



Express

© 2017, ACTIBYTI PROJECT SLU, Barcelona
Autor: Ricardo Ahumada

ÍNDICE DE CONTENIDOS

1. Introducción
2. Request y Response con Express
3. Middlewares
4. Templates
5. Middleware personalizados
6. SimpleAuth

1

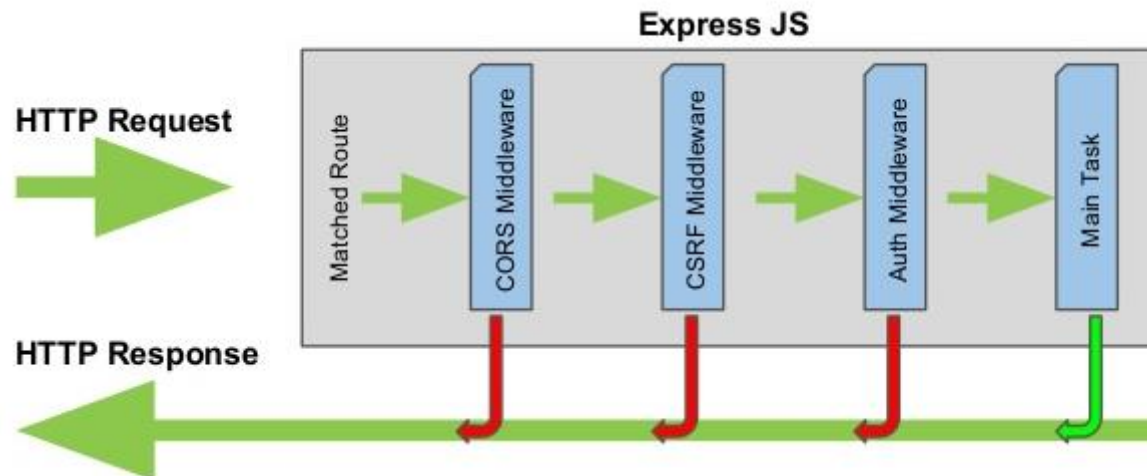
Introducción

¿Qué es express?

express

- Un framework web para Node.js
 - › Estrictamente web (microframework)
 - › Sencillo y flexible
 - › Muy popular
 - › Se adapta muy bien a la filosofía de Node
 - › <http://expressjs.com>
- Express nos va ayudar con...
 - › Rutas
 - › Parámetros
 - › Formularios y subida de ficheros
 - › Cookies
 - › Sesiones
 - › Templates
- API:
 - › <http://expressjs.com/en/api.html>

- Construye sobre http
- Procesando la petición por un **stack de middleware** que se encarga de decorar las peticiones
 - Asocia rutas a manejadores
 - Decorar los objetos req y res (parseo de parámetros, multipart, etc,...)
 - Rendeear templates
- Nosotros escogemos qué middlewares queremos usar, y en qué orden





Instalación y uso

- Para instalar Express en el proyecto
 - `npm install express -save`
- Uso:

```
var express = require("express");
```

```
var app = express();
```

```
// configuración + rutas
```

```
app.listen(3000);
```

Ejemplo de uso



```
var app = require("express")();  
  
app.get("/", function(req, res) {  
    res.end("Hola desde express!");  
});  
  
app.listen(3000);!
```

2

Request y Response con Express

Request

- req.params: parámetros de la ruta

```
app.get("/user/:id", function(req, res) {  
    req.params.id;  
});
```

- req.query: la querystring, parseada

```
app.get("/search", function(req, res) {  
    // GET /search?text=nodejs+express  
    req.query.text;  
});
```

Request

- req.ip: IP del cliente conectado
- req.host: Hostname del servidor
- req.xhr: ¿Es ajax?
- req.acceptedLanguages: Array de locales

Response

- `res.cookie(nombre, valor, [opciones])`
 - Modifica la cookie “nombre” con el valor “valor”

```
res.cookie("visitas", "1", {domain: ".ejemplo.com"});
```

- `res.redirect([status], url)`
 - Redirige a url
 - El código de estado es opcional (302 por defecto)

```
res.redirect(301, "http://www.google.com");
```

Response

- `res.send(body)`
 - › Envía una respuesta (escribe en el buffer)
 - › Lo más adecuado para respuestas sencillas
 - › no streaming
 - › Automatiza ciertas cabeceras
 - › Content-Type, Content-Length
 - › Convierte objetos a JSON

```
res.status(500).send({msg: "Oh, oh..."});
```

Response

- Muchos otros métodos auxiliares:
 - Cabeceras
 - Envío de ficheros
 - JSONP
 - Content-Type

3

Middlewares

Configurar la aplicación

- Crear algunas propiedades globales
- Especificar el stack de **middleware**

```
app.set(prop, value) / app.get(prop)
```

- Escribir/consultar valores globales en app
 - Básicamente inútil...
 - Excepto para cambiar alguna configuración más avanzada

```
app.set('title', 'Mi App');  
app.get('title');!
```

Middleware

- Middleware son módulos “plug and play” que se pueden apilar arbitrariamente en cualquier orden y proveen cierta funcionalidad
 - Filtros: procesan tráfico entrante/saliente, pero no responden a ninguna request. (ejemplo: bodyParser)
 - Proveedores: ofrecen respuestas automáticas a algún tipo de petición (ejemplo: static provider)

```
app.use(favicon());  
app.use(logger('dev'));  
app.use(bodyParser());  
  
app.use(cookieParser('your secret here'));  
  
app.use(express.static(__dirname + '/public'));
```

- Express trae unos cuantos preinstalados
 - <http://www.senchalabs.org/connect/>
- Lista de módulos de terceros
 - <https://github.com/senchalabs/connect/wiki>

favicon(ruta)

- Sirve el favicon de la aplicación
- Debe ser el primero
- Para evitar capas innecesarias
 - › log
 - › parseo
 - › cookies
 - › etc...

logger([opciones])

- Registro de actividad
- Muchas opciones:
<http://www.senchalabs.org/connect/logger.html>
- Se suele poner debajo de `express.favicon()`

cookieParser([secret])

- Parsea las cookies de la petición
- Opcional: firmar cookies con secret
- Crea el objeto req.cookies

```
app.use(cookieParser('secreto'));  
app.get("/", function(req, res) {  
    console.log(req.cookies);  
    res.send(200);  
})
```

bodyParser()

- Parsea el cuerpo de las peticiones POST
- Decodifica
 - application/json
 - application/x-www-form-urlencoded
 - multipart/form-data
- Crea el objeto req.body con los parámetros POST
- Crea el objeto req.files con los ficheros que se han subido desde un formulario

cookieSession([opciones])

- Inicializa y parsea los datos de sesión del usuario
- Crea el objeto **req.session**
- Utilizando cookies como almacenamiento
- Opciones:
 - › **secret**: firma de seguridad para la cookie
 - › **maxAge**: duración, en ms (default: sin caducidad)
 - › **path**: ruta para la que es válida la cookie (default: /)
 - › **httpOnly**: protegida del cliente (default: true)

cookieSession([opciones])

```
var express = require("express"),
    app = express();

app.use(express.cookieParser());
app.use(express.cookieSession({secret: "secreto"}));

app.get("/", function(req, res) {
    req.session.visitas || (req.session.visitas = 0);
    var n = req.session.visitas++;
    res.send("Me has visitado: " + n + " veces!");
})

app.listen(3000);
```

express.static(dir)

- Sirve los ficheros estáticos dentro de **dir**
- ¡Muy útil!
- Se pone cerca del final
- Cachea los ficheros
- La variable global `__dirname` contiene el
- directorio donde reside el script en ejecución

app.param(...)

- Mapear parámetros de url

```
app.param("postId", function(req, res, next, postId) {  
  req.post = Post.find(postId);  
  if (!req.post) {  
    next(new Error("Post no encontrado (" + postId + ")"));  
  } else {  
    next();  
  }  
});  
  
app.get("/posts/:postId", function(req, res) {  
  console.log(req.post);  
});
```


4

Templates

Renderizado de templates

- Express tiene un mecanismo para renderizar templates
- Agnóstico
- Modular
- Simple
- NO trae ningún motor de templates por defecto

```
res.render(view, [locals], callback)
```

- view: ruta del template
- locals: valores a interpolar
- callback: function(err, html) { ... }
- Tenemos muchos motores de templates para elegir
 - <https://npmjs.org/browse/keyword/template>
- Haml, Hogan/Mustache, Twig/Swig, **Ejs**, Jinja, **Jade**, ...

Ejemplo de templates

➤ Plantilla EJS

```
<h1><%= title %></h1>
<ul>
<% for(var i=0; i<supplies.length; i++) { %>
<li>
<a href='supplies/<%= supplies[i] %>'>
<%= supplies[i] %>
</a>
</li>
<% } %>
</ul>
```

➤ Plantilla Jade

```
doctype 5
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
        bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
    p.
      Jade is a terse and simple
      templating language with a
      strong focus on performance
      and powerful features.
```

Ejemplo de Renderizado de templates

➤ app.js

```
var express = require("express"),
    app = express();

app.set("views", "./views");
app.set("view engine", "jade");

app.get("/", function(req, res) {
  res.render("welcome", {user: "Pepito"});
})

app.listen(3000)
```

➤ /views/welcome.jade

```
doctype html!
html(lang="es")!
body!
h1 Bienvenido, #{user}
```



Creando un blog

- Vamos a hacer un “blog”, simplificado
- Sin BBDD (Guarda los datos en memoria, en un array)
- Tiene 7 rutas (4 pantallas)
 - › **GET /posts**
 - › **GET /posts/new**
 - › **POST /posts**
 - › **GET /posts/:id**
 - › **GET /posts/:id/edit**
 - › **PUT /posts/:id**
 - › **DEL /posts/:id**

5

Middleware personalizado

Creando un

- Escribir middleware para express es muy sencillo
- Una función que recibe tres parámetros:
 - req
 - res
 - next
- Al terminar su tarea, tiene que invocar a next()
 - Sin parámetro: se invoca al siguiente middleware del stack
 - Con parámetro: se cambia la ruta a lo que se pase como parámetro

Ejemplo: un log simple

```
app.use(function(req, res, next) {  
  console.log(" * %s: %s %s",  
    req.connection.remoteAddress,  
    req.method,  
    req.url);  
  next();  
});
```


Gestión de Errores

- Un caso especial de middleware: una función que reciba un parámetro más

```
var express = require("express"),
    app = express();

app.get("/", function(req, res) {
    throw new Error("??")
})

app.use(function(err, req, res, next) {
    console.log("* SOCORRO! ALGO VA MAL! ", err);
    res.send(500)
})

app.listen(3000)
```

Gestión de Errores

➤ Sería más correcto así

```
var express = require("express"),
    app = express();

app.get("/", function(req, res, next) {
    next(new Error("esta app no hace nada"));
})

app.use(function(err, req, res, next) {
    console.log("* SOCORRO! ALGO VA MAL! ", err);
    res.send(500)
})

app.listen(3000)
```

Middleware locales

- Dos maneras de activar middlewares
 - Globalmente (app.use), activos para toda la app
 - Locales para ciertas rutas

```
var express = require("express"),
    app = express()

function logThis(req, res, next) {
  console.log("-> ", req.url);
  next();
}

app.get("/", logThis, function(req, res) {
  res.send("Ok!");
})

app.listen(3000);
```

Middleware local: Ventajas

- Afinar la funcionalidad de cada ruta
 - › Pre-procesado de URL (parámetros, formato)
 - › Seguridad
 - › Caché
 - › Métricas

6

SimpleAuth

simpleAuth.js

- Módulo para autenticar usuarios
 - › Sencillo y flexible
 - › Independiente de la BBDD
 - › Utilizando las sesiones de express
 - › Middleware de ruta
- El módulo provee 3 middlewares:
- `createSession`: comprueba los credenciales de usuario y crea una nueva sesión si son correctos
- `requiresSession`: protege una ruta y solo deja pasar a usuarios autenticados
- `destroySession`: desloguea a un usuario

SimpleAuth

```
var auth = require("./simpleAuth");

app.post("/login", auth.createSession({ redirect: "/secret" }))

app.get("/secret", auth.requiresSession, function(req, res) {
    res.end("Hola, " + req.user.email);
})

app.get("/logout", auth.destroySession, function(req, res) {
    res.end("Te has deslogueado");
});
```

Serializar - deserializar

- El usuario ha de definir una estrategia:
 - serializeUser: ¿Cómo serializar un usuario?
 - deserializeUser: ¿Cómo des-serializar un usuario?
 - checkCredentials: ¿Son correctos los credenciales?

Serializer - deserializar

```
var auth = require("./simpleAuth")!

var users = [{email: "admin@asdf.com", pass: "asdf", id: 0}];

auth.setStrategy({
  serializeUser: function(user) {
    return user.id;
  },
  deserializeUser: function(userId, cb) {
    cb(users[userId]);
  },
  checkCredentials: function(email, pass, cb) {
    var admin = users[0];
    if (email === admin.email && pass === admin.pass) {
      cb(null, admin);
    } else {
      cb(null, false);
    }
  }
});
```

auth.createSession(config)

- Genera una función que...
 1. Extrae username y password según config de req.body
 2. Llama a strategy.checkCredentials con username y password
 3. Si son credenciales correctos:
 - 3.1. Crea una entrada en la sesión o cookie con el resultado de llamar a strategy.serializeUser(usuario), donde usuario es lo que devuelve strategy.checkCredentials
 - 3.2. Redirige a la URL de éxito
 4. Si **no** son correctos:
 1. Redirige a la URL de error
- config.username: nombre del campo del formulario
- config.password: nombre del campo del formulario
- config.redirect: URL en caso de éxito
- config.failRedirect: URL en caso de error
 - default: /login
- Devuelve: una función para usar como middleware

auth.requiresSession

1. Se asegura de que exista la entrada adecuada en la sesión o cookie
2. Llama a `strategy.deserializeUser` con el valor
3. Guarda el usuario des-serializado en `req.user`
4. En caso de error, borra la sesión o cookie

auth.destroySession

- Borra la entrada adecuada en la sesión o cookie

auth.setStrategy(strategy)

- Configura la estrategia:
 - › strategy.serializeUser
 - › strategy.deserializeUser
 - › strategy.checkCredentials
 - › strategy.loginRoute



Proteginedo el blog

- Protege la edición del blog con simpleAuth
 - Utilizando un array de usuarios escrito directamente en el código
 - Añade una página de login y un link para logout
 - Solo los usuarios logueados pueden crear o editar posts
 - Si te sobra tiempo, haz que los usuarios se puedan registrar



 **netmind**

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es



MINISTERIO
DE ENERGÍA, TURISMO
Y AGENDA DIGITAL

red.es



UNIÓN EUROPEA

Fondo Social Europeo
"El FSE invierte en tu futuro"