Express.js

Javier Miguel @JavierMiguelG jamg44@gmail.com CTO & Freelance Developer

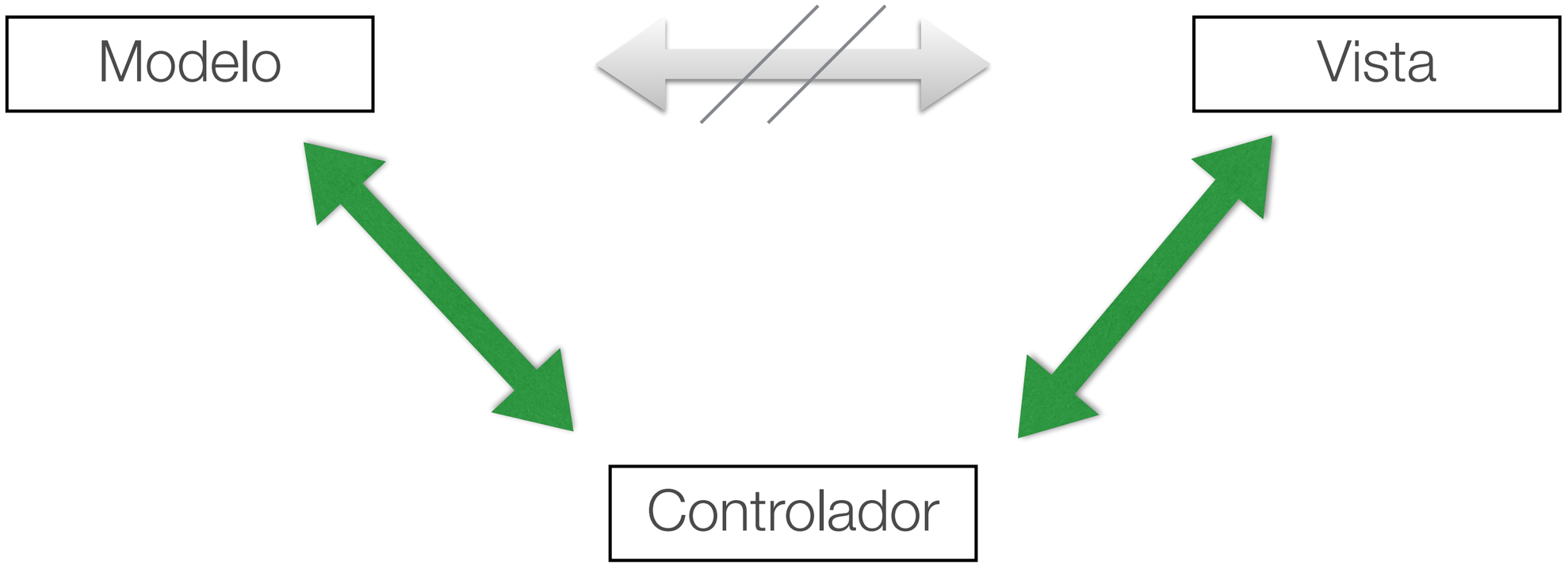
Estructurar nuestra aplicación

Estructura

En Node.js podemos usar cualquier patrón para estructurar nuestro código.

El patrón MVC es comúnmente usado por muchos desarrolladores por su buen resultado.

MVC



MVC

- El controlador recoge datos del modelo y los da a la vista para que los represente en su interfaz
- La vista recibe las acciones del usuario y da información al controlador que opcionalmente guardará lo necesario en el modelo
- El modelo avisará el controlador de que hay nuevas versiones de los datos, y este opcionalmente las entregará a la vista

Express

Express

Express es un framework web para Node.js

http://expressjs.com/

Web Frameworks

Existen múltiples frameworks web para node.js y surgen nuevos con frecuencia, por ejemplo:

- Express.js
- Koa
- Hapi
- Restify
- …

Muchos de ellos extienden la funcionalidad de Express.js, siendo este el más usado.

Express

Podemos revisar toda la funcionalidad en:

- http://expressjs.com/api.html

Ejercicio

Una app básica con Express

Usando Express Generator

Usando Express Generator

Express Generator nos crea una estructura **base** para una aplicación.

```
$ [sudo] npm install express-generator -g

$ express -h

$ express <nombreApp> [--ejs]

$ cd <nombreApp>
$ npm install
```

ejemplos/express/generated

Express Generator

Como arrancar nuestra aplicación:

```
$ npm start

// entorno desarrollo, puerto por defecto (3000)
```

Express Generator

Podemos establecer variables de entorno para variar la forma de arranque:

```
$ NODE_ENV=production npm start


// entorno producción
```

Express Generator

Podemos establecer variables de entorno para variar la forma de arranque:

```
$ DEBUG=nombreApp:* PORT=3001 NODE_ENV=production npm start


// con log debug activado
// puerto 3001
// entorno producción
```

Express Generator

Si queremos podríamos incluir esto en el comando start de npm, especificándolo en el package.json

```
npm install --save-dev nodemon cross-env
```

```
...
  "scripts": {
    "dev": "cross-env DEBUG=nombreApp:* PORT=3000 nodemon"
  },
...
```

Middlewares

Middlewares

Un Middleware es un handler que se activa ante unas determinadas peticiones o todas.

Debe responder o llamar next().

Podemos poner tantos middlewares como nos hagan falta.

Los middleware’s usan callbacks por defecto.

Middlewares

Tiene acceso al objeto de la petición (request), al objeto de la respuesta (response) y al siguiente middleware (`next`).

Si un middleware no quiere responder en una llamada debe llamar al siguiente con `next()` para pasarle el control, de lo contrario la petición quedará sin respuesta, y el cliente que la hizo (por ejemplo un browser) se quedará esperando hasta su tiempo límite de time-out.

Middlewares - app

Se conecta al objeto instancia de la aplicación (app) con app.use o app.METHOD, por ejemplo:

```
var app = express();

app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

Middlewares - router

Se conecta a un router con router.use o router.METHOD, por ejemplo:

```
var router = express.Router();

router.use(function (req, res, next) {
  req.user = userModel.find(req.body.userId);
  next();
});
```

Middlewares - error

Los pondremos los últimos, tras todas nuestras rutas.
Reciben un parámetro más err.

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

// en una ruta anterior podemos haber hecho next(err);
```

Podemos devolver lo que nos convenga, un JSON con un error, una página de error, un texto, etc.

Rutas

Rutas

Express define las rutas para tengan la siguiente estructura:

```
app.METHOD(PATH, HANDLER)
```

donde:

- app es la instancia de express
- METHOD es el método de la petición HTTP (GET, POST, …)
- PATH es la ruta de la petición
- HANDLER es la función que se ejecuta si la ruta coincide.

Rutas

HTTP pone a nuestra disposición varios métodos, de los cuales generalmente hacemos distintos usos:

- **GET** para pedir datos, es idempotente (p.e. listas)
- **POST** para crear un recurso (p.e. crear un usuario)
- **PUT** para actualizar, es idempotente (p.e. guardar un usuario existente)
- **DELETE** eliminar un recurso, es idempotente (p.e. eliminar un usuario)

* idempotente: si lo ejecutas varias veces los resultados no cambian

Rutas

Por lo tanto podríamos construir rutas como estas:

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

app.post('/', function (req, res) {
  res.send('Guardado!');
});
```

Orden de las rutas

El orden es importante!

En el orden que carguemos nuestras rutas a express es el orden en que las interpretará.

Si ponemos los estáticos después de nuestras rutas podremos '*sobre-escribir*' un fichero estático con una ruta, por ejemplo para comprobar si el usuario tiene permisos para descargarlo.

Rutas

Express nos permite también usar *all* como comodín.

```
app.all('/admin', function (req, res) {
  console.log('Accediendo a sección admin ...');
  res.send('Admin zone');
});
```

Rutas

```
app.all('/admin', function (req, res, next) {
  //verificar credenciales
  next(); // pasa el control al siguiente handler
});
```

next pasa la ejecución al siguiente handler definido para esa ruta.

Servir ficheros estáticos

Servir ficheros estáticos

Servir estáticos como CSS, imágenes, ficheros javascript, etc, se especifica con un middleware llamado `express.static`

```
app.use( express.static( path.join(__dirname, 'public') ) );
```

Con esto serviremos lo que haya en la carpeta public como estáticos de la raíz de la ruta.

Servir ficheros estáticos

Si queremos añadir otras carpetas de estáticos tenemos que especificar en que rutas colgarlos.

```
// la ruta virtual '/otros' servirá la carpeta '/otros'
app.use('/otros', express.static(path.join(__dirname, 'otros')));
```

Template engines

Templates

Además de servir html estático podemos usar sistemas de plantillas.

Express-generator por defecto monta Jade, podemos cambiarlo fácilmente, por ejemplo a EJS que es uno de los más usados.

Templates

Hay muchos sistemas de templates en Javascript, pero los que cumplen con el estándar de Express están listados en:

https://www.npmjs.com/package/consolidate#supported-template-engines

Templates

Para instalar un sistema de templates lo instalaremos con **`npm install ejs –save`** y nos aseguramos de que nuestra aplicación lo usa con un par de settings:

- **views**, el directorio donde estarán nuestras plantillas, por ejemplo: `app.set('views', './views')`
- **view engine**, el template engine a usar, por ejemplo: `app.set('view engine', 'jade')`

Express lo carga automáticamente y ya podremos usarlo.

Templates - Jade

```
app.set('view engine', 'jade');
```

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    p esto es un párrafo
```

ejemplos/express/jade

Templates - Jade

```
<!DOCTYPE html>
<html>
  <head>
    <title>Express</title>
    <link rel="stylesheet" href="/stylesheets/style.css">
  </head>
  <body>
    <p>esto es un párrafo</p>
  </body>
</html>
```

ejemplos/express/jade

Templates - EJS

EJS añade su funcionalidad sobre HTML estándar.

Esto puede hacer que nos sea más fácil depurar errores o integrar y mantener una maquetación realizada por un maquetador especializado.

Templates

Para proporcionar variables a las vistas tenemos 3 opciones

Templates

Variables globales a toda la app

```
app.locals.titulo = 'Anuncios';

...

res.render('index');
```

Templates

Variables locales a la respuesta

```
res.locals.titulo = 'Anuncios';

res.render('index');
```

Templates

Variables locales a la respuesta

```
res.render('index', { titulo: 'Anuncios' });
```

Templates

En la vista tendremos disponibles esas variables.

```
<title><%= titulo %></title>
```

Templates - sin escapar

El valor será escapado para evitar la inyección de código. Si queremos incluir html usaríamos <%- %>

```
<p><%- sinEscapar %></p>
```

Templates - include

Podemos incluir el contenido de otras plantillas.

```
<% include otra/plantilla %>
```

```
views
  - index.js
  - otra
    - plantilla.ejs
```

Templates - condiciones

Sacar bloques de forma condicional es sencillo.

```
<% if (condicion.estado) { %>
    <p><%= condicion.segundo %> es par</p>
<% } else { %>
    <p><%= condicion.segundo %> es impar</p>
<% } %>
```

Templates - iterar

O por ejemplo iterar bucles.

```
<% users.forEach(function(user){ %>
     <p><%= user.name %></p>
<% }) %>
```

Templates - código

En resumen, entre los tags <% y %> (sin usar '=' o '-') podemos colocar cualquier estructura de código válida en Javascript.

Pero sin pasarnos. Esto es una vista, y meter código aquí significa que tenemos funcionalidad en las vistas.

Es recomendable tener toda o la mayor parte de la funcionalidad en los modelos (o al menos en los controladores).

Templates - html

Si quisiéramos, podríamos tener las vistas con extensión .html

```
app.set('view engine', 'html');
app.engine('html', require('ejs').__express);
```

Templates

Podemos encontrar su documentación y más ejemplos en:

https://github.com/mde/ejs

Recibiendo parámetros

Recibiendo parámetros

Habitualmente recibiremos parámetros en nuestros controladores de varias formas:

- En la ruta (/users/5)
- Con parámetros en query string (/users?sort=name)
- En el cuerpo de la petición (POST y PUT generalmente)
- También podemos recibirlos en la cabecera, pero esta zona solemos dejarla para información de contexto, como autenticación, formatos, etc.

Recibiendo parámetros - en la ruta

Lo definimos en el argumento PATH de la ruta

```
router.put('/ruta/:id', function(req, res) {
  console.log('params', req.params);
  var id = req.params.id;
});
```

PUT http://localhost:3000/apiv1/anuncios/55

Podemos combinarlo con los otros

Recibiendo parámetros

En el paso por ruta podemos usar expresiones regulares en los parámetros, incluir varios o hacerlos opcionales.

```
router.put('/ruta/:id?', function); // parámetro opcional
// params { id: 'dato'}

router.put('/ruta/:id([0-9]+)', function); // parámetro con regexp
// params { id: '26'}

router.put('/ruta/:id([0-9]+)/piso/:piso(A|B|C)', function); // varios
// params { id: '26', piso: 'A' }
```

Recibiendo parámetros - en query string

Lo definimos en la query string

```
router.put('/ruta', function(req, res) {
  console.log('query-string', req.query);
  var id = req.query.id;
});
```

PUT http://localhost:3000/apiv1/anuncios?id=66

Podemos combinarlo con los otros

Recibiendo parámetros - en el body

Los recibimos en req.body. Esta forma no la podemos usar en GET ya que no usa body.

```
router.put('/ruta', function(req, res) {
  console.log('body', req.body);  // body
  var nombre = req.body.nombre;
});
```

PUT http://localhost:3000/apiv1/anuncios
{nombre: 'Pepe'}

Validaciones

Validaciones

https://github.com/ctavan/express-validator

```
npm install express-validator
```

Validaciones

En el middleware:

```
const { query, validationResult } = require('express-validator');

...

router.get('/', [
  query('age').isNumeric().withMessage('must be numeric')
], (req, res, next) => {
  validationResult(req).throw(); // excepción si hay errores de validación

  // todo validado!
  res.send('ok');
});
```

Validaciones

En app.js:

```
// error handler
app.use(function(err, req, res, next) {

  if (err.array) { // validation error
    err.status = 422;
    const errInfo = err.array({ onlyFirstError: true })[0];
    err.message = `Not valid - ${errInfo.param} ${errInfo.msg}`;
  }

  res.status(err.status || 500);
  ...
```

Validaciones

Express.js middleware for validating requests against JSON schema:

https://github.com/vacekj/express-json-validator-middleware

Métodos de respuesta

Métodos de respuesta

res.download() Prompt a file to be downloaded.
res.end() End the response process.
res.json() Send a JSON response.
res.jsonp() Send a JSON response with JSONP support.
res.redirect() Redirect a request.
res.render() Render a view template.
res.send() Send a response of various types.
res.sendFile Send a file as an octet stream.
res.sendStatus() Set the response status code and send its string representation as the response body.

Podemos ver su documentación. Veamos los más usados…

Métodos de respuesta - send

Para responder a una petición podemos usar el método genérico res.send().

El cuerpo de la respuesta puede ser un buffer, un string, un objeto o un array.

```
res.send(new Buffer('whoop'));
res.send({ some: 'json' });
res.send('<p>some html</p>');
res.status(404).send('Sorry, we cannot find that!');
res.status(500).send({ error: 'something blew up' });
```

Express detecta el tipo de contenido y pone el header Content-Type adecuado.

Si es un array o un objeto devuelve su representación en JSON

Métodos de respuesta - json

Podemos usar res.json, que ajusta un posible null o undefined para que salga bien en JSON

```
res.json(null)
res.json({ user: 'tobi' })
res.status(500).json({ error: 'message' })
```

Métodos de respuesta - download

Transfiere el fichero especificado como un attachment. El browser debe solicitar al usuario que guarde el fichero.

```
// el nombre del fichero es opcional

res.download('/report-12345.pdf', 'report.pdf');
```

Métodos de respuesta - redirect

Devuelve una redirección con el status code 302 por defecto (found).

```
res.redirect('/foo/bar'); // relativa al root host name
res.redirect('http://example.com'); // absoluta
res.redirect(301, 'http://example.com'); // con status
res.redirect('../login'); // relativa al path actual
res.redirect('back'); // vuelve al referer
```

Métodos de respuesta - render

Renderiza una vista y envia el HTML resultante. Acepta un parámetro opcional 'locals' para dar variables locales a la vista.

```
// render de la vista index
res.render('index');

// render de la vista user con el objeto locals
res.render('user', { name: 'Tobi' });
```

Métodos de respuesta - sendFile

Envía un fichero como si fuera un estático.

Además de la ruta del fichero acepta un objeto de opciones y un callback para comprobar el resultado de la transmisión.

```
var options = {
    headers: {
          'x-timestamp': Date.now(),
          'x-sent': true
      }
  };

res.sendFile(fileName, options);
```

Middlewares de terceros

Podemos instalarlos con npm y cargarlos como los anteriores.

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

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

// load the cookie parsing middleware
app.use(cookieParser());
```

Hay una lista de los más usados en http://expressjs.com/resources/middleware.html