Tema 2

Express y Swig

¿Qué es Express?

- Express según sus creadores es “un framework para Node.js minimalista y flexible”. Es el framework de Node.js más famoso y utilizado por la sensación de sencillez que genera.

- Express oculta las tareas más difíciles o tediosas de forma que no necesites saber siquiera como funciona, simplemente hace lo que quieres que haga.

¿Qué es Express?

- Express incluye las siguientes características:
    - Crea toda las estructura de carpetas y ficheros de un proyecto.
    - Gestionar las peticiones y las rutas.
    - Da soporte para frameworks de creación de interfaces gráficas como pueden ser Jade y Swig.
    - Acepta diferentes tipos de bases de datos.
    - Permite depuración de la aplicación.

Instalar Express

- Para instalar express utilizaremos el gestor de paquetes **npm** que vimos en el tema anterior, para instalarlo globalmente, utilizaremos el siguiente comando:

```
sudo npm install -g express
sudo npm install -g express-generator
```

- Si únicamente lo necesitamos para un proyecto podemos hacerlo con el comando:

```
sudo npm install express
```

Generación de proyectos

- La primera característica que encontramos en Express es la generación de proyectos. Crea automáticamente una jerarquía de carpetas y ficheros:

```
express node_app
```

- Podemos comprobar la generación que ha realizado en la salida de la consola:

```
create : node_app
create : node_app/package.json
create : node_app/app.js
create : node_app/public
create : node_app/public/javascripts
create : node_app/public/images
create : node_app/public/stylesheets
create : node_app/public/stylesheets/style.css
create : node_app/routes
create : node_app/routes/index.js
create : node_app/routes/users.js
create : node_app/views
create : node_app/views/index.jade
create : node_app/views/layout.jade
create : node_app/views/error.jade
create : node_app/bin
create : node_app/bin/www
```

Generación de proyectos

- Vamos a repasar los ficheros y archivos que genera:

• node_app/package.json	Contiene las dependencias del proyecto para npm
• node_app/app.js	Es el donde se inicia toda nuestra aplicación
• node_app/public/javascripts	En estas carpetas se almacenarían todas las
• node_app/public/images	imágenes, scripts de JavaScript y estilos
• node_app/public/stylesheets	
• node_app/public/stylesheets/style.css	
• node_app/routes/index.js	Estos ficheros se encargan de gestionar las rutas
• node_app/routes/users.js	
• node_app/views/index.jade	En esta carpeta se almacenan los ficheros jade que
• node_app/views/layout.jade	son los que definen la interfaz que tendrá nuestra
• node_app/views/error.jade	aplicación.

Generación de proyectos

- Una vez creada la estructura de proyecto, debemos instalar las dependencias que se han registrado en el fichero package.json:

```
cd node_app
sudo npm install
```

- Una vez instalados los módulos necesarios se puede lanzar nuestra aplicación básica y ver cómo funciona, para ello podemos hacerlo de dos maneras diferentes:

```
DEBUG=node_app:* ./bin/www

npm start
```

- Al lanzar el servidor nos debe indicar el puerto en el que está disponible, podemos acceder a través del navegador con la url: `localhost:3000`

Routing

- Express proporciona métodos para la gestión de las rutas:

```
var express = require('express')
var app = express()

// responderá con "hello world" cuando una petición GET se hace a la
página principal
app.get('/', function(req, res) {
  res.send('hello world')
})
```

- Tanto para peticiones GET como POST:

```
// petición GET
app.get('/', function (req, res) { res.send('GET request to the homepage' )})

// petición POST
app.post('/', function (req, res) { res.send('POST request to the homepage' )})
```

- Express soporta los siguientes tipos de peticiones: `get`, `post`, `put`, `head`, `delete`, `options`, `trace`, `copy`, `lock`, `mkcol`, `move`, `purge`, `propfind`, `proppatch`, `unlock`, `report`, `mkactivity`, `checkout`, `merge`, `m-search`, `notify`, `subscribe`, `unsubscribe`, `patch`, `search` y `connect`

Routing

- Express también puede gestionar las rutas que son llamadas, independientemente del tipo de petición:

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

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

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

- También tiene ciertos caracteres que amplían las posibilidades de las rutas:

```
// La "b" es opcional, aceptará acd y abcd
app.get('/ab?cd', function(req, res) { res.send('ab?cd') })

// La "b" podrá repetirse, aceptará abcd, abbcd, abbbcd, etc...
app.get('/ab+cd', function(req, res) { res.send('ab+cd') })

// Se aceptará cualquier cadena dentro, aceptará abcd, abxcd, abREBDOMcd, ab123cd, etc...
app.get('/ab*cd', function(req, res) { res.send('ab*cd') })

// Lo incluído entre paréntesis puede aparecer o no, aceptará abe o abcde
app.get('/ab(cd)?e', function(req, res) { res.send('ab(cd)?e') })
```

Routing

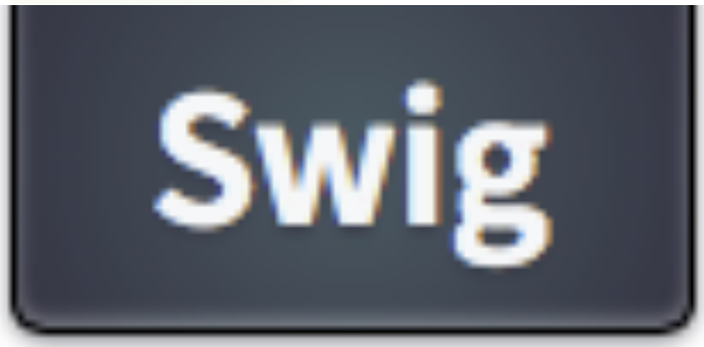
- Express también puede devolver diferentes tipos de respuestas:
  - **res.download()**: Proporciona la descarga de un archivo
  - **res.end():** Termina el proceso de la respuesta
  - **res.json():** Devuelve un *JSON* por respuesta
  - **res.redirect():** Redirige la petición
  - **res.render():** Renderiza una vista
  - **res.send():** Devuelve una respuesta genérica

Swig

- Swig es un editor de plantillas simple y potente, es uno de los más usados solo superado por Jade.

- Es compatible con Express y se usa de forma similar a las plantillas de Jinja2, Django y Twig.

- Está orientado a objetos, por ellos es extendible y personalizable.



Swig

- Podemos instalar Swig con npm:

```
npm install swig --save
```

- Una vez instalado ya podemos añadir Swig a nuestro proyecto de la siguiente manera:

```
var swig = require('swig');

// Compila y guarda el fichero
var tpl = swig.compileFile('/path/to/template.html');

// De esta manera se renderiza desde un string directamente
swig.render('{% if foo %}Hooray!{% endif %}', { locals: { foo: true }});
```

Swig

- Las variables que se pasan a las plantillas se pueden imprimir utilizando el . o la variable entre corchetes

```
{{ foo.bar }}
// es equivalente a
{{ foo['bar'] }}
```

- Sin embargo, el estilo de notación sigue las mismas reglas que JavaScript. Si una clave incluye caracteres no alfanuméricos debe acceder mediante corchetes y no con el .

```
{{ foo.chicken-tacos }}      Erróneo
{{ foo['chicken-tacos'] }}   Correcto
```

- Si una variable no está definidas la plantilla mostrará: `0`, `null` o `false`

Swig

- Los filtros son estructuras que pueden modificar el valor de las variables, por ejemplo:

```
{{ name|title }} was born on {{ birthday|date('F jS, Y') }}
// => Jane was born on July 6th, 1985
```

- Las variables también pueden ser funciones JavaScript. Es importante tener en cuenta que los caracteres especiales de escape no se mostrarán:

```
var locals = { mystuff: function mystuff() { return '<p>Things!</p>'; } };
swig.render('{{ mystuff() }}', { locals: locals });
// => <p>Things!</p>
```

- Si queremos explícitamente que se muestren lo debemos hacer con el filtro *scape*:

```
{{ mystuff()|escape }}
// => &lt;p&gt;Things&lt;/p&gt;
```

Swig

- Swig incluye bloques operacionales, llamados etiquetas, que ayudan a mostrar una lista de variables. Las etiquetas se escriben usando `{% %}`:

```
{% if foo %}bar{% endif %}

// Crea una lista de personas si hay
elementos en la lista de personas
{% for person in people %}
  {% if loop.first %}<ol>{% endif %}
  <li>{{ person.name }}</li>
  {% if loop.last %}</ol>{% endif %}
{% endfor %}
```

- Las etiquetas `end` se usan para finalizar el bloque y se les puede añadir un texto que será ignorado, pero que ayudará a entender el código:

```
{% block tacos %}
  //...
{% endblock tacos %}
{% block burritos %}
  {% if foo %}
    // ...
  {% endif comprueba si foo == true %}
{% endblock burritos %}
```

Swig

- Las etiquetas de comentarios son ignoradas por el compilador. Se eliminan cuando se renderiza la plantilla por lo que nadie, que no tenga el código fuente, podrá verlo:

```
{#
Esto es un comentario.
Será completamente ignorado en tiempo de ejecución.
#}
```

- Todos los espacios en blanco en las plantillas acaban apareciendo en las plantillas finales, a no ser que lo impidamos. Para ello simplemente hay que añadir – en la etiqueta del principio o del final:

```
// seq = [1, 2, 3, 4, 5]
{% for item in seq -%}{{ item }}
{%- endfor %}
// => 12345
```

Swig

- Swig permite extender bloques de nuestras plantillas, si creamos la plantilla `layout.html`, después podremos extender bloques de esa plantilla en `index.html`:

```
//layout.html

<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>
      {% block title %}My Site{% endblock %}
  </title>

  {% block head %}
  <link rel="stylesheet" href="main.css">
  {% endblock %}
</head>
<body>
  {% block content %}{% endblock %}
</body>
</html>
```

```
//index.html

{% extends 'layout.html' %}

{% block title %}My Page{% endblock %}

{% block head %}
  {% parent %}
  <link rel="stylesheet" href="custom.css">
{% endblock %}

{% block content %}
<p>This is just an awesome page.</p>
{% endblock %}
```

Swig

- Swig es compatible con Express de forma muy sencilla, un ejemplo de integración con Express sería el siguiente:

```
var app = require('express')(),
  swig = require('swig'),
  people;

// Esta función es la encargada de integrar Swig con Express
app.engine('html', swig.renderFile);

app.set('view engine', 'html');
app.set('views', __dirname + '/views');

// Swig hará caché de las plantillas en tu lugar, aunque podemos
// deshabilitar esta y habilitar la de Express, si nos interesa especialmente:
app.set('view cache', false);
// Para deshabilitar la caché de Swig:
swig.setDefaults({ cache: false });
// NOTA: Siempre debes hacer caché de las plantillas en un entorno de
// producción, pero nunca dejes ambas deshabilitadas

app.get('/', function (req, res) {
  res.render('index', { /* template locals context */ });
});

app.listen(1337);
console.log('Application Started on http://localhost:1337/');
```

Twig.js

- Si eres un desarrollador web de PHP seguramente te habrá parecido muy similar Swig a un sistema de plantillas que con casi toda seguridad ya conoces, Twig. Y la verdad, si parecen en mucho más que en el nombre, ya que una vez que Twig fue portado a node **Swig dejó de tener soporte.**

- Twig.js es una implementación en JS puro de Twig Php

- Para utilizar este sistema de plantillas lo primero que tendremos que hacer es instalarlo.

```
$ npm install twig
```

Twig

- Y como hemos hecho con Swig, decirle a nuestro express que utilice este sistema y no Jade:

```
//Twig
var Twig = require("twig");
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'twig');
// This section is optional and used to configure
twig.
app.set("twig options", {
    strict_variables: false
});
```

- * Con esta configuración vistas ahora seguirán la nomenclatura /views/index.twig y no /views/index.html