GUIA EXPRESS

Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.

¿Por qué utilizar Express?

Express lo ayuda a responder a las solicitudes con soporte de ruta para que pueda escribir respuestas a URL específicas
Admite múltiples motores de plantillas para simplificar la generación de HTML.
Lo bueno de esto es que es muy simple y de código abierto.

Instalación
Suponiendo que ya ha instalado Node.js, cree un directorio para que contenga la aplicación y conviértalo en el directorio de trabajo.

$ mkdir myapp
$ cd myapp

Utilice el mandato npm init para crear un archivo package.json para la aplicación. Para obtener más información sobre cómo funciona package.json, consulte Specifics of npm's package.json handling.
$ npm init

Este mandato solicita varios elementos como, por ejemplo, el nombre y la versión de la aplicación. Por ahora, sólo tiene que pulsar INTRO para aceptar los valores predeterminados para la mayoría de ellos, con la siguiente excepción:
entry point: (index.js)

Especifique app.js o el nombre que desee para el archivo principal. Si desea que sea index.js, pulse INTRO para aceptar el nombre de archivo predeterminado recomendado.

A continuación, instale Express en el directorio myapp y guárdelo en la lista de dependencias. Por ejemplo:
$ npm install express --save

Para instalar Express temporalmente y no añadirlo a la lista de dependencias, omita la opción --save:
$ npm install express

Direccionamiento básico
El direccionamiento hace referencia a la determinación de cómo responde una aplicación a una solicitud de cliente en un determinado punto final, que es un URI (o una vía de acceso) y un método de solicitud HTTP específico (GET, POST, etc.).

Cada ruta puede tener una o varias funciones de manejador, que se excluyen cuando se correlaciona la ruta.

La definición de ruta tiene la siguiente estructura:

app.METHOD(PATH, HANDLER)

Donde:

app es una instancia de express.
METHOD es un método de solicitud HTTP.
PATH es una vía de acceso en el servidor.
HANDLER es la función que se ejecuta cuando se correlaciona la ruta.

El siguiente ejemplo ilustra la definición de rutas simples.
Responda con Hello World! en la página inicial:

```
app.get('/', function (req, res) {
            res.send('Hello World!');
            });
```

Responda a la solicitud POST en la ruta raíz (/), la página de inicio de la aplicación:

```
app.post('/', function (req, res) {
            res.send('Got a POST request');
            });
```

Responda a una solicitud PUT en la ruta /user:

```
app.put('/user', function (req, res) {
            res.send('Got a PUT request at /user');
            });
```

Responda a una solicitud DELETE en la ruta /user:

```
app.delete('/user', function (req, res) {
            res.send('Got a DELETE request at /user');
            });
```

La función tiene 2 parámetros.

El parámetro principal que usaremos es el parámetro ‘res’ que se puede usar para enviar información al cliente.
El parámetro ‘req’ tiene información sobre la solicitud que se está realizando. A veces, se pueden enviar parámetros adicionales como parte de la solicitud que se realiza, y por lo tanto, el parámetro ‘req’ se puede utilizar para encontrar los parámetros adicionales que se envían.

Servicio de archivos estáticos en Express
Para el servicio de archivos estáticos como, por ejemplo, imágenes, archivos CSS y archivos JavaScript, utilice la función de middleware incorporado express.static de Express.

Pase el nombre del directorio que contiene los activos estáticos a la función de middleware express.static para empezar directamente el servicio de los archivos. Por ejemplo, utilice el siguiente código para el servicio de imágenes, archivos CSS y archivos JavaScript en un directorio denominado public:

```
app.use(express.static('public'));
```

Ahora, puede cargar los archivos que hay en el directorio public:

http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html

Express busca los archivos relativos al directorio estático, por lo que el nombre del directorio estático no forma parte del URL.
Para utilizar varios directorios de activos estáticos, invoque la función de middleware express.static varias veces:

```
app.use(express.static('public'));
app.use(express.static('files'));
```

Express busca los archivos en el orden en el que se definen los directorios estáticos con la función de middleware express.static.

Para crear un prefijo de vía de acceso virtual (donde la vía de acceso no existe realmente en el sistema de archivos) para los archivos a los que da servicio la función express.static,

especifique una vía de acceso de montaje para el directorio estático, como se muestra a continuación:

app.use('/static', express.static('public'));

Ahora, puede cargar los archivos que hay en el directorio public desde el prefijo de vía de acceso /static.

http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html

No obstante, la vía de acceso que proporciona a la función express.static es relativa al directorio desde donde inicia el proceso node. Si ejecuta la aplicación Express desde cualquier otro directorio, es más seguro utilizar la vía de acceso absoluta del directorio al que desea dar servicio:

a**pp.use('/static', express.static(__dirname + '/public'));**

Vías de acceso de ruta

Las vías de acceso de ruta, en combinación con un método de solicitud, definen los puntos finales en los que pueden realizarse las solicitudes. Las vías de acceso de ruta pueden ser series, patrones de serie o expresiones regulares.

Estos son algunos ejemplos de vías de acceso de ruta basadas en series.

Esta vía de acceso de ruta coincidirá con las solicitudes a la ruta raíz, /.

```
app.get('/', function (req, res) {
  res.send('root');
});
```

Esta vía de acceso de ruta coincidirá con las solicitudes a /about.

```
app.get('/about', function (req, res) {
  res.send('about');
});
```

Esta vía de acceso de ruta coincidirá con las solicitudes a /random.text.

```
app.get('/random.text', function (req, res) {
  res.send('random.text');
});
```

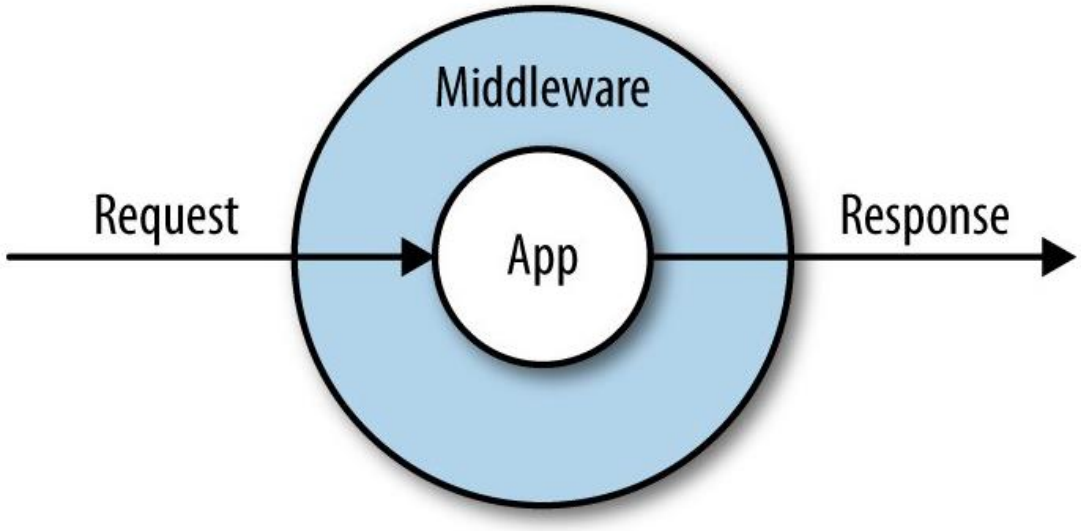
Métodos de respuesta

Los métodos en el objeto de respuesta (res) de la tabla siguiente pueden enviar una respuesta al cliente y terminar el ciclo de solicitud/respuestas. Si ninguno de estos métodos se invoca desde un manejador de rutas, la solicitud de cliente se dejará colgada.

Método	Descripción
res.download()	Solicita un archivo para descargarlo.
res.end()	Finaliza el proceso de respuesta.
res.json()	Envía una respuesta JSON.
res.jsonp()	Envía una respuesta JSON con soporte JSONP.
res.redirect()	Redirecciona una solicitud.
res.render()	Representa una plantilla de vista.
res.send()	Envía una respuesta de varios tipos.
res.sendFile()	Envía un archivo como una secuencia de octetos.
res.sendStatus()	Establece el código de estado de la respuesta y envía su representación de serie como el cuerpo de respuesta.

Fuente:https://expressjs.com/es/guide/routing.html

Middleware



Fuente:https://medium.com/@aarnlpezsosa/middleware-en-express-js-5ef947d668b

Un middleware es una función que se puede ejecutar antes o después del manejo de una ruta. Esta función tiene acceso al objeto Request, Response y la función next().
Las funciones middleware suelen ser utilizadas como mecanismo para verificar niveles de acceso antes de entrar en una ruta, manejo de errores, validación de datos, etc.

Veamos el siguiente ejemplo:

definir una ruta en la cual solo los usuarios administradores pueden ingresar, por lo tanto, se necesita verificar si el usuario es o no, un administrador. Para ello se enviará un parámetro llamado isAdmin, que puede tener dos valores: true o false. El cual indicará si el usuario es administrador (valor true).
primero definir la ruta a la que acceden solo los administradores.como lo muestra la siguiente linea de código:

```
const express = require('express');
const app = express();
const port = 3000;

// Permite recibir parámetros en formato JSON.
app.use(express.json());
```

```
// Ruta a la cual solo deben ingresar usuarios administradores.
app.get('/admind', (req, res) => {
    res.send('bienvenido administrador');
});

app.listen(port, () => {
    console.log(`Server listeting on port ${port}`)
});
```

Una vez creado el servidor, se define la función middleware, recordemos que un middleware tiene acceso al objeto Request, Response y la función next(), por lo tanto la función que se definirá contará con tres parámetros: req, res y next. Los cuales hacen referencia a los objetos mencionados anteriormente.

function isAdmin(req, res, next) {
}

Después, se agregará la validación para comprobar si el usuario es administrador, utilizando el parámetro isAdmin enviado por el cliente, quedando de la siguiente forma:

```
function isAdmin(req, res, next) {
    if (req.body.isAdmin) {
        next();
    } else {
        res.status(403).send(`acceso solo para usuarios
administradores`);
    }
}
```

en la función anterior se puede observar si el valor de req.body.isAdmin es igual a true pasa el control de petición del acceso a la ruta utilizando el parámetro la función next() de caso contrario se envía un mensaje advirtiendo que no tiene el acceso ya que no es un usuario administrador.

en el siguiente fragmento de código se muestra como se debe adicionar la función de middleware en la aplicación:

```
const express = require('express');
const app = express();
const port = 3000;

// Middlewa que verifica si el usuario es un administrador.
function isAdmin(req, res, next) {
    if (req.body.isAdmin) {
        next();
    } else {
        res.status(403).send(`acceso solo para usuarios
administradores`);
    }
}

// Permite recibir parámetros en formato JSON.
app.use(express.json());

// Se agrega el middleware en la aplicación.
app.use(isAdmin);

// Ruta a la cual solo deben ingresar usuarios administradores.
app.get('/admin', (req, res) => {
    res.send('bienvenido administrador');
});


app.listen(port, () => {
    console.log(`Server listeting on port ${port}`)
});
```

Bibliografía

https://expressjs.com/

https://medium.com/@aarnlpezsosa/middleware-en-express-js-5ef947d668b