Git Ignore

Como ahora la instalación de paquetes de terceros nos crea la carpeta de node_modules que contiene mucha información, vamos a excluirla de git mediante un archivo que se llama .gitignore (es un archivo oculto).

.gitignore

node_modules/

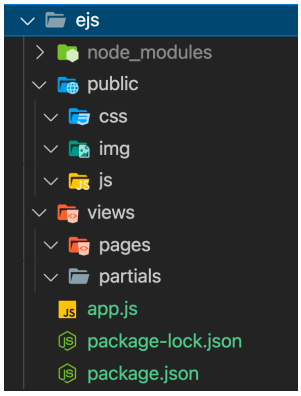
Guía paso a paso

Vamos a crear la estructura de un e-commerce **dinámico**. Para eso vamos a estar usando Node JS y EJS como sistema de témplales dinámicos.

Paso 1 - Iniciar un proyecto con npm

1. Creamos una carpeta nueva que se llame “ecommerce-ejs”
2. Creamos un archivo app.js
3. Ejecutamos el comando npm init.
4. Seguimos los pasos y configuramos el package JSON

Paso 2 – Estructura de Carpetas



1. Creamos una carpeta llamada “views”
    1. Creamos una subcarpeta llamada “partials"
    2. Creamos una carpeta llamada “pages”
2. Creamos una carpeta llamada “public”
    1. Creamos las subcarpetas css, js e img

Paso 3 – Servidor con Express

1. Instalamos express a nuestro proyecto con el comando npm i express
2. Lo requerimos e inicializamos el servidor

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

app.listen(3000, () => {
    console.log('Puerto 3000');
})
```

Paso 4 - Configurando EJS

Tenemos que indicar dónde van a estar las página y dónde van a estar los partials.

1. Configuramos el path
    1. Requerimos el paquete path
2. Configuramos el view engine
    1. Utilizamos el paquete path y decimos dónde van a estar las views

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

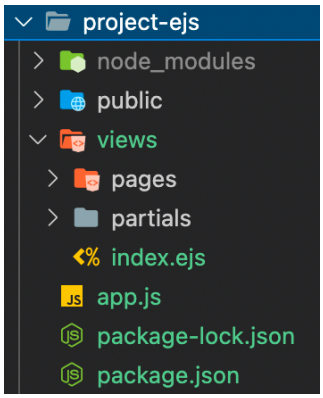
3. Configuramos express para que use EJS
    1. Instalamos EJS con *npm i ejs*
    2. Seteamos la View Engine

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

4. Creamos dentro de nuestra carpeta views, nuestro primer archivo index.ejs

5. Utilizamos el app.get para renderizar el index.ejs que creamos

Todo junto:




```
const express = require('express');
const path = require('path');
const app = express();

app.set('views', path.join(__dirname, 'views'));
app.set('view engine',  'ejs');

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

app.listen(3000, () => {
    console.log('Puerto 3000');
})
```

Paso 5 - Nuevas páginas con EJS

Vamos a crear nuevas páginas. Dentro de la carpeta *views,* dentro de la carpeta *pages* vamos a crear una página de contacto.

1. Creamos la página contacto.ejs
2. Creamos la llamada a la página de contacto. Noten que ahora la página está dentro de la carpeta *pages.*

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

Paso 6 - JavaScript y CSS

Estamos creando páginas ejs, y como tal tenemos que poder estilarlas con CSS y JS.

Nos habíamos creado una carpeta llamada *public* cuya función es alojar los archivos de dominio público como los CSS y JS del lado del cliente.

1. Vamos a crear un *style.css* dentro de la carpeta CSS
2. Vamos a vincularla en nuestro *ejs* como ya sabemos, mediante la etiqueta link.. la pregunta es ¿Qué ruta usamos?

```
<link rel='stylesheet' href='/css/style.css' />
```

```
<link rel='stylesheet' href='../css/style.css' />
```

Como dijimos antes, la carpeta *public* va a servir archivos estáticos. Para lograr poder servir esos archivos estáticos, debemos decirle a Express dónde van a estar alojados y que - efectivamente -los use como archivos de dominio público.

Paso 7 - Archivos estáticos

Cuando queremos tener una carpeta de archivos públicos, así hay que indicárselo a express mediante el método .static.

```
app.use(express.static(path.join(__dirname, 'public')));
```

En esta línea le estamos diciendo a nuestra app que use archivos públicos que se encuentran dentro de la carpeta public.

Una vez agregada esta línea de código (en las configuraciones de nuestra app) la configuración de la carpeta pública está sobre public, por lo tanto el link nos quedaría de la siguiente manera:

```
<link rel='stylesheet' href='/css/style.css' />
```

Todo junto:

App.js

index.ejs

```
<link rel='stylesheet' href='/css/style.css' />
```

```
const express = require('express');
const path = require('path');
const app = express();

app.set('views', path.join(__dirname, 'views'));
app.set('view engine',  'ejs');
app.use(express.static(path.join(__dirname, 'public')));

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

app.get('/contacto', (req,res) => {
    res.render('pages/contacto')
});

app.listen(3000, () => {
    console.log('Puerto 3000');
})
```

Paso 8 - Partials

Muchas veces vamos a necesitar que haya componentes que se repitan a lo largo de las páginas, como por ejemplo la barra de navegación o el footer. Para esto existen los partials.

Si tenemos 3 páginas

- Principal
- Contacto
- Productos

Es muy probable que las tres páginas usen la misma nav. Vamos a crear un header que contenga la nav y vamos a utilizarla como partial.

```
<header>
  <nav>
    <ul>
      <li>
        <a href="#">Prinicipal</a>
      </li>
      <li>
        <a href="#">Contacto</a>
      </li>
      <li>
        <a href="#">Historia</a>
      </li>
    </ul>
  </nav>
</header>
```

1. En la carpeta *partials* vamos a crear un nuevo archivo **header.ejs** pero esta vez no vamos a crear todo el HTML, sino solamente una sección. En nuestro caso el header.
2. Para incluir este partial en otros archivos, simplemente lo que hacemos es buscar el archivo .ejs que va a integrar nuestro partial y hacer el <%- include('ruta') %>

Ejemplo en el index.ejs

```
<%- include('./partials/header') %>
```

Si quisiéramos incluir un partial en otra ubicación, por ejemplo dentro de la carpeta *pages* en el archivo de contacto que creamos, lo que tendríamos que hacer es cambiar la ruta relativa.

```
<%- include('../partials/header') %>
```

Paso 9 - Objetos en EJS

Ya creamos partials y sabemos que podemos tener bloques de código separados para reutilizar en diversas páginas o incluso de otros partials.

Muchas veces hay valores que no cambian en la web, como por ejemplo el título de la página. Si pensamos en la estructura de web que estamos armando que tiene 3 páginas:

- Principal
- Contacto
- Productod

Todas van a tener el mismo title y, preferentemente, la sección en la que se encuentran.

E-commece

Contacto | E-commerce

Historia | E-commerce

Cuando nuestro servidor da respuestas en el res, podemos enviarle información al EJS, como por ejemplo el nombre de título de nuestra web. Para esto lo que podemos hacer en el el *res.render*, pasar como segundo parámetro un objeto **con la información que queremos disponible en el EJS**

```
app.get('/', (req,res) => {
    res.render('index', {
        title: 'E-commerce'
    })
});
```

Lo que estamos haciendo en esta línea es pasarle un objeto al el EJS, solo nos queda usarlo, y para esto, lo que hacemos es imprimir el valor utilizando uno de los tags disponibles en EJS

```
<%= title %>
```

En uso:

```
    <title><%= title %></title>
```

```
app.get('/', (req,res) => {
    res.render('index', {
        title: 'E-commerce',
        test: {
            name: 'Prueba'
        }
    })
});
```

```
<p>
    <%= test.name %>
</p>
```

¿Puedo pasar un objeto más complejo?

Paso 10 - Arrays en EJS

De a poco vamos construyendo una página, del lado del servidor, que tiene valores desde el BackEnd y se inyectan en el HTML que luego ve el cliente… Con el tiempo estos valores van a venir de una base de datos y el flujo va a quedar completo.

Pensemos que tenemos una lista de productos y queremos que se muestren estos productos, podemos pensar que hacemos una consulta a la base de datos, nos devuelve un array de productos, nosotros pasamos ese array de productos al Front End y con EJS construimos múltiples cards.

Vamos a crearnos un *custom module* que contenga una lista de productos y la vamos a exportar.

1. Creo una carpeta llamada *utils*
2. Creo un archivo llamado *products.js* que va a tener una lista con los productos de nuestro e-commerce.
3. Exporto la lista y la importo en mi app.js con el require del custom module.
4. El index, además del title, va a tener un objeto products que va a tener el array importado.

Todo junto

products.js

```
const products = [
  {
    id: 0,
    name: 'Mini Pymer',
    price: 2700,
    color: 'white'
  }
]

module.exports = products;
```

app.js

```
const products = require('./utils/products');
```

```
app.get('/', (req,res) => {
    res.render('index', {
        title: 'E-commerce',
        products
    })
});
```

Ahora solo queda utilizar ese array de productos en nuestro ejs. Para utilizar un array en nuestro ejs lo que hacemos es usar el tag de scrips <%%> y utilizamos una función foreach.

```
<% products.forEach(product => { }) %>
```

La única diferencia, es que lo que queremos nosotros es que se renderice HTML (o ejs en nuestro caso) por cada vuelta, por eso lo que terminamos haciendo es pensar en esta estructura en bloques.

```
<% products.forEach(product => { %>
        <h3><%= product.name %></h3>
    <% }) %>
```

Línea 1: abro y cierro sentencia de forEach
Línea 2: Creo el ejs que necesito y pinto el valor
Líne 3: Cierro la sentencia

Paso 11 - A tener en cuenta

La idea al construir nuestro sitio con EJS no es manipular los objetos y los arrays dentro del EJS, sino antes. Es por eso que todo lo que tenga que ver con manipulación de arrays y objetos se suele hacer antes, del lado de BE, para luego finalmente pasárselo a ejs ya procesado y poder construir una página dinámica.