



**UTN.BA**  
UNIVERSIDAD TECNOLÓGICA NACIONAL  
FACULTAD REGIONAL BUENOS AIRES

**Centro de  
e-Learning**

**CURSO:**

# **Desarrollo en Node JS**

**Centro de e-Learning SCEU UTN - BA.**

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

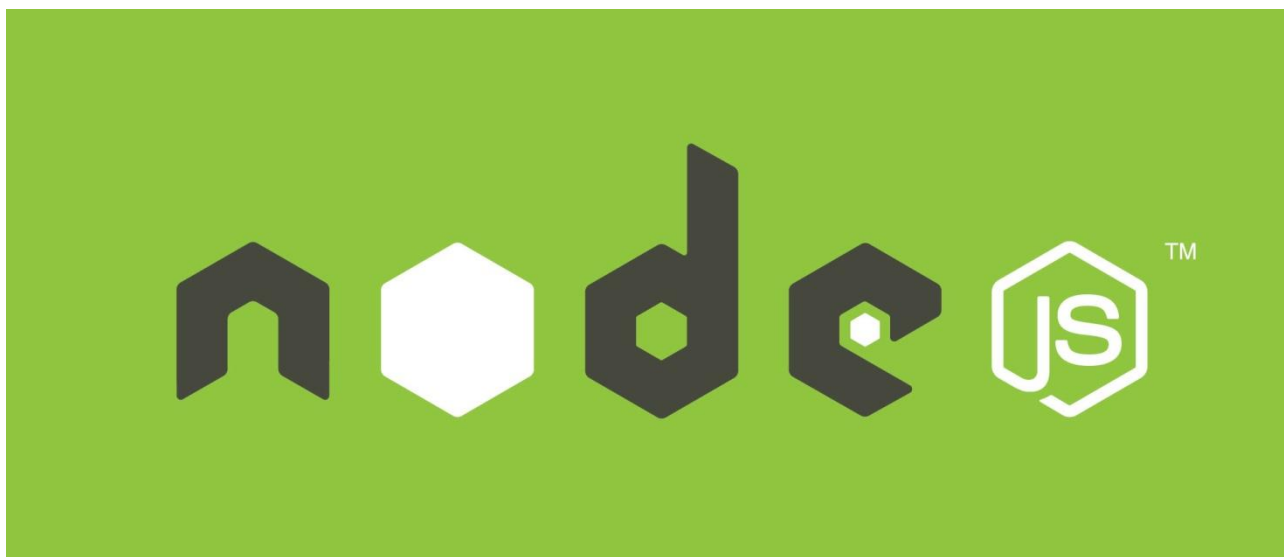
**[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)**



## Unidad 3:

### NodeJS y Mysql - Framework Express

---





## Presentación:

En esta Unidad nos introduciremos en el uso del framework Express, que nos proveerá de algunas facilidades a la hora de programar con Node.

Este framework incorpora de manera automática las plantillas Jade. Este sistema de plantillas junto con Express nos van a facilitar mucho la tarea de interactuar con el cliente.



## Objetivo:

Al terminar la Unidad los participantes:

- Sabrán qué es Express, cómo instalarlo y utilizarlo.
- Sabrán qué son las plantillas Jade.
- Harán implementación de ejemplos prácticos, interactuando con Express, Jade, HandleBars, HTML5 y CSS.



## Bloques temáticos:

### 1. Express

Instalar Express.

Primera aplicación con Express.

Utilización del middleware.

Servir archivos estáticos css, jpg, html etc.

Generador de aplicaciones Express.

Sistemas de Plantillas

JADE.

Handlebars.

Rutas de acceso y plantillas HTML.

ABM y listado de una tabla SQL.

Ejemplo ABM de una tabla SQL con plantillas JADE.

Ejemplo: CRUD con Node.js y plantillas JADE.



## Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC\*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

- El MEC es el modelo de E-learning colaborativo de nuestro Centro.



## Tomen nota

---



Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.



## 1 – Express

---



**Express**, como ya he dicho, es un **framework de Node.js**, y lo que hace es crearte una estructura de trabajo para darte las funcionalidades más repetitivas ya servidas, por ejemplo, la creación del servidor, el manejo de las rutas, la carga de todas las dependencias necesarias y más cosas que iremos viendo poco a poco, esa es la esencia de cualquier framework.

Aparte de esto, **Express** incorpora de forma automática las **plantillas Jade**, que son un sistema de plantillas en las cuáles se hace uso de un pseudo-código que funciona con tabulaciones, también tenemos la posibilidad de hacer uso de herencia, como veremos, es muy sencillo de utilizar y muy limpio.

## Instalar Express

Para instalar Express no tenemos que hacer más que lo que ya veníamos haciendo para instalar cualquier modulo. Creamos la carpeta de nuestro proyecto y desde la línea de comandos de la consola de Node tipeamos:

```
npm install express
```

Esto nos hará la instalación de **Express** en nuestro proyecto.





Si entramos en la carpeta de nuestro proyecto, y luego dentro de la carpeta node\_modules encontraremos la carpeta express como sucedía con todos los módulos que veníamos instalando, dentro de ella encontrarán a su vez una carpeta node\_modules que contiene todos los módulos de los que depende express.

## Primera aplicación con Express

Vamos a crear nuestro “Hola Mundo”, con Express aunque realmente las bondades del uso del framework las veremos en proyectos de mediana y gran complejidad, comencemos por el principio para entender bien cómo utilizarlo.

El código del js es el siguiente:

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

app.get('/',function (req,res){
    res.send('<!doctype html><html><head></head><body><h1>'+
        'Mi primer pagina</h1></body></html>');
});

var server=app.listen(8888,function(){
    console.log('Servidor web iniciado');
});
```

Lo que hacemos es requerir el modulo Express y mediante el método get del objeto app procedemos a especificar que para la ruta '/' (raíz de nuestro sitio) ejecute la función anónima que le pasamos como segundo parámetro.

La función anónima recibe dos objetos 'req' y 'res'. Mediante el objeto 'res' respondemos al navegador que hizo la solicitud enviando código HTML.

Este ejemplo lo encuentran en la carpeta holamundo.



## Utilización del middleware

Express es una infraestructura web de direccionamiento y middleware que tiene una funcionalidad mínima propia: una aplicación Express es fundamentalmente una serie de llamadas a funciones de middleware.

Middleware o lógica de intercambio de información entre aplicaciones ("interlogical") es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones y sincronizaciones que son necesarias en los sistemas distribuidos. De esta forma, se provee una solución que mejora la calidad de servicio, así como la seguridad, el envío de mensajes, la actualización del directorio de servicio, etc.

Es el software que proporciona un enlace entre aplicaciones de software independientes. Middleware a veces se llama a la vía que conecta dos aplicaciones y pasa los datos entre ellas. Por ejemplo los middleware permiten que los datos contenidos en una base de datos puedan ser accedidos a través de otra, ahorrando tiempo a los programadores.

Las funciones de *middleware* son funciones que tienen acceso al **objeto de solicitud** (req), al **objeto de respuesta** (res) y a la siguiente función de middleware en el ciclo de solicitud/respuestas de la aplicación. La siguiente función de middleware se denota normalmente con una variable denominada next.

Las funciones de middleware pueden realizar las siguientes tareas:

- Ejecutar cualquier código.
- Realizar cambios en la solicitud y los objetos de respuesta.
- Finalizar el ciclo de solicitud/respuestas.
- Invocar la siguiente función de middleware en la pila.

Si la función de middleware actual no finaliza el ciclo de solicitud/respuestas, debe invocar next() para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada.

Una aplicación Express puede utilizar los siguientes tipos de middleware:

- [Middleware de nivel de aplicación](#)



- [Middleware de nivel de direccionador](#)
- [Middleware de manejo de errores](#)
- [Middleware incorporado](#)
- [Middleware de terceros](#)

Los métodos o funciones que estuvimos viendo desde el comienzo del curso hasta ahora y que continuaremos viendo en adelante se encuentran dentro de esta clasificación.

Mencionemos algunos ejemplos:

### *Middleware de nivel de aplicación*

Un ejemplo de una función de middleware de nivel de aplicación es la función `use` que veremos en el siguiente ejemplo. La función se ejecuta cada vez que la aplicación recibe una solicitud.

```
var app = express();

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

### *Middleware de nivel del direccionador*

El middleware de nivel de direccionador funciona de la misma manera que el middleware de nivel de aplicación, excepto que está enlazado a una instancia de `express.Router()`.

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

Un ejemplo de función de middleware de nivel de direccionador es `router.use`.

### *Middleware de terceros*

Utilizamos el middleware de terceros para añadir funcionalidad a las aplicaciones Express.

Instalamos el módulo Node.js para la funcionalidad necesaria y lo cargamos en la aplicación.



El siguiente ejemplo ilustra la instalación y carga de la función de middleware de análisis de cookies cookie-parser.

```
$ npm install cookie-parser

var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');

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

Encuentre más información en <http://expressjs.com/es/guide/using-middleware.html>

## Servir archivos estáticos css, jpg, html, etc

Primero que nada vamos a crear una carpeta archivosEstaticos en la cual vamos a instalar Express a través de npm install express .

Tomamos el ejercicio11 de la unidad anterior y copiamos todos los archivos dentro de la carpeta public.

Creamos app.js con el siguiente código:

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

app.use(express.static(__dirname + '/public'));

var server=app.listen(8888,function(){
    console.log('Servidor web iniciado');
});
```

En el cual estamos utilizando la función de middleware “use” de express.static. La variable global ‘\_\_dirname’ almacena el path donde se encuentra la aplicación Node.js propiamente dicha, que es la carpeta donde hayan colocado el proyecto. Le concatenamos la subcarpeta donde se almacenan los archivos estáticos (la carpeta public).

Pueden encontrar más información sobre cómo servir archivos estáticos desde nodejs en <http://expressjs.com/es/starter/static-files.html>

## Recuperar datos de un formulario (post) y de una url (get)

Vamos a ver como tomar datos de un formulario a través de post y como tomar los datos recibidos en una url a través de un ejemplo sencillo, la aplicación permite ingresar dos enteros en el formulario. Luego al presionar el botón submit generamos una página dinámica mostrando todos los números comprendidos entre el primer valor y el segundo de uno en uno. Los números los mostramos como hipervínculos que al ser presionados generan una página dinámica con la tabla de multiplicar del valor seleccionado.

Como lo veníamos haciendo para realizar determinadas tareas tenemos que incluir módulos adicionales. Para recuperar los datos de un formulario HTML o los parámetros de una url debemos agregar un módulo para parsear los datos que llegan del navegador. Hay muchos módulos que hacen esta actividad el más común es el módulo 'body-parser'.

Creamos una carpeta llamada tablas e instalamos a través de npm install los dos módulos, express y body-parser.

Creamos la carpeta public donde incluimos un html que tiene un formulario y creamos la aplicación app.js

Este ejemplo lo encuentran en la carpeta tablas y el código esta comentado para su mejor comprensión.



## Generador de aplicaciones Express

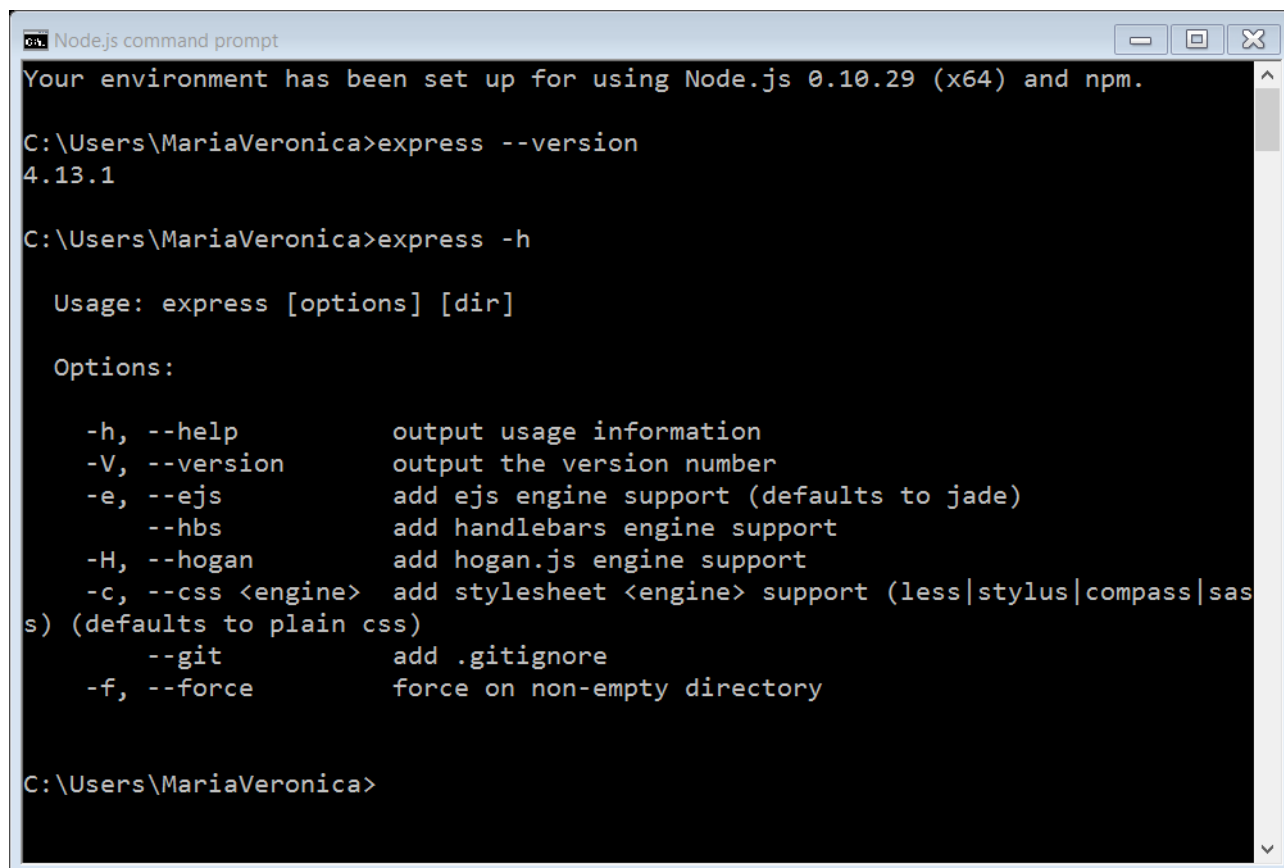
Express posee una herramienta muy útil para ayudarnos a crear rápidamente el esqueleto de una aplicación. Si bien no es obligatorio, de hecho en los ejemplos anteriores no la hemos utilizado, pero es una herramienta de gran utilidad que nos servirá para estructurar en forma ordenada nuestros proyectos.

En la página <http://expressjs.com/es/starter/generator.html> encuentran los pasos para su instalación, de todas formas los describiremos a continuación:

Vamos a instalar express-generator de manera global en nuestro sistema a través de npm install como veníamos haciendo, pero le vamos a agregar el parámetro -g (que indica que la instalación va a ser global)

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

A través del comando `express -h` obtenemos la lista de opciones del mandato `express`:



```
Node.js command prompt
Your environment has been set up for using Node.js 0.10.29 (x64) and npm.

C:\Users\MariaVeronica>express --version
4.13.1

C:\Users\MariaVeronica>express -h

Usage: express [options] [dir]

Options:

  -h, --help            output usage information
  -V, --version          output the version number
  -e, --ejs              add ejs engine support (defaults to jade)
    --hbs                add handlebars engine support
  -H, --hogan            add hogan.js engine support
  -c, --css <engine>    add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
    --git                add .gitignore
  -f, --force            force on non-empty directory

C:\Users\MariaVeronica>
```



Por ejemplo tipeando `express -version` obtenemos la versión de Express que tenemos instalada. En mi caso 4.13.1

Veamos cómo funciona. Nos posicionamos en la carpeta en la que queremos crear nuestra aplicación generada con el generador automático y tipeamos:

```
express myapp
```

Esto va a crear una carpeta `myapp` con todas las carpetas y archivos:

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

Si editamos el archivo `package.json`

```
{
  "name": "ejercicio21",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.13.2",
    "cookie-parser": "~1.3.5",
    "debug": "~2.2.0",
    "express": "~4.13.1",
    "jade": "~1.11.0",
    "morgan": "~1.6.1",
    "serve-favicon": "~2.3.0"
  }
}
```



"name": Indicamos el nombre de proyecto

"version": Indicamos la versión del proyecto.

"start": Indicamos el archivo que se debe ejecutar para arrancar el proyecto

"dependencies": Indicamos todos los otros módulos que se requieren en nuestro proyecto.

Parados en la carpeta myapp tipeamos:

```
npm install
```

Cuando llamamos a 'npm install' sin ningún otro parámetro lo que hace es buscar el archivo 'package.json' y proceder a instalar todos los módulos especificados en la propiedad 'dependencies'.

Para iniciar nuestra aplicación escribimos:

```
npm start
```

Esto va a buscar en el archivo json en el que hay una propiedad start donde definimos el archivo que inicia nuestra aplicación.

Es importante decir que el generador establece por defecto el puerto 3000. Así que para levantar en el navegador nuestra aplicación deberán tipear: <http://localhost:3000/>

## Sistemas de Plantillas

Express da soporte a distintos sistemas de plantillas. Cuando hicimos la instalación a través de express-generator le podemos indicar el motor de plantilla que vamos a utilizar. Por ejemplo podemos escribir:

```
express myapp --hbs
```

En este caso estamos llamando al programa 'express' y le pasamos dos parámetros, el primero indica el nombre de nuestro proyecto y el segundo el sistema de plantillas que utilizaremos para generar nuestras páginas dinámicas (**handlebars**).

En el ejemplo anterior no le pasamos segundo parámetro y asumió por defecto jade.

Hay una gran cantidad de plantillas que se pueden utilizar, pueden encontrar una lista de ellas aquí: <https://www.npmjs.com/package/consolidate>





Nosotros vamos a trabajar con Jade y con Handlebars.

Para que Express pueda representar archivos de plantilla, deben establecerse los siguientes valores de aplicación:

- views, el directorio donde se encuentran los archivos de plantilla. Ejemplo: `app.set('views', './views')`
- view engine, el motor de plantilla que se utiliza. Ejemplo: `app.set('view engine', 'jade')`

Si se fijan en el directorio myapp que creamos con el generador, van a encontrar la carpeta views con 3 archivos:

index.jade, error.jade y layout.jade

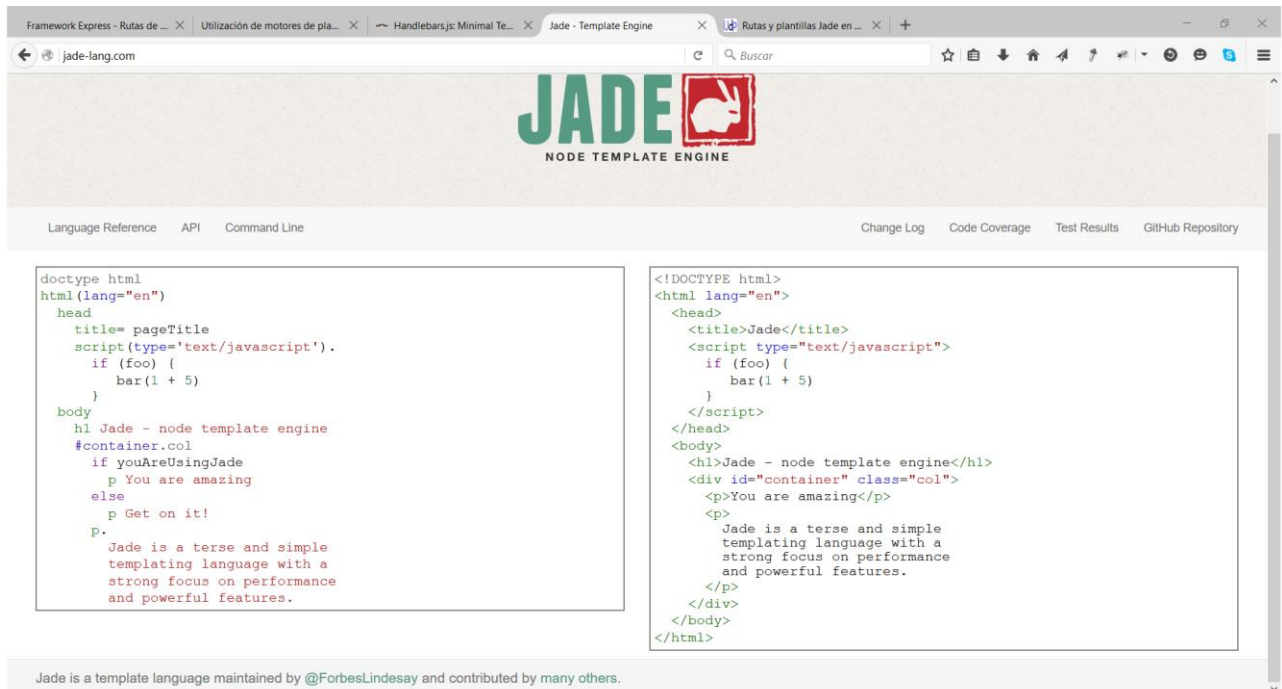
Y en el archivo app.js las siguientes líneas en donde se establece el motor de plantillas que van a utilizar:

```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
```



## JADE

<http://jade-lang.com/>



Jade es un lenguaje de plantillas desarrollado por el creador de Express para simplificar la sintaxis de HTML y acelerar el proceso de desarrollo.

Este lenguaje intercambia tener que cerrar las etiquetas HTML por la indentación, es decir, todo bloque de texto que esté hacia la derecha de la etiqueta que abre, significa que va dentro.

También elimina los símbolos "<" y ">", y los parámetros de cada etiquetas se pasan entre paréntesis como si fuera una función.

Lo más importante a tener en cuenta, es respetar los espacios usados en las tabulaciones, si empiezas con dos espacios, sigue siempre con dos espacios, prefieres tres, tres, pero siempre igual o la plantilla no renderizará.

Veamos un ejemplo:

```
h1 Hola, mi nombre es Oscar
```

h2 y me gusta programar en los siguientes lenguajes:

```
ul
  li Javascript
  li Objective-C
```

Si queremos agregar clases o id solo lo colocamos al lado:

```
h1.titulo Hola, mi nombre es Oscar
h2.subtitulo y me gusta programar en los siguientes lenguajes:

ul#lenguajes
  li Javascript
  li Objective-C
```

Podemos agregar los que queramos y combinarlos:

```
h1.titulo Hola, mi nombre es Oscar
h2.subtitulo.cursiva y me gusta programar en los siguientes lenguajes:

ul#lenguajes.lista
  li Javascript
  li Objective-C
```

Ahora supongamos que queremos escribir un texto muy largo, en una sola línea podría ser tedioso. En jade podemos separar el contenido de una etiqueta en varias líneas utilizando el carácter "|" (pipe):

```
p un texto corto
```

```
p
  | un
  | texto
  | muy
  | largo
```

También hay una forma simplificada, y es agregando un punto después de la etiqueta:

```
p.
  un
  texto
  muy
  largo
```



Todo el contenido de las etiquetas style y script será convertido a CSS y Javascript respectivamente:

```
style
  h1 {
    color: blue;
  }
```

```
script
  var a = 1;
```

Jade soporta JavaScript embebido en el documento agregando el carácter "-" por delante, este código será ejecutado en el lado servidor y no del lado del cliente.

```
- var name = 'oscar';

ul
  li Mi nombre es: #{ name }
```

También podemos mostrar el contenido de la variable asignándolo a la etiqueta:

```
- var name = 'oscar';

ul
  li= #{ name }
```

Si quisiéramos agregar HTML en el código tenemos que reemplazar "#" por "!", de lo contrario el código será impreso como texto:

```
- var name = '<span>oscar</span>';

ul
  li != { name }
```

Los parámetros de las etiquetas se pasan entre paréntesis como si fueran funciones:

```
a(href="/profile", rel="nofollow") ver el perfil
```



Otro ejemplo más complejo: coloca un link dentro de un elemento de una lista justo después de su valor:

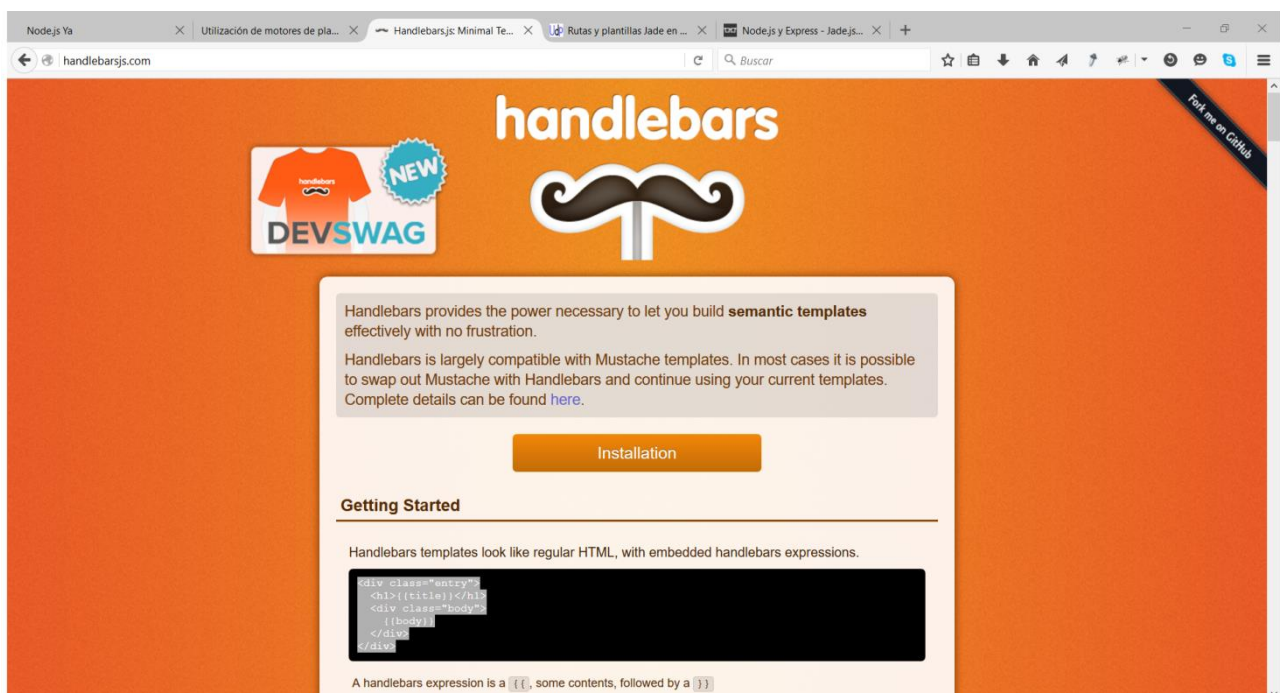
```
- var name = '<span>oscar</span>';

ul
  li= !{ name }
    a(href="/delete") x
```

Veremos más detalle del uso de estas plantillas en los sucesivos ejemplos.

## Handlebars

<http://handlebarsjs.com/>



Las plantillas handlebars se ven como html regular con expresiones handlebars embebidas. Las expresiones handlebars están compuestas por {{ contenido, seguido de }}

Como se ve a continuación en este código:

```
<div class="entry">
```



```
<h1>{{title}}</h1>
<div class="body">
  {{body}}
</div>
</div>
```

Veremos más detalle del uso de estas plantillas en los sucesivos ejemplos.

## Rutas de acceso y plantillas HTML

En el ejemplo myapp con el cual vimos el uso del generador de Express trabajamos con las plantillas JADE.

Vamos a ver como creamos con el generador, una estructura con la plantilla Handlebars.

Para eso creamos la carpeta myapp2 a través del comando:

```
express myapp2 --hbs
```

Esto como ya vimos nos crea la estructura de carpetas y archivos que necesitamos para trabajar. Con npm install instalamos todas las dependencias, y a través de npm start levantamos la aplicación.

Vamos a desarrollar una aplicación de una sola página que nos muestra un listado de 3 artículos y los días que hay descuentos (en este ejemplo los datos no los extraemos de una base de datos para facilitar la comprensión del manejo de ruta y plantillas HTML)

Veremos las porciones de código de los archivos generados que nos interesan y que tenemos que modificar para lograr el objetivo.

Si abrimos el archivo app.js que creó el generador podemos encontrar dos líneas que especifican que archivo procesará la petición HTTP de la raíz del sitio:

Requerimos el archivo index que se encuentra en la carpeta routes:

```
var routes = require('./routes/index');
```

Llamamos al método use del objeto app que se creó previamente y le pasamos la ruta raíz y la referencia del módulo que requerimos previamente:

```
app.use('/', routes);
```

No hacemos ningún cambio en el archivo app.js, es lo generado por express-generator.

Dentro de la carpeta routes encontraremos dos archivos que dejó el generador, el archivo index.js y el archivo users.js.

Dentro de la carpeta views quedan las vistas que son las que contendrán el código html que se mostrara en el navegador.

Partiendo de esta base, los archivos de estas carpetas serán los que tendrán que ir modificando o agregando nuevos según la aplicación que quieran construir. Ya se van a ir dando cuenta como a medida que avancemos con los ejemplos.

En la carpeta routes el código original de nuestro archivo index.js tiene el siguiente código:

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

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

En este archivo se llama a la vista pasándole un objeto literal para que sea incluido en el html.

El archivo index.hbs original tiene el siguiente código para mostrar el title en la página HTML:

```
<h1>{{title}}</h1>
<p>Welcome to {{title}}</p>
```

Modificamos index.js para que pase los datos de los artículos que hay que mostrar:

```
var datos={
  titulo:'Articulos disponibles a la fecha',
  articulos: [
    { codigo: 1,precio:12,descripcion: 'peras' },
    { codigo: 2,precio:132,descripcion: 'manzanas' },
    { codigo: 3,precio:23,descripcion: 'naranjas' },
```



```
],
  descuento:{lunes:'5%',martes:'10%'}
};

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

Y modificamos el archivo index.hbs para que muestre los datos recibidos:

```
<h1>{{titulo}}</h1>
<table border="1">
  <tr>
    <td>Codigo</td><td>Descripcion</td><td>Precio</td>
  </tr>
  {{#each articulos}}
    <tr>
      <td>{{codigo}} </td> <td>{{descripcion}}</td> <td>{{precio}}</td>
    </tr>
  {{/each}}
</table>
<p>Descuentos vigentes el día lunes:{{descuento.lunes}}</p>
<p>Descuentos vigentes el día martes:{{descuento.martes}}</p>
```

## ABM y listado de una tabla SQL

Este ejemplo lo encuentran en la carpeta mysql.

Vamos a hacer lo mismo que en el ejemplo anterior, creamos la estructura con express-generator e instalamos todas las dependencias, luego vamos a instalar el módulo para comunicarnos con Mysql desde la línea de comandos:

```
npm install mysql --save
```

Luego de esto ya tenemos instalado en la carpeta node\_modules el paquete mysql y mediante la directiva --save modificamos el archivo package.json agregando la nueva dependencia.

Vamos a modificar el archivo index.hbs de la carpeta views para agregarle un menú de opciones:

```
<a href="/articulos/creartabla">Creacion de una tabla 'articulos' con MySQL</a></p>
<a href="/articulos/alta">alta de articulos</a></p>
<a href="/articulos/listado">Listado completo de articulos</a></p>
```





```
<a href="/articulos/consulta">Consulta de un articulo por codigo</a></p>  
<a href="/articulos/modificacion">Modificacion de un articulo</a></p>
```

Tenemos que agregar en el archivo app.js la ruta del paquete artículos que será el encargado de manejar la lógica relacionada a los artículos. Agregamos entonces estas dos líneas:

```
var articulos = require('./routes/articulos');  
  
app.use('/articulos', articulos);
```

En el archivo routes/bd.js creamos un módulo con la conexión a la base de datos. Utilizamos:

```
module.exports=conexion;
```

para poder incluir este archivo en artículos.js como vimos en las unidades anteriores.

El código de artículos.js lo encuentran comentado para comprender su funcionamiento.

## Ejemplo ABM de una tabla SQL con plantillas JADE

En la carpeta mysql-jade encontrarán un ejemplo similar al anterior, de un ABM en una tabla sql, haciendo uso también del módulo mysql y utilizando como plantillas JADE en lugar de Handlebars.

Para poder probarlo tendrán que crear la base de datos basenode en Mysql e importar el archivo universidades.sql para crear la tabla.

El código es muy sencillo y similar al anterior. También, como podrán observar, independientemente de la plantilla con la cual decidan trabajar, el mecanismo es siempre el mismo, lo que varía es la sintaxis de cada plantilla.

## Ejemplo: CRUD con Node.js y plantillas JADE

Como ya deben saber CRUD (Create, Read, Update, Delete) es el acrónimo de Crear, Leer Actualizar y Borrar. Llamamos comúnmente aplicación CRUD a aquella que puede hacer estas 4 operaciones sobre una tabla de una base de datos.

En el ejemplo anterior utilizamos ABM (Alta, Baja y Modificación), suele utilizarse también ABMC (Alta, Baja, Modificación y Consulta).

En este ejemplo realizaremos las altas, bajas, modificaciones y consulta sobre una tabla de clientes.

Dentro de la base de datos basenode en MySQL crearemos la tabla customer.

Dentro del directorio en el cual colocaremos la aplicación creamos la estructura de directorios necesarios para trabajar, public, routes, views y la carpeta node-modules que contiene los paquetes o módulos instalados a través de npm install.

En esta oportunidad además de express, mysql y jade vamos a utilizar el módulo express-myconnection, es un módulo basado en mysql, y que nos brinda algunas facilidades en el acceso a la base de datos.

Al igual que en lo ejemplos anteriores en el archivo app.js definimos los módulos propios que vamos a usar:

```
var routes = require('./routes');  
  
var customers = require('./routes/customers');
```

y asociamos cada ruta con el módulo que va a tratar cada una:

```
app.get('/', routes.index);  
app.get('/customers', customers.list);  
app.get('/customers/add', customers.add);  
app.post('/customers/add', customers.save);  
app.get('/customers/delete/:id', customers.delete_customer);  
app.get('/customers/edit/:id', customers.edit);  
app.post('/customers/edit/:id', customers.save_edit);
```

Para probarlo levantamos app.js y tipeamos en el navegador: localhost:4300/customers

Recuerden que para levantar la aplicación pueden tipear:



```
node app
```

o si instalaron nodemon de manera global como lo hicimos en la anterior unidad pueden tipear:

```
nodemon app
```

De esta forma si hacen cualquier cambio en la aplicación no necesitarán bajar y levantar app.js para que se vean reflejados los cambios.

Al tipear en el navegador localhost:4300/customers es capturada la ruta en nuestro app.js:

```
app.post('/customers', customers.list);
```

entonces se ejecuta el módulo list definido dentro del archive routes/customers.js que accede a la tabla customer y envía los datos a la plantilla para que los muestre por pantalla. Renderizamos la plantilla customers.jade de la carpeta views.

Recordemos que la sintaxis de las plantillas jade está basada en las tabulaciones y que para imprimir un dato recibido en la plantilla lo hacemos a través de #{nombre\_variable}.

La lista de usuarios presenta tres botones para dar de alta un cliente nuevo, para editar o borrar un cliente.

Al presionar el botón de alta redireccionamos customers/add, la ruta es capturada:

```
app.get('/customers/add', customers.add);
```

Y se ejecuta el módulo add que muestra la plantilla add\_customer.jade que contiene el formulario para dar de alta el cliente.

Una vez cargados los datos del cliente, cuando el usuario submite el formulario se captura la ruta:

```
app.post('/customers/add', customers.save);
```

que ejecuta el módulo save para dar de alta los datos en la tabla y luego redireccionamos a la lista de clients.

```
res.redirect('/customers');
```

En los vínculos para editar y dar de baja un cliente, pasamos por get en ambos casos el id del mismo. Ese parámetro es tomado en sus respectivos módulos y utilizado, en el caso de la edición para recuperar los datos de la tabla y mostrarlos en un formulario para permitir la modificación y en el caso de la baja es utilizado para hacer el delete de los datos del cliente en la tabla.

El manejo de las rutas, la ejecución de los módulos y la carga de las plantillas es exactamente igual que como lo explicamos en al alta.



## **Lo que vimos:**

---

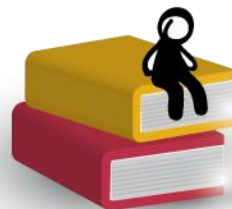
Aprendimos a utilizar el framework Express con plantillas JADE y Handlebars. También Incorporamos el módulo mysql para acceso a una base de datos



## **Lo que viene:**

---

En la próxima unidad veremos cómo trabajar con variables de sesión y cookies y algunos ejemplos prácticos con el uso de nuevos módulos que nos resultarán de utilidad en nuestro trabajo cotidiano.



## Bibliografía

---

- Alex Young y Marc Harter (2014). Node.js in practice. Manning
- Barsarat Ali Syed (2014). Begginig Node.js. Apress.
- Ethan Brown (2014). Web Development with Node & Express. O'Reilly
- Gimson, Matthew. (2015). Practical Guide for Begginers. Kindle Edition.
- Sitios web de referencia y consulta:
  - <https://nodejs.org/api/index.html>
  - <https://www.npmjs.org/>
  - <http://expressjs.com/es/>
  - <http://www.desarrolloweb.com/manuales/manual-nodejs.html>
  - <http://www.javascriptya.com.ar/nodejsya/>
  - <https://www.gitbook.com/book/ewiggin/expressjs-middleware/details>