Aplicaciones Web con Express

Fernando Saez

Instalación de Express

1. Creamos la carpeta de la aplicación y nos movemos a esta
2. Inicializamos la aplicación con npm init
3. Instalamos express añadiéndolo a la lista de dependencias

   npm install express --save
4. Incluimos el módulo en el app.js con require

```
var express = require('express');
var app = express();
 app.get('/', function (req, res) {
  res.send('Hola mundo');
});
 app.listen(3000, function () {
  console.log('App de ejemplo escuchando en puerto 3000!');
});
```

Ejercicio

- npm install express –save
- Cree app.js

```
const express = require('express');
const app = express();
app.get('/ping', (request, response) => {
        response.send('pong');
});
app.listen(8080, 'localhost');
```

- Node app.js
- Ejecute http://localhost:8080/ping

Enrutamiento Básico

app.get (ruta [, middleware], callback[, callback])
app.put (ruta [, middleware], callback[, callback])
app.post (ruta [, middleware], callback[, callback])
app.use (ruta [, middleware], callback[, callback])
app.use (callback)

- Definir rutas

```
app.get('/someUri', function (req, res, next) {})
app.all('/myPath', function (req, res, next, error) {})
```

- Encadenados para una sola vía de acceso de ruta

```
app.route('/myPath')
.get(function (req, res, next) {})
.post(function (req, res, next) {})
.put(function (req, res, next) {})
```

Vías de acceso de ruta

- Las vías de acceso de ruta pueden ser series, patrones de serie o expresiones regulares

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

app.get('/personas?', function(req, res) {
  res.send('ab?cd');
});

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

Obtener Información del Request

GET /user/32135?field=name

```
// obtener path completo
req.originalUrl                 => /user/32135?field=name

console.dir(req.path)           => '/user'

// obtener valores coincidentes con parámetro de la ruta
app.get('/user/:user_id', function (req,res){
req.params.user_id              => 32135

// obtener parámetro de la querystring
console.dir(req.query.field)    => 'name'
```

Otras propiedades de Request

```
req.get('Content-Type’)    //Podemos obtener cualquier encabezado
// => "text/plain“


// https://ocean.example.com/creatures?filter=sharks
app.get('/creatures', (req, res) => {
  console.log(req.protocol)     // "https"
  console.log(req.hostname)     // "example.com"
  console.log(req.path)         // "/creatures"
  console.log(req.originalUrl)  // "/creatures?filter=sharks"
  console.log(req.subdomains)   // "['ocean']"
})
```

Leer datos de Formularios (post)

- Request del tipo application/x-www-form-urlencoded
- Usamos la función urlencoded incluida en express

```
app.use(express.urlencoded());
..
..
let nombre = req.body.nombre || ‘’;
let documento = req.body.documento;
```

Contenido JSON express.json()
Contenido Texto plano express.text()
Contenido Crudo express.raw()

Objeto Response

- Si ninguno de estos métodos se invoca desde un manejador de rutas, la solicitud de cliente se Perderá.

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.

Objeto Response (2)

```
1  // El parámetro de send puede ser un buffer, string, object, array o boolean
   res.send(Buffer.from('Saludosss'))
   res.send({ some: 'json' })
   res.send('<p>Algún párrafo HTML</p>')

2  //Tambien podemos realizar encadenamiento
   res.status(404).send('Sorry, No podemos encontrar este recurso!')
   res.status(500).send({ error: 'algo esta mal' })

3  //configurar cabeceras
   res.set('Content-Type', 'text/html')
   res.send(Buffer.from('<p>some html</p>'))

4  //La salida será un json
   res.send({ user: 'tobi' })
   res.send([1, 2, 3])
```

Objeto Response (3)

1
```
var fileName = req.params.name
 res.sendFile(fileName, options, function (err) {
  if (err) {
    next(err)
  } else {
    console.log('Sent:', fileName)
  }
 })
})
```

2
```
res.redirect('/foo/bar')
res.redirect('http://example.com')
res.redirect(301, 'http://example.com')
res.redirect('../login')
```

API JSON con ExpressJS

```
var express = require('express');
var cors = require('cors'); // Use cors para permitir Cross-origin RS
var app = express();
app.use(cors()); // middleware para todas las rutas
var port = process.env.PORT || 8080; //podemos pasar el puerto desde el entorno
app.get('/', function(req, res) {
        var info = {
        'string_value': 'StackOverflow',
        'number_value': 8476
}
res.status(200).json(info);
})
app.listen(port, function() { //levanta el servidor
console.log('Node.js escuchando en puerto ' + port)
})
```

Obtener JSON del Request Usando express.json

- Supongamos la petición json:

```
PUT /settings/32135
{
"name": “Pedro"
}
```

1

- Podemos usar el middleware body-parser

```
app.use(express.json());
..
req.body.name // “Pedro“
```

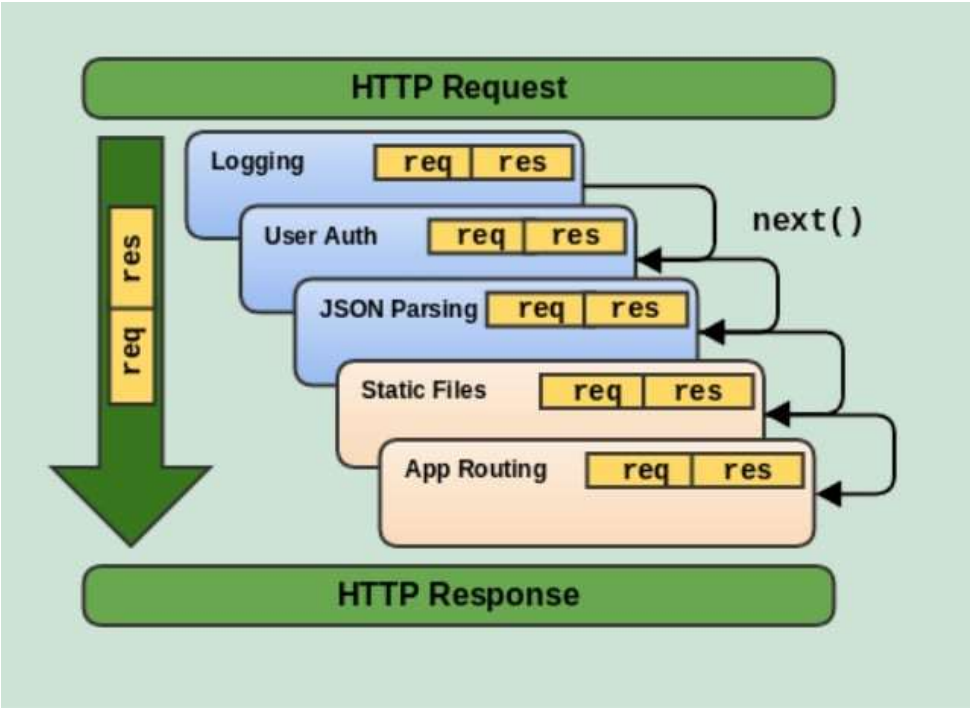
2

- Podemos usar el middleware cookie-parser

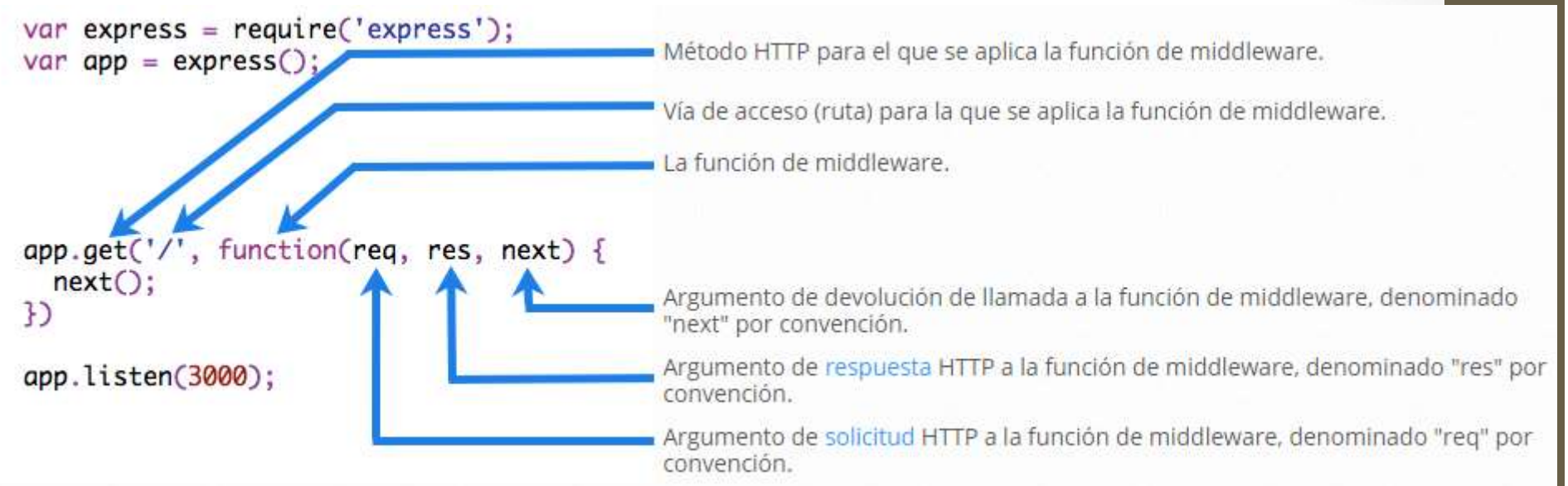
```
var cookieParser = require('cookie-parser')
app.use(cookieParser())
..
req.cookies.name
```

Escritura de middleware

- Las funciones de middleware pueden realizar las siguientes tareas:
  - Ejecutar cualquier código.
  - Realizar cambios en la solicitud y los objetos de respuesta.
  - Finalizar el ciclo de solicitud/respuestas.
  - Invocar el siguiente middleware en la pila.



Escritura de middleware (2)



- Este es un ejemplo simple de una función de middleware denominada “myLogger”. Esta función simplemente imprime “LOGUEADO” cuando una solicitud de la aplicación pasa por ella.

```
var myLogger = function (req, res, next) {
  console.log('LOGUEADO');
  next();
};
```

Escritura de middleware (3)

- Para cargar la función de middleware, llame a app.use(), especificando la función de middleware.

```
var express = require('express');
var app = express();
var myLogger = function (req, res, next) {
  console.log('LOGGED');
  next();
};
app.use(myLogger);
app.get('/', function (req, res) {
  res.send('Hello World!');
});
app.listen(3000);
```

- El orden de carga del middleware es importante: las funciones de middleware que se cargan primero también se ejecutan primero.

Utilización de middleware

- Una aplicación Express puede utilizar los siguientes tipos de middleware:

- Middleware de nivel de aplicación
- Middleware de nivel de direccionador *
- Middleware de manejo de errores
- Middleware incorporado
- Middleware de terceros

- * ver express.Router

Middleware Nivel de Aplicación

- Utiliza las funciones app.use() y app.METHOD()

```
//La función se ejecuta cada vez que la aplicación recibe una solicitud.
var app = express();
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
//se ejecuta para cualquier tipo de solicitud en la vía de acceso /user/:id
app.use('/user/:id', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
```

Middleware de manejo de errores

- El middleware de manejo de errores se define de la misma forma que otro middleware, excepto con cuatro argumentos en lugar de tres; específicamente con la firma (err, req, res, next):

```
app.use(function(err, req, res, next) {
        console.error(err.stack);
        res.status(500).send('Something broke!');
});
```

Middleware Incorporado

- La única función de middleware incorporada en Express es express.static. Esta función es responsable del servicio de recursos estáticos de una aplicación Express.

```
project root
├── server.js
├── package.json
└── public
        ├── index.html
        └── script.js
```

```
const express = require('express');
const app = express();
app.use(express.static('public'));
app.use(express.static('images')); //Ojo con nombres duplicados
app.use(express.static('files'));
```

No hace falta específicar /public/ en la url.

Middleware de terceros

- Instale el módulo Node.js para la funcionalidad necesaria y cárguelo en la aplicación a nivel de aplicación o a nivel de direccionador.

```
$ npm install cookie-parser
```

```
var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');

// cargar el middleware cookie-parsing
app.use(cookieParser());
```

Configurar manejo de 404

- Las respuestas 404 no son el resultado de un error, por lo que el middleware de manejador de errores no las capturará.

```
app.use(function(req, res, next) {
        res.status(404).send('Sorry cant find that!');
});
```

- El middleware de manejo de errores se define al final, después de otras llamadas de rutas y app.use()

```
var bodyParser = require('body-parser');
var methodOverride = require('method-override');

app.use(bodyParser());
app.use(methodOverride());
app.use(function(err, req, res, next) {
  // logic
});
```

Utilización de motores de pug plantilla con Express

- Para que Express pueda representar archivos de plantilla, deben establecerse los siguientes valores de aplicación:
  - views, el directorio donde se encuentran los archivos de plantilla. Ejemplo: app.set('views', './views')
  - view engine, el motor de plantilla que se utiliza. Ejemplo: app.set('view engine', 'pug')

$ npm install pug –save

Index.pug

```
html
  head
    title= title
  body
    h1= message
```

```
app.get('/', function (req, res) {
  res.render('index', { title: 'Hey', message: 'Hello there!'});
});
```