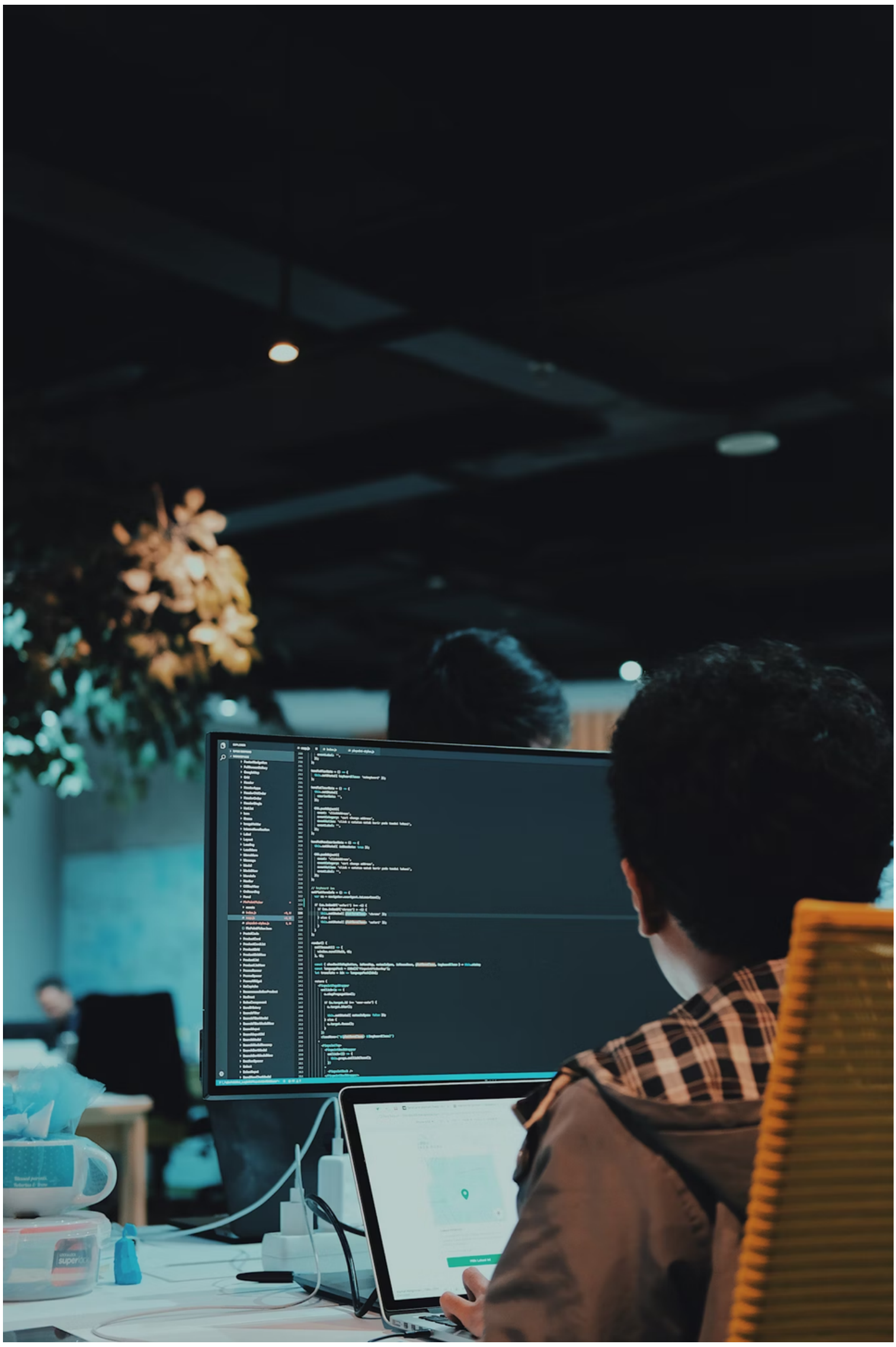
# Lección 3:
# Rutas y Vistas



**Planteamiento de la sesión**

**Tiempo de ejecución: 4 horas**

**Materiales:**

- EJS
- https://github.com/mde/ejs/wiki/Using-EJS-with-Express
- ejs - npm

**Motor de Vistas**

Esta lección se centra en una de las letras del MVC, y son las Vistas, para los proyectos a trabajar se va a configurar el motor de vistas EJS como motor por defecto a la hora del renderizado de vistas.

Revisando el proyecto creado con Express Generator, en su momento de inicialización, se indicó con la línea

express --view=ejs nombre_app

Que el motor con el que se quiere iniciar el proyecto sería EJS, se revisarán detalles del mismo más adelante.


La configuración de un motor u otro de los soportados por Express no es diferente, lo que cambia en sí, es la escritura del código “HTML” que lleva cada uno a la hora de renderizar la información, algunos distan mucho de lo que es código HTML y puede generar un reto a la hora de aprender la herramientas más una nueva notación del lenguaje. Sin embargo EJS comparte todos los elementos de HTML y solo agrega una capa adicional de una especie de JS entre etiquetas o marcas especiales en medio del resto de código.

Un motor permite generar vistas que contengan información y estructuras dinámicas. Se utilizan cuando se necesita mostrar CONTENIDO DINÁMICO. Por un lado están los datos y por el otro el template, donde se definen los bloques de contenido que luego el template irá rellenando con datos variables.


Los beneficios de utilizar un motor como EJS son:

- Fomenta la organización del código fuente en capas, separando responsabilidades.
- Permite reutilizar partes de vistas que se repiten como el header, footer, menú, productos, etc.
- Evita modificaciones recurrentes cuando la información cambia de manera constante.
- Permite usar más fácilmente JavaScript embebido en el HTML.

**EJS**

En el caso de no trabajar con un proyecto generado, para el ejercicio de instalar y configurar cada una de las dependencias, el proceso es muy sencillo, lo primero es instalar EJS en el proyecto.

los pasos detallados junto con ejemplos pueden encontrarse en la wiki de github del proyecto. https://github.com/mde/ejs/wiki/Using-EJS-with-Express

npm install ejs

Luego de instalado, se puede verificar en el archivo package.json del proyecto, se le debe indicar a Express que se trabajara con el motor instalado. Para ello se utiliza el método .set('view engine', 'motor'); en la variable en la que se haya inicializado el proyecto, normalmente la variable app del index.js o app.js.

app.set('view engine', 'ejs');

Ahora solo falta una configuración recomendada más y es definir la carpeta donde se almacenarán todas las vistas, esto permite que al hacer la referencia a un nombre de vista, el aplicativo cuente con la configuración para saber donde ubicarla. Para esa configuración, se usará el paquete path.

Lo primero es crear la carpeta views en la ubicación deseada, siguiendo las recomendaciones que se mencionaron anteriormente, se puede crear al nivel de la raíz.

Luego de creada la carpeta, puede crear dentro un primer archivo index.ejs para probar luego de finalizada la configuración, que todo haya quedado correcto.

```
<html>
  <body>
    Hello <%= foo %>
  </body>
</html>
```

Ahora, nuevamente en el entry point, se procede a configurar la carpeta, lo primero es que se va a utilizar el paquete path, se debe importar

```
...
var path = require('path');
...
```

Luego, utilizando el método .set(), se establece la ubicación de la carpeta

```
app.set('views', path.join(__dirname, 'views'));
```

La variable __dirname, es nativa de Node yhace referencia a la ruta del proyecto desde la raíz hasta la ubicación donde se haga el llamado, en este caso hasta la raíz del proyecto, el método path.join, une la ruta entregada por el __dirname con el nombre de la carpeta que asignamos para las vistas, en este caso views.

O la opción sin utilizar path

```
app.set("views", __dirname + "/views")
```

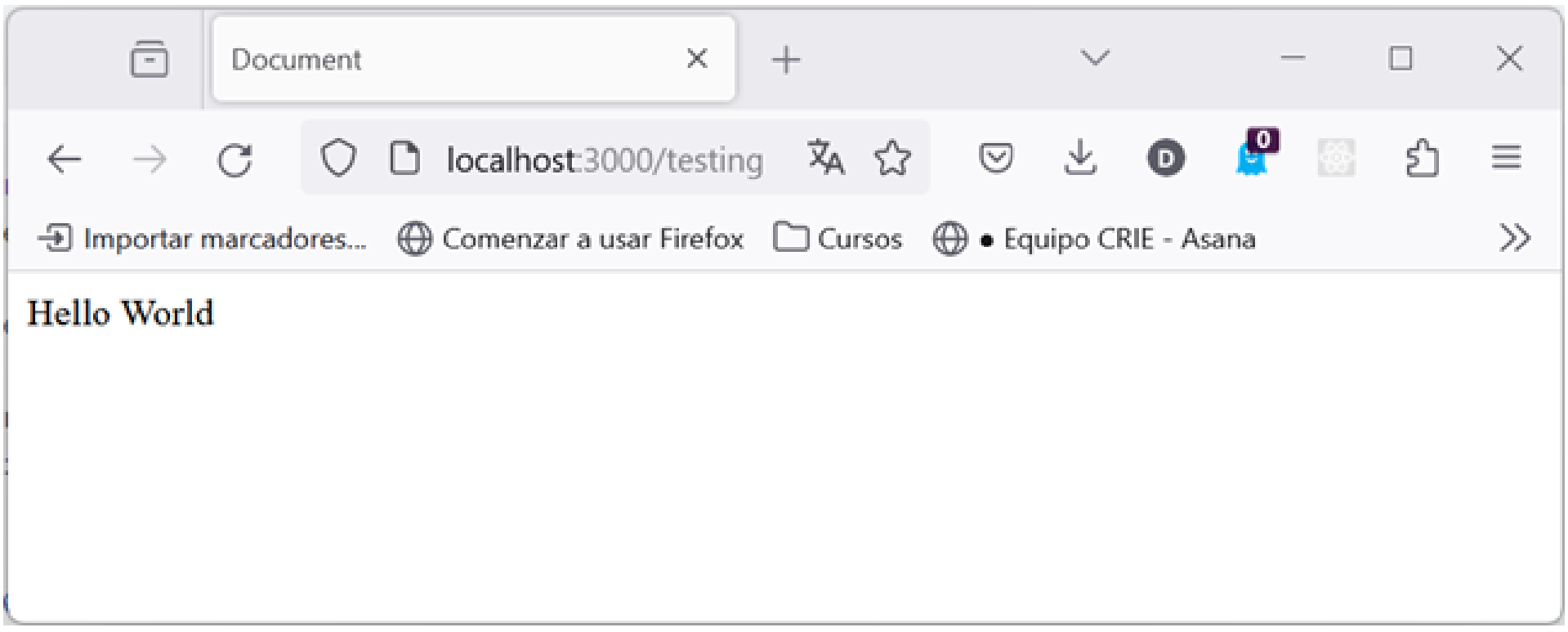
Ahora solo falta probar que la vista funcione correctamente, para ello, por ahora, desde el entry point se debe crear una ruta que responda la petición con un método llamado render, que recibe el nombre de la vista que debe mostrar en pantalla, algo como lo siguiente

```
app.get('/testing', (req, res) => {
 res.render('index');
});
```

Si se ejecuta el proyecto y se navega a la ruta creada debe cargar un error indicando que algo llamado “foo” no está definido. Esto es debido a que en el momento que se creó el archivo .ejs junto a la palabra “Hello” se utilizo una etiqueta especial propia de ejs <%= %>, esta etiqueta se usa para imprimir una variable, en el caso del ejemplo, una variable llamada foo que se debe enviar a la vista, que a propósito no se envió, para corregir este error, hay que volver a la ruta (tene presente que parte del código que se tiene allí se moverá en un futuro a un controlador) y luego de la ruta, como segundo parámetro del método .render, se debe enviar la variable requerida, algo como lo siguiente

```
app.get('/testing', (req, res) => {
 res.render('index', {foo : "World"});
})
```

Ahora al reiniciar el servidor, todo debe funcionar sin problema, imprimiendo en pantalla en mensaje “Hello World” desde la ruta y vista definidas.



**Bases de EJS**

EJS funciona de manera sencilla y cuenta con unas pocas opciones o etiquetas propias que ayudan a definir ciclos, flujos o variables a la hora de estar presentando la información.

Dos herramientas que apoyan este proceso es, como siempre, la documentación oficial de npm ejs - npm y una extensión para VS Code llamada EJS language support, EJS language support - Visual Studio Marketplace

La primer etiqueta que se necesita es `<%%>`. Es una sentencia del control del flujo.

```
<% for(let i = 1; i <= 0; i++) { %>

<!-- Entre la apertura y el cierre del tagpuede ir cualquier
contenido HTML -->

<% } %>
```

Para imprimir de manera puntual dentro del HTML <%= %> el igual le indica a ejs que lo que esté ahí adentro tiene que ser impreso de manera literal dentro del HTML, por lo que se puede usar para cualquier sentencia de JavaScript que sea dinámica, por ejemplo la variable i del for del ejemplo, entonces cuando se imprima la vista estará el resultado de la variable i en cada iteración del ciclo.

```
<% for (let i = 1; i <= 0; i++) { %>

<%= i %>

<% } %>
```

Este tag puede estar contenido por cualquier tag tradicional de HTML. De esta manera, su contenido interno (de la etiqueta HTML) estará relacionado con el valor dinámico existente dentro de la etiqueta de valor dinámico de ejs.

```
<ul>
<% for( let i = 0; i < 10; i++ ) { %>
<li>
 Soy el elemento # <%= i + 1 %>
 </li>
 <% } %>
</ul>
```

**Envío de parámetros**

Una de las mejores funcionalidades de una vista es que tiene la posibilidad de consumir información que sea suministrada directamente por el controlador, o de rutas como en los ejemplos presentados.

Básicamente, desde el controlador, vamos a poder compartir con la vista cualquier tipo de dato existente en JavaScript. Objetos, arrays y hasta funciones.
Se le pasa al método render, un segundo parámetro que será un objeto literal.

```
 app.get('peliculas', (req, res) => {
   res.render('peliculas', { listaPeliculas: nombrePelicula});
              }
}
```

**¿Cómo recibe los datos la vista?**

Para poder renderizar la variable enviada desde el controlador, la encerramos con la etiqueta de impresión: <%=  %>

Como `listaPeliculas` es una variable de JavaScript, podemos usar todos los métodos disponibles, sabiendo que es un string, podríamos escribir algo como:

```
<div>
   <h1>Película título:</h1>
   <span><%= listaPelicula %></span>

</div>
```

o enviar más de un dato a la vista e imprimirlo en un ciclo:

```
const listaPeliculas = ["Deadpool", "The Joker", "Batman"];

 app.get('peliculas', (req, res) => {
 res.render('peliculas', {
 listaPeliculas,
 enOferta: true,
 genero: "acción",
 })
});

<% for (let pelicula of listaPeliculas) { %>
   <h2>El título de lapelícula es <%= pelicula %> y el género es
   <%= genero %>
   </h2>
   <% if(enOferta){ %>
            <p>25% dedescuento alquilando esta película</p>
   <% } %>
<% } %>
```

**Vistas parciales**

Cuando desarrollamos un sitio web, es bastante común tener ciertos elementos visuales que suelen repetirse constantemente. Por ejemplo, una barra de navegación o el pie de página son de uso habitual y casi siempre son el mismo en todas las vistas. EJS cuenta con una funcionalidad que permite facilitar esta labor.

EJS provee una manera de modularizar nuestra estructura HTML y generar bloques de código que se repiten, los cuales pueden ser fácilmente incrustados en el resto de las vistas, logrando simplificar muchísimo el trabajo de desarrollo y programación de nuestro proyecto web.

Lo primero para modularizar, es identificar cuál va a ser el componente o bloque de código que se repite en las vistas, por ejemplo el header con su menú o el footer con los datos de contacto.

Lo siguiente es llevar ese bloque de código a un archivo EJS aparte, comúnmente, se crea una carpeta dentro de views llamada /views/partials para almacenar los bloques parciales de las vistas. Debe tener presente que si el bloque de código recibe un parámetro a la hora de renderizar, como por ejemplo que el nombre de la página en la etiqueta title se ponga de forma dinámica, es necesario envíar el parámetro a la hora de invocar el partial.
Cómo Insertarla?

EJS nos da una función que se llama include. Para que funcione es necesario utilizar los tags de EJS. Es necesario agregar un guión antes de la función include.

```
<%- include(‘partials/navbar’) %>
```

```
<!-- views/partials/navbar.ejs -->
<div class="header clearfix">
            <nav>
           <ul class="nav nav-pills pull-right">
            <li role="presentation"><a href="/">Home</a></li>
           </ul>
           <h3 class="text-muted">Node.js Blog</h3>
            </nav>
</div>
```

```
<div class="container">
            <%- include('partials/navbar') %>
            <div class="jumbotron">
           <h1>All about Node</h1>
           <p class="lead">Check out our articles below!</p>
            </div>
</div>
```

En el caso de requerir parámetros entonces la invocación deberá incluirlos

```
<!-- views/partials/navbar.ejs -->
<div class="header clearfix">
```

```
 <nav>
 <ul class="nav nav-pills pull-right">
 <li role="presentation"><a href="/">Home</a></li>
 </ul>
 <h3 class="text-muted"><%= title %></h3>
 </nav>
</div>

<div class="container">
 <%- include('partials/navbar', {title: Node.js Blog}) %>
 <div class="jumbotron">
 <h1>All about Node</h1>
 <p class="lead">Check out our articles below!</p>
 </div>
</div>
```