

DESARROLLO WEB CON MEAN

Curso: DESARROLLO WEB CON MEAN (WEB FULL STACK DEVELOPER)

¿Qué es nodejs?

- **Node.js (de ahora en adelante Node) es un entorno JavaScript de lado de servidor que utiliza un modelo asíncrono y dirigido por eventos.**
- **Node soporta protocolos TCP, DNS y HTTP.**

¿Qué es nodejs?

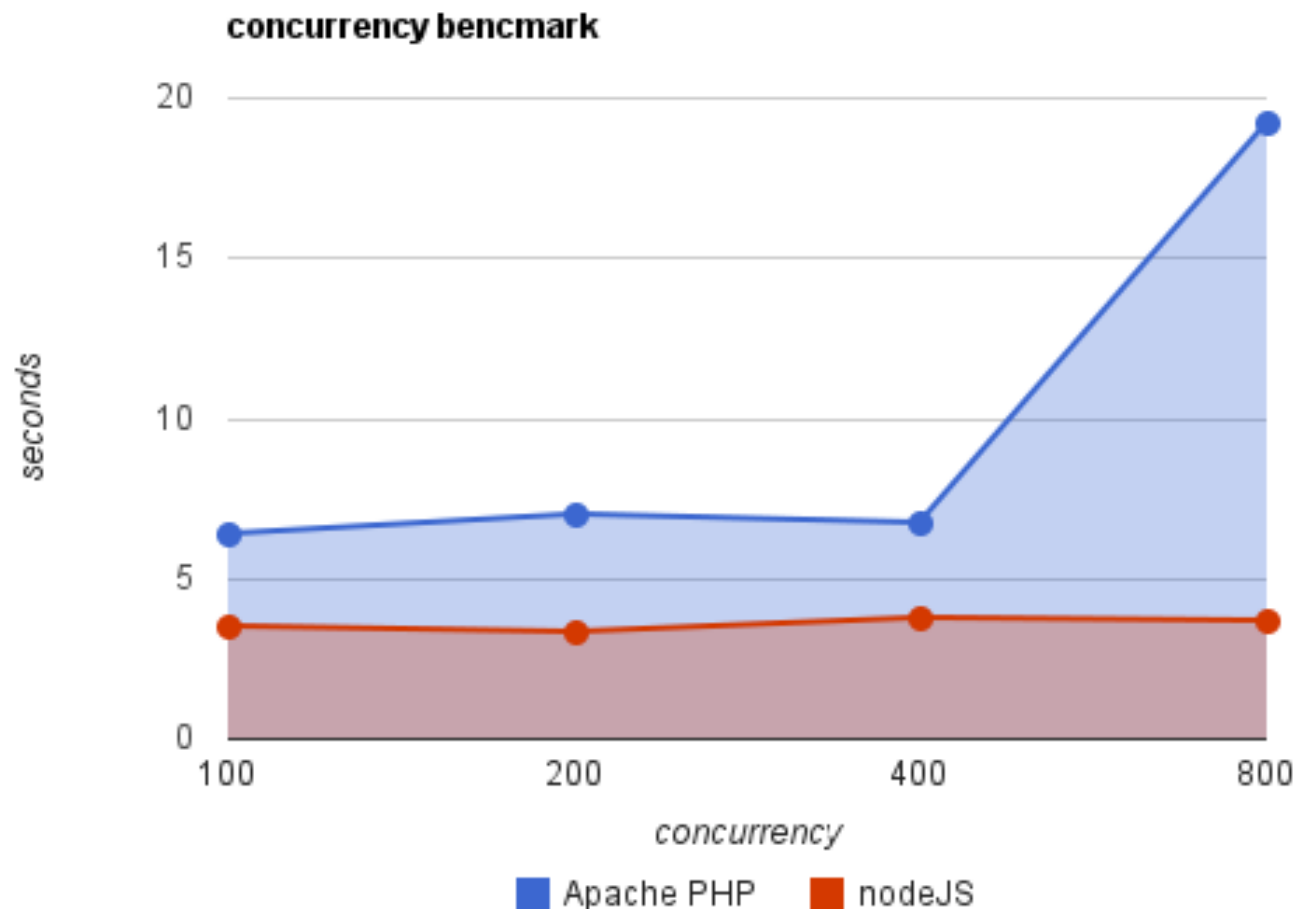
- **Node usa el motor de JavaScript V8 de Google: una VM tremendamente rápida y de gran calidad escrita por gente como Lars Bak, uno de los mejores ingenieros del mundo especializados en VMs.**

¿Qué es nodejs?

- **No olvidemos que V8 es actualizado constantemente y es uno de los intérpretes más rápidos que puedan existir en la actualidad para cualquier lenguaje dinámico.**
- **Además las capacidades de Node para I/O (Entrada/Salida) son realmente ligeras y potentes, dando al desarrollador la posibilidad de utilizar a tope la I/O del sistema.**

NODE.JS

¿Qué diferencias tiene respecto a Apache u otros servidores web?



NODE.JS

- Para que os hagáis la idea de cómo

```
var http = require('http');
var s = http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
});
s.listen(8000);
console.log('Server running!');
```

(hello_world.js):

- Ahora iniciamos la aplicación:

- Cuando iniciamos la aplicación, Node le dice al sistema que le avise (mediante epoll, kqueue, etc.) cuando un cliente se conecte. Mientras tanto se pondrá a dormir (sleep). Si alguien se conecta, se ejecuta el callback anónimo definido en createServer. Por cada conexión se realiza una pequeña reserva de memoria en un heap.
- Si abrimos `http://localhost:8000/` en el navegador entonces obtenemos "Hello World".



```
TERMINAL
$ node hello_world.js
Server running!
```

NODE.JS

- Otro ejemplo:

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

```
fs.readFile('report.txt', function(data) {console.log('Read: ' + data)});
```

```
fs.writeFile('message.txt', 'Hello World!', function() {console.log('File saved!')});
```

- A Node se le dará la función de leer y escribir en los ficheros y a continuación pasará a dormir (sleep). Cuando las operaciones se terminen se ejecutarán los callbacks asociados a cada tarea. Como explicamos anteriormente, no hay nada que garantice el orden en el que van a ser mostrados los mensajes de los callbacks. Esta manera de ejecutar la I/O asegura que el hilo principal de programa siempre va a estar en continuo movimiento llamando nuevas tareas para realizar en segundo plano.

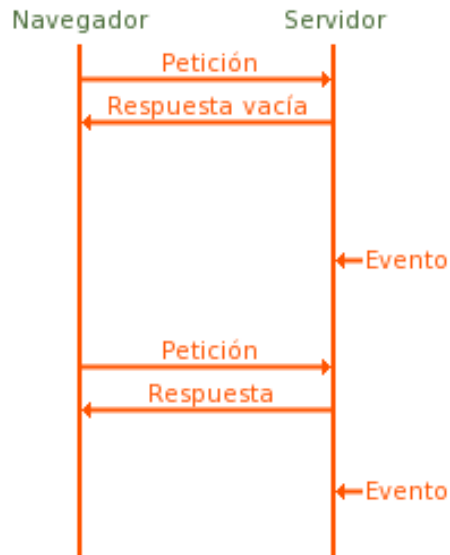
NODE.JS

- **Node es especialmente bueno en aplicaciones web que necesiten una conexión persistente con el navegador del cliente.**
- **Mediante una serie de técnicas llamadas Comet, puedes hacer una aplicación que envíe datos al usuario en tiempo real; es decir, que el navegador mantenga la conexión siempre abierta y reciba continuamente nuevos datos cuando los haya.**
- **Para servidores y aplicaciones clásicas que no estén preparadas para mantener muchas conexiones, la forma más sencilla es solicitar desde el navegador un cada X segundos nueva información (polling).**
- **Si tenemos muchos usuarios a la vez abriendo conexiones y haciendo peticiones a la BD nos encontramos con que nuestro servidor no da más de sí y deja de atender peticiones -acaba petando, vamos-. Un ejemplo de polling clásico es la fisgona de Menéame que realiza peticiones mediante AJAX a un script PHP que devuelve nuevos datos en JSON cada 3 segundos y actualiza la tabla de novedades.**

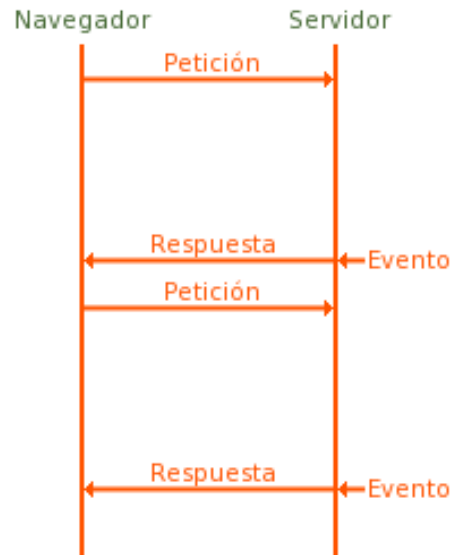
NODE.JS

Ajax vs. Comet

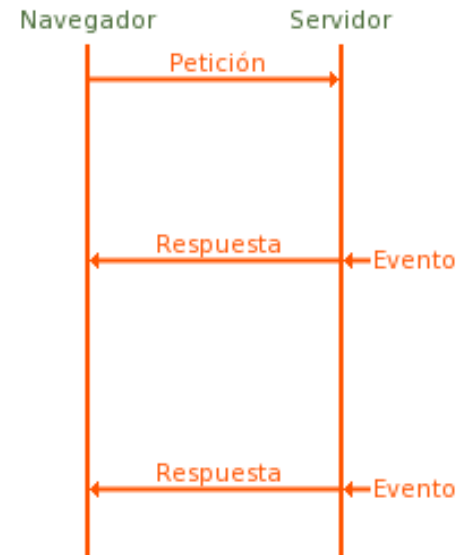
Ajax



Comet (con long-polling)



Comet (con websockets)



¿Cuándo usarlo?

- **Una lista de aplicaciones para las que Node encajaría perfectamente:**
 - ☐ Juegos online.
 - ☐ Gestores de correo online: de esta manera teniendo el navegador abierto podríamos ver notificaciones en tiempo real de nuevos correos recibidos.
 - ☐ Herramientas de colaboración.
 - ☐ Chats.
 - ☐ Redes sociales: por ejemplo para actualizar automáticamente tu muro de novedades.
 - ☐ Herramientas de traducción en tiempo real.

Librerías interesantes

- **Socket.IO: "Socket.IO aims to make realtime apps possible in every browser and mobile device, blurring the differences between the different transport mechanisms. Under the hoods, Socket.IO will use feature detection to decide if the connection will be established with WebSocket, AJAX long polling, etc (see supported transports), making creating realtime apps that work everywhere a snap."**

Librerías interesantes

- **Express: un framework web sobre Node. Manual Daniweb, getting started.**
- **node.dblayer.js: "it's a very basic and easy-to-use library to connect to a DBSlayer server, which effectively provides non-blocking and scalable MySQL support for Node."**

Librerías interesantes

- **Handlebars.js: "provides the power necessary to let you build semantic templates effectively with no frustration." Buen manual de Think Vitamin.**
- **MongoDB driver o Mongoose.**

NODE.JS

módulos

- Node posee un sencillo sistema de carga. En Node, los ficheros y módulos son de correspondencia biunívoca. A modo de ejemplo, foo.js carga el módulo circle.js en el mismo directorio.
- El contenido de foo.js:

```
var circle = require('./circle.js');  
console.log( 'El área de un círculo con radio 4 es ' + circle.area(4));
```

- El contenido de circle.js:

```
var PI = Math.PI;  
exports.area = function (r) { return PI * r * r;};  
exports.circumference = function (r) { return 2 * PI * r;};
```

- El módulo circle.js ha exportado las funciones area() y circumference(). Para exportar a un objeto, debe añadir el objeto especial exports.
- Las variables locales del módulo serán privadas. En este ejemplo la variable PI es privada en circle.js.

NODE.JS

Módulos básicos

- Node posee varios módulos compilados en binario. Estos módulos son descritos con más detalle en las siguientes secciones del documento.
- Los módulos básicos son definidos en el código fuente de node en la carpeta lib/.
- Los módulos básicos tienen la preferencia de cargarse primero si su indentificador es pasado desde require(). Por ejemplo, require('http') siempre devolverá lo construido en el módulo HTTP, incluso si hay un fichero con ese nombre.

NODE.JS

Módulo File

- Si el nombre exacto del fichero no es encontrado, entonces node intentará cargar el nombre del fichero seguido de la extensión .js, y a continuación con .node.
- Los ficheros .js son interpretados como ficheros de texto en JavaScript, y los ficheros .node son interpretados como extensiones de módulos compilados cargados con dlopen.
- Un módulo con el prefijo '/' indica la ruta absoluta al fichero. Por ejemplo, require('/home/marco/foo.js') cargará el fichero en /home/marco/foo.js.
- Un módulo con el prefijo './' es relativo al fichero llamado con require(). Es decir, circle.js debe estar en el mismo directorio que foo.js para que require('./circle') lo encuentre.
- Si se omite el uso de '/' o './' en el fichero, el módulo puede ser un "módulo básico" o se cargará desde la carpeta node_modules.

NODE.JS

Cargando desde la carpeta

- Si el identificador del módulo pasa a `require()` no es un módulo nativo, y no comienza con `'/'`, `'../'`, o `'./'`, entonces node inicia en el directorio principal del módulo actual, y añade `/node_modules`, e intenta cargar el módulo desde esa ubicación.
- Si no se encuentra, entonces se dirige al directorio principal, y así sucesivamente, hasta que el módulo es encontrado, hasta en la raíz del árbol es encontrado.
- Por ejemplo, si el fichero en `'/home/ry/projects/foo.js'` es llamado como `require('bar.js')`, entonces node buscaría en las siguientes ubicaciones, en este orden:
 - `/home/ry/projects/node_modules/bar.js`
 - `/home/ry/node_modules/bar.js`
 - `/home/node_modules/bar.js`
 - `/node_modules/bar.js`
- Esto permite que los programas encuentren sus dependencias, de modo que no entren en conflicto.

NODE.JS

Optimización de proceso de búsqueda

- Cuando existen muchos niveles de dependencias anidadas, es posible que los árboles de directorios tomen bastante tiempo. Las siguientes optimizaciones se realizan para este proceso.
- Primero, `/node_modules` no debe ser anexado a una carpeta ya que termina en `/node_modules`.
- Segundo, si el fichero es llamado con `require()` ya esta en la jerarquía de `node_modules`, entonces el nivel superior de la carpeta `node_modules` es tratada como la raíz del árbol de búsqueda.
- Por ejemplo, si el fichero en `'/home/ry/projects/foo/node_modules/bar/node_modules/baz/quux.js'` llama como `require('asdf.js')`, entonces node buscaría en las siguientes ubicaciones:
 - `/home/ry/projects/foo/node_modules/bar/node_modules/baz/node_modules/asdf.js`
 - `/home/ry/projects/foo/node_modules/bar/node_modules/asdf.js`
 - `/home/ry/projects/foo/node_modules/asdf.js`

NODE.JS

Carpetas como módulos

- Es conveniente organizar los programas y librerías en los mismos directorios, y proporcionar un único punto de entrar a la biblioteca. Existe tres formas en donde una carpeta pueda usar `require()` como un argumento.
- Lo primero es crear el fichero `package.json` en la raíz de la carpeta, que especifique el módulo `main`. Un ejemplo de `package.json` podría verse como esto:

```
{ "name": "some-library", "main": "./lib/some-library.js" }
```
- Si fuera una carpeta en `./some-library`, entonces `require('./some-library')` trataría de cargar `./some-library/lib/some-library.js`.
- Este es el mayor grado de conciencia de Node con el fichero `package.json`.
- Si no hay ningún fichero `package.json` presente en el directorio, entonces node intentará cargar el fichero `index.js` o `index.node` de ese directorio. Por ejemplo, si no hay ningún fichero `package.json` en el ejemplo anterior, entonces `require('./some-library')` intentará cargar:
 - `./some-library/index.js`
 - `./some-library/index.node`

NODE.JS

Almacenamiento en la caché

- Los módulos se almacenan en la caché después que fueron cargados por primera vez. Esto significa (entre otras cosas) que todas las llamadas a `require('foo')` devuelve el mismo objeto exacto, si se resolvería en el mismo fichero.

NODE.JS

Cargar desde las carpetas

- En node, `require.paths` es un array de strings que representa las rutas de acceso a los módulos cuando estos no tienen el prefijo `'/'`, `'./'`, o `'../'`. Por ejemplo, si establece `require.paths` como:

```
[ '/home/micheil/.node_modules', '/usr/local/lib/node_modules' ]
```
- A continuación se llama a `require('bar/baz.js')` y buscará en las siguientes ubicaciones:
1: `'/home/micheil/.node_modules/bar/baz.js'`
2: `'/usr/local/lib/node_modules/bar/baz.js'`
- El array en `require.paths` puede ser transformado en tiempo de ejecución para modificar este comportamiento.
- Se establece inicialmente la variable de entorno `NODE_PATH`, que contiene una lista delimitada por dos puntos de rutas exactas. En el anterior ejemplo, la variable de entorno `NODE_PATH` pudo haber sido establecido como:
 - `/home/micheil/.node_modules:/usr/local/lib/node_modules`
- Cargar las ubicaciones desde `require.paths` sólo se realiza si el módulo no se ha encontrado desde el algoritmo `node_modules`. Los módulos globales son de baja prioridad para las dependencias de los paquetes.

NODE.JS

¿Que es NPM?

- Como bien dice la página oficial de npm, "npm es un gestor de paquetes para javascript", es el predeterminado para node.js, cuando instalas node también se instala npm. ¿Y que quiere decir esto? Pues que a través de npm podremos instalar y gestionar los paquetes para nuestras aplicaciones.

- Después de esta pequeña introducción



NODE.JS

Crear una aplicación con npm

- Si lo que queremos es crear una aplicación desde 0, tenemos el comando `init`
npm init
- Cuando ejecutamos este comando npm nos hace varias preguntas para realizar la configuración inicial de nuestro proyecto, como el nombre del proyecto, si tenemos un repositorio git, descripción del proyecto, la versión actual, etc. Respondemos a todas las preguntas y npm nos creará un archivo `package.json` con los datos que le acabamos de introducir.
- El archivo `package.json` lleva la configuración del paquete, donde se guardarán las dependencias de paquetes del proyecto y la configuración básica de este.

NODE.JS EJERCICIO

- **USANDO NODEJS CREE UN SERVIDOR HTTP QUE ESCUCHE EN EL PUERTO 8088**
- **AL SER VISITADA LA PÁGINA, SI SE ENVIA POR PARÁMETRO LA CLAVE NIF, SE ANALIZARÁ EL VALOR Y DEVOLVERÁ UN TEXTO HTML CON TRUE SI ES UN NIF/NIE/CIF Y TEXTO FALSE SI NO ES CORRECTO.**

Introducción

Express.js está basado en Connect, framework extensible para servidores http.

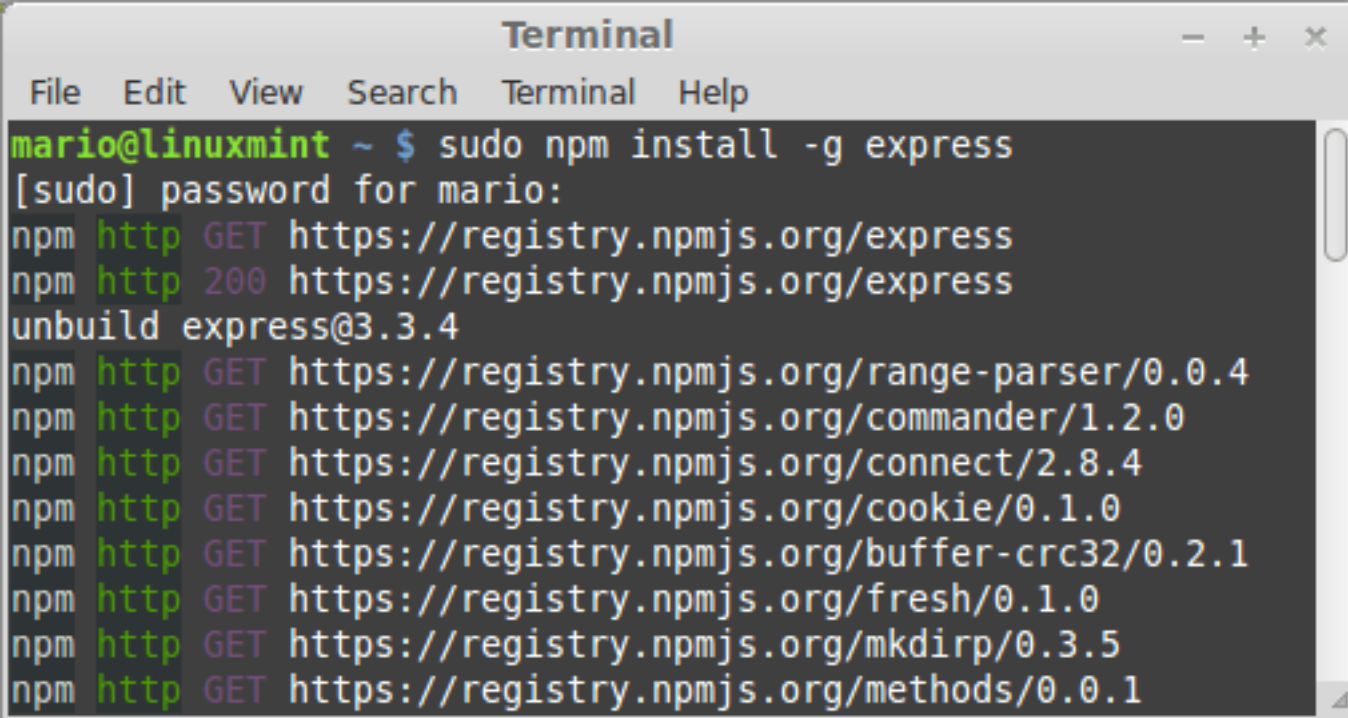
Posee plugins de alto rendimiento conocidos como Middleware.

Middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos

EXPRESS

Instalación

npm install -g express

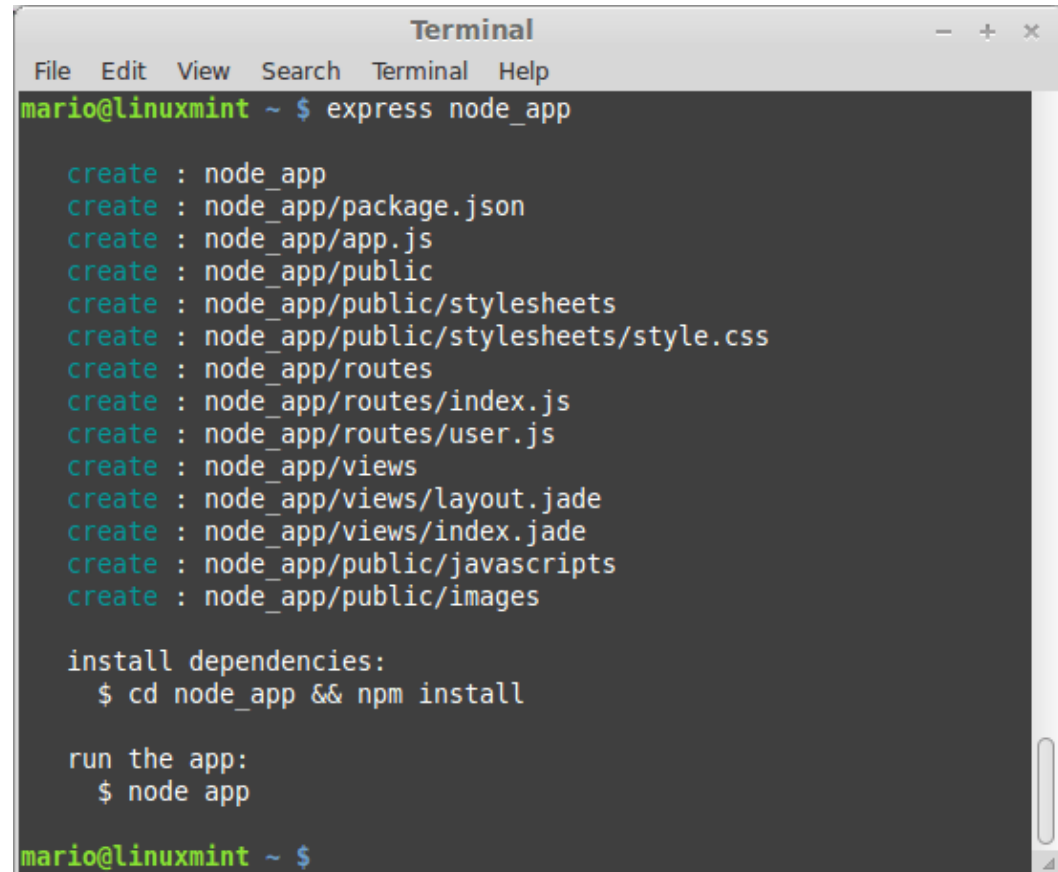
A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help) and window control buttons. The terminal shows the command "sudo npm install -g express" being executed by user "mario" on a "linuxmint" machine. It prompts for a password, then shows the npm process downloading "express" and its dependencies. The output includes the version "express@3.3.4" and a list of dependencies being fetched from the npm registry.

```
Terminal
File Edit View Search Terminal Help
mario@linuxmint ~ $ sudo npm install -g express
[sudo] password for mario:
npm http GET https://registry.npmjs.org/express
npm http 200 https://registry.npmjs.org/express
unbuild express@3.3.4
npm http GET https://registry.npmjs.org/range-parser/0.0.4
npm http GET https://registry.npmjs.org/commander/1.2.0
npm http GET https://registry.npmjs.org/connect/2.8.4
npm http GET https://registry.npmjs.org/cookie/0.1.0
npm http GET https://registry.npmjs.org/buffer-crc32/0.2.1
npm http GET https://registry.npmjs.org/fresh/0.1.0
npm http GET https://registry.npmjs.org/mkdirp/0.3.5
npm http GET https://registry.npmjs.org/methods/0.0.1
```

EXPRESS

Construcción de aplicación con Express

express node_app



```
Terminal
File Edit View Search Terminal Help
mario@linuxmint ~ $ express node_app

create : node_app
create : node_app/package.json
create : node_app/app.js
create : node_app/public
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/user.js
create : node_app/views
create : node_app/views/layout.jade
create : node_app/views/index.jade
create : node_app/public/javascripts
create : node_app/public/images

install dependencies:
$ cd node_app && npm install

run the app:
$ node app

mario@linuxmint ~ $
```

EXPRESS

Ejecución de una aplicación Express

Node mi_app



Express

Welcome to Express

```
Terminal
File Edit View Search Terminal Help
mario@linuxmint ~/node_app $ node app.js
Express server listening on port 3000

GET / 200 311ms - 170b
GET /stylesheets/style.css 200 10ms - 110b
```

Ejemplo

Creamos una app que responda a peticiones según path con Express

```
var express = require("express");
var app = express();
app.get('/', function(req, res) {
    res.send('Prueba de express.js');
});
app.get('/test', function(req, res) {
    res.send('Prueba de express.js en /test');
});
app.listen(8080);
```

Creación de servidores

Invocar con `createServer()`

```
var app = require('express').createServer();
```

```
app.get('/', function(req, res){  
  res.send('hello world');  
});
```

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

EXPRESS

Configuraciones soportadas por Express

basepath: Path base en el uso de `res.redirect()`

views: Directorio base de vistas

view engine: Nombre del motor base de vista

view options: Objeto que especifica las opciones del engine

view cache: Si se activa la caché (activado en producción)

case sensitive routes: Activar rutas case-sensitive

strict routing: Las barras al final no son ignoradas en el path

jsonp callback: Activa `res.send()` y `res.json()`. Activa el soporte transparente de jsonp

Enrutado

Express define los verbos http como un sistema de enrutado expresivo y lógico

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

Una ruta es una cadena que se compila por medio de una expresión regular. Por ejemplo, para el caso anterior:

```
VuserV([ ^V ]+ )V?
```


EXPRESS

Enrutado

En caso de necesitar mayor complejidad de enrutados, se permite definir la expresión regular en el path.

```
app.get(/^\/users?(?:\/(\d+)(?:\.\/(\d+))?)?\/, function(req, res){  
  res.send(req.params);  
});
```

Respuestas coincidentes a la expresión anterior

```
http://dev:3000/user  
[null,null]
```

```
http://dev:3000/users  
[null,null]
```

```
http://dev:3000/users/1  
["1",null]
```

```
http://dev:3000/users/1..15  
["1",null]
```

Otros ejemplos:

`"/users/:id?"`

`/users/5`

`/users`

`"/files/*"`

`/files/jquery.js`

`/files/javascripts/jquery.js`

`"/file/*.?"`

`/files/jquery.js`

`/files/javascripts/jquery.js`

`"/user/:id/:operation?"`

`/user/1`

`/user/1/edit`

Otras opciones

`app.all()`

Aplica la misma lógica a todas los verbos http en una sola llamada

```
app.all('/user/:id/:op?', function(req, res, next){  
  req.user = users[req.params.id];  
  if (req.user) {  
    next();  
  } else {  
    next(new Error('cannot find user ' + req.params.id));  
  }  
});
```

Middleware

Podemos definir middleware a la hora de crear el servidor por medio de Connect

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

```
var app = express.createServer(  
  express.logger()  
  , express.bodyParser()  
);
```

Middleware

Para usar el middleware:

```
app.use(express.logger({ format: ':method :url' }));
```

Se debe declarar connect

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

```
app.use(connect.logger());
```

```
app.use(connect.bodyParser());
```

Route Middleware

Podemos definir middleware en el propio enrutado

```
app.get('/user/:id', function(req, res, next){
  loadUser(req.params.id, function(err, user){
    if (err) return next(err);
    res.send('Viewing user ' + user.name);
  });
function loadUser(req, res, next) {
  // Obtenemos de la base de dato el usuario
  var user = users[req.params.id];
  if (user) {
    req.user = user;
    next();
  } else {
    next(new Error('Failed to load user ' + req.params.id));
  }
}
```

Métodos HTTP

Anteriormente hemos usado el verbo GET

En POST, uso típico de formularios, debemos indicar a Express como parsear la información

Para indicarle que hacer usamos el middleware bodyParser

```
app.use(express.bodyParser());
```

Mostramos el nombre del campo user almacenado en el formulario con name=user

```
app.post('/', function(req, res){  
  console.log(req.body.user);  
  res.redirect('back');  
});
```

Uso de otros verbos

Por defecto, via formulario no es posible usar los verbos put y delete

Para ello usamos una variable interna que indique cual es el verbo a sobrescribir

```
<form method="post" action="/">
  <input type="hidden" name="_method" value="put" />
  <input type="text" name="user[name]" />
  <input type="text" name="user[email]" />
  <input type="submit" value="Submit" />
</form>
```

Responde:

```
app.put('/', function(){
  console.log(req.body.user);
  res.redirect('back');
});
```


Gestión de errores

Poseemos el método `app.error()` para gestionar errores

```
function NotFound(msg){  
  this.name = 'NotFound';  
  Error.call(this, msg);  
  Error.captureStackTrace(this, arguments.callee);  
}
```

```
NotFound.prototype.__proto__ = Error.prototype;
```

```
app.get('/404', function(req, res){  
  throw new NotFound;  
});
```

Pre-condiciones

Útiles para temas como validaciones

Podemos esperar un valor y evaluarlo. En caso de que no sea correcto, redirigir.

```
app.param('userId', function(req, res, next, id){  
  User.get(id, function(err, user){  
    if (err) return next(err);  
    if (!user) return next(new Error('failed to find user'));  
    req.user = user;  
    next();  
  });  
});
```

Renderizado de la vista

Por defecto se utiliza el formato

Nombre.engine

El engine indica el motor de renderizado a utilizar para compilar el resultado

```
app.get('/', function(req, res){  
  res.render('index.jade', { title: 'My Site' });  
});
```

En este método se usa el motor de jade para compilar la página index

Renderizado de la vista

Para usar un renderizado por defecto

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

Y permite renderizar jade con

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

Renderizado de la vista

Express permite cambiar los renderizados de la vista para que se comporten de distinta manera según necesidades

```
app.set('view options', {  
  layout: false  
});
```

E incluso hacerlo de forma individual

```
res.render('myview.ejs', { layout: true });
```

Vistas parciales

Express tiene soporte para vistas parciales

Son pequeñas vistas de fragmentos de una página

```
partial('comment', { collection: comments });
```

Motores de plantillas

Express soporta distintos motores como

- HAML
- JADE
- EJS
- COFFEECUP
- JQUERY TEMPLATES

EJERCICIO

Genere un formulario de tipo cliente que valide los siguientes campos

- Nombre (10 caracteres)
- Apellido1 (10 caracteres)
- Apellido2 (10 caracteres)
- NIF/NIE/CIF (Valido)

Si es mandado por método POST, se debe enviar un ID obligatorio

Si es mandado por PUT, el ID debe estar vacío.

En caso de error se debe mandar un mensaje al cliente indicando el tipo de error detectado.