesar toda la pila. En ocasiones, algunos controladores pueden dar por finalizado el procesamiento, porque han generado toda la respuesta o se consideran el último eslabón de la cadena para esa determinada petición. Para que una petición fluya a lo largo de todo el flujo de procesamiento o de parte de él se utiliza el parámetro función next().

Cuando una función termina su procesamiento debe invocar el parámetro next() para informar, así, que ha finalizado su funcionalidad, pero no da por terminado el flujo de procesamiento. Y por lo tanto hay que invocar el siguiente controlador registrado. Si no se hace, el motor asumirá que este controlador ha finalizado el procesamiento de la petición HTTP y la respuesta está generada completamente, por lo que *no* continuará ejecutando más controladores. Y dará por terminado el procesamiento de la petición HTTP.

Por ejemplo, supongamos que deseamos implementar un registro de eventos que escriba una entrada por cada solicitud HTTP recibida. Esta función de *middleware* no genera nada en el objeto respuesta. Sólo lee la petición HTTP, extrae la información que necesita y escribe la entrada en el registro de eventos. Nada más. Pues este tipo de funciones debe invocar la función next() para indicarle así a Express que continúe con la función que le sigue. Si no lo hace, el procesamiento finalizará sin que la respuesta se haya redactado.

Otro ejemplo. Ahora, consideremos una función que se encarga de servir determinados archivos estáticos del disco. Si el archivo no se encuentra en disco, la función no puede hacer su trabajo. En vez de generar un error, lo que hace es indicarle al motor que continúe con el flujo de procesamiento. La función no ha podido hacer nada, pero igual alguna de las funciones que le sigue puede hacer algo. En cambio, si la función localiza el archivo, generará la respuesta a remitir al cliente. En este caso, está claro que nadie más debe hacer nada en la respuesta por lo que se considera el único o el último que debe participar en su redacción. Por lo que no invocará el parámetro función next(), indicándole así al motor que la respuesta *ya* se ha generado y *no hay que continuar* con lo que resta del flujo de procesamiento.

He aquí un ejemplo ilustrativo de una función controladora que escribe en la salida estándar información sobre la petición HTTP en procesamiento:

```
app.use(function(req, res, next) {
  console.log(req.ip, req.method, req.originalUrl);
  next();
});
```

