Ejercicios Tema 6

Nuestro proyecto sigue creciendo, y aunque todavía le queda mucho para estar en producción y hablar de problemas de optimización o de carga todavía nos queda lejos… hay mucho trabajo que podemos ir avanzando.

Ejercicio 1

En este ejercicio vamos a realizar algunas tareas de optimización a nuestro proyecto de ejemplo.

Primero vamos a instalar y configurar la compresión gzip.

```
$ sudo npm install compression
```

Ahora solo queda llamarlo en app.js:

```
//app.js
//———————————

var express = require('express');
//…
var compression = require('compression');
//…
app.use(compression());
app.use(express.static(path.join(__dirname, 'public')));
```

Ejercicio 2

Ahora vamos a instalar `grunt-cli`, la interfaz de linea de comandos de Grunt, lo haremos con `npm`:

```
$ sudo npm install grunt-cli --save-dev
```

Podemos comprobar el correcto funcionamiento de nuestra aplicación antes de utilizar Grunt, para ello primero debemos instalar todas las dependencias con `npm`:

```
$ sudo npm install
```

Una vez se haya instalado todo correctamente, podemos lanzar nuestro proyecto (`$npm start`) para comprobar que todo sigue funcionando.

Una vez que comprobamos que el servidor funciona, vamos a añadir dos tareas automáticas que serán:

> **cssmin**: Se encarga de eliminar los espacios en ficheros css que no son necesarios, reduciendo así su tamaño y por lo tanto el tiempo de carga.
>
> **jshint**: Comprueba los ficheros JavaScript en busca de errores.

Para utilizarlos necesitamos crear el fichero Gruntfile.js en la raíz de nuestro proyecto, el esqueleto básico será como el que sigue más adelante

Dentro del este esqueleto básico pondremos la configuración correspondiente a jshint y a cssmin, cargaremos los contribs de cada tarea, y las registraremos.

```
//Gruntfile.js
//——————————————

module.exports = function(grunt) {

  grunt.initConfig({
    pkg: grunt.file.readJSON("package.json"),
    //configuración de jshint

    //configuración de cssmin
  });

  //cargamos los contrib de cada tarea
  grunt.loadNpmTasks('grunt-contrib-cssmin');
  grunt.loadNpmTasks('grunt-contrib-jshint');

  //registramos las tareas
  grunt.registerTask("default", ["cssmin", "jshint"]);
};
```

Ejercicio 3

La configuración de `jshint` se basa en diferentes opciones que podemos especificar para evaluar nuestros scripts .

La configuración de `jshint` en Gruntfile será (no olvides cambiar las rutas de los scripts por las tuyas):

```
//Gruntfile.js
//————————————

//…

//configuración de jshint
jshint: {
     // definimos los ficheros a analizar
     files: ['public/javascripts/segundaPrueba.js',
'public/javascripts/test.js'],
     // configuramos JSHint
     options: {
     // opciones generales
     globals: {
          jQuery: true,
          console: true,
          module: true
          }
     }
},
//...
```

En este caso hemos especificado dos ficheros sobre los cuales vamos a revisar si el código javascript es correcto. Para poder comprobar si funciona la tarea podemos modificar el código javascript que insertemos dentro de los scripts para forzar los errores.

Extra: no vamos a utilizar angularjs,ni ningún otro framework de frontend basado en js, pero si lo has hecho anteriormente sabrás que el número de librerías puede dispararse. Validar el contenido de nuestros js se convierte en una tarea imprescindible.

Ejercicio 4

Seguramente te diste cuenta desde el primer momento, pero tenemos no uno… si no dos archivos de CSS. Y le podemos sumar el de font-awsome, y nada nos invita a pensar que en un futuro cercano sean menos…

Después de una interminable reunión con el equipo de frontend se ha decidido (impuesto) el criterio de que no es bueno meter mano en el css de Bootstrap, por si hay que actualizar el día de mañana (algo seguro, es una alpha), y que a su vez el contenido de personalización es necesario. Además tenemos en mente un slidder de un tercero y… y sabes por experiencia que andar preocupándose en que el equipo de diseño te proporcione los archivos minificados carece de sentido a estas alturas. Que además te los pasasen todos juntos ya sería demasiado pedir.

¿Qué hacer? ¿Cargar bootstrap por CDN es una opción… pero, y el resto? ¿Es buena idea hacerle cargar a alguien 5 archivos css sin comprimir?

Automatizar todos estos problemas con cssmin.

La configuración de cssmin es mucho más sencilla, únicamente indicaremos los ficheros de entrada y el fichero de salida de la siguiente forma:

```
//Gruntfile.js
//————————————

//…

//configuración de cssmin

cssmin: {
  options: {
    shorthandCompacting: false,
    roundingPrecision: -1
  },
  target: {
    files: {
      'output.css': ['foo.css', 'bar.css']
    }
  }
}
//…
```

De paso revisa las vistas de tu proyecto, y pasa todo el css que tengas repartido en etiquetas <style> a personalizacion.css (asegurándote de que sigue funcionando correctamente).

Ejercicio 5
Ya hemos creado nuestro fichero Gruntfile.js con la configuración necesaria para optimizar nuestra app.

Antes de probar nuestras tareas, debemos instalar los módulos que vamos a usar si no lo has hecho todavía:

```
$ npm install grunt-contrib-jshint –save-dev

$ npm install grunt-contrib-cssmin ––save-dev
```

Ha llegado el momento de probar que ejecuta correctamente los plugins. Para ello podemos hacerlo con el comando grunt:

```
$ grunt
```

Deberá producir una salida como la siguiente:

```
Running "cssmin:combine" (cssmin) task

Running "jshint:bootstrap" (jshint) task

✔ No problems


Done, without errors.
```

Una vez pasado el grunt deberás de enlazar en tus vistas al css minificado que tengamos como resultado.

Ejercicio 6

Tienes miles de plugins disponibles en gruntjs. Explora entre todas las posibilidades y monta una tarea de ejemplo que puedas necesitar el día de mañana.

Instala y configura otro plugin que consideres interesante.

Plugins destacados:

uglify: permite comprimir, parsear etc js.

Sass y Less: compilar archivos SAS y Less a CSS

Yuidoc: genera documentación a partir de comentarios en Js

Changelog: generador automático de un “changelog” a partir de los commits en git.

