



**Argentina  
programa  
4.0**



Ministerio de Economía  
**Argentina**

Secretaría de  
Economía del Conocimiento

*primero  
la gente*

## Clase 5: Motores de Plantillas

# Presentación general del curso

**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

Temas

# Agenda de hoy

- A. Motores de plantillas
  - a. El concepto de Server Side Rendering
  - b. Diferencias entre SSR y Plantillas web
  - c. Plantillas web
    - i. cuál elegir (EJS)
- B. EJS: Crear el proyecto base
  - a. Configuración
  - b. Sintaxis
  - c. Implementación
  - d. Dinamizar contenido
- C. Prácticas



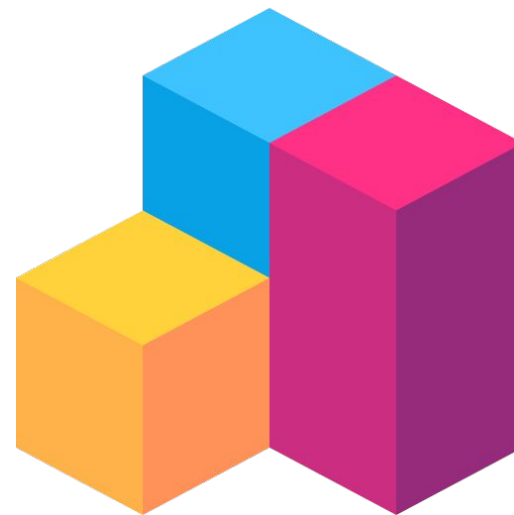
# Motores de plantillas

## Motores de plantillas

**En esta unidad, nos alejaremos brevemente del trabajo con aplicaciones de backend netamente de servidores, para conocer lo que son motores de plantillas.**

Estas herramientas están enfocadas en el desarrollo Frontend, pero implementando las mismas desde el backend.

Veamos entonces de qué se trata:



# **Motores de plantillas el concepto SSR**

## El concepto SSR

El paradigma **Server-Side Rendering** implica generar contenido HTML dinámico en el servidor y enviarlo al cliente como respuesta a una solicitud.

En este caso, el servidor mezcla HTML estático con código dinámico, todo en el servidor web utilizando un framework que permita SSR.





## El concepto SSR

En un sitio web conformado por varios archivos HTML, CSS y JS, la lógica de generar bloques HTML dinámicos, con acciones específicas de JS, y demás cuestiones de diseño suele interpretarse siempre del lado del cliente.

Esto se suele denominar **Client-Side Rendering**. El navegador web del cliente recibe los archivos mencionados, interpreta el código renderizando la estructura que se debe ver en el webbrowser para, finalmente, “*pintar*” la misma en la pestaña del navegador.



## El concepto SSR

**Client-Side Rendering** pone a trabajar al navegador y, sobre todo, a JS. Interpreta el código, si hay peticiones a servidores remotos las realiza, las recibe, las interpreta el motor y luego se muestra todo el resultado en el documento HTML en cuestión.

**El trabajo de procesamiento lo realiza la computadora del cliente. El servidor web le envía los documentos y recursos asociados “*en crudo*”.**



## El concepto SSR

En el modelo **Server-Side Rendering**, el webserver recibe una petición a una ruta del servidor, ejemplo:

(<https://servidor.com/productos>)

Dicha petición es analizada por el webserver, quien recopila la información de productos de una bb.dd., arma el HTML con la información a mostrar para, finalmente, enviar al cliente web un HTML resultante, el recurso CSS asociado y, tal vez, algún recurso JS con funcionalidades básicas.



## El concepto SSR

Toda la carga que vemos aquí, la absorbe el servidor web. Y ni siquiera es necesario crear diferentes archivos HTML para utilizar SSR.

En el ecosistema SSR frontend podemos encontrar frameworks web que nos permiten realizar estas tareas.

Entre los más conocidos, están:

- **Next.js** (*React en el servidor*)
- **Nuxt.js** (*Vue en el servidor*)
- **Gatsby** (*de los creadores de GraphQL*)



# El concepto SSR (ventajas)

**Aquí te presentamos algunas de las ventajas notables que podemos encontrar utilizando SSR.**

## HTML PURO

SSR permite generar HTML puro, CSS y algo muy básico de JS para generar interactividad básica en el documento HTML.

## SEGURIDAD

El concepto de seguridad mejora significativamente ya que nuestro servidor manda información pura, evitando servir archivos JS con tokens, y otros datos críticos, que permitirán a cualquier hacker, tener más herramientas para acceder a los datos alojados en los servidores.

## VELOCIDAD

Con un gran clúster de servidores detrás de un sitio o aplicación web que funciona con SSR, cada cliente obtendrá una performance notable en el acceso a la información dado que, al recibir casi todo HTML, la interpretación del mismo casi no requiere esfuerzo por parte del motor web del browser.

# **Motores de plantillas**

## **Diferencias entre SSR y plantillas web**

## Diferencias entre SSR y plantillas web

**Podemos definir al Motor de Plantillas como un mecanismo que genera contenido HTML dinámico que luego se envía al cliente como respuesta a una solicitud.**

Los motores de plantillas permiten mezclar HTML estático con código dinámico escrito en JavaScript, lo que hace posible generar contenido personalizado para cada solicitud.



## Diferencias entre SSR y plantillas web

En base a esta última descripción, podemos notar que **ambos enfoques comparten el mismo objetivo:**

- **generar contenido HTML dinámico en el servidor y enviarlo al cliente**, en lugar de generar contenido HTML estático en el cliente utilizando JavaScript en el navegador (*como ocurre con Client-Side Rendering*).



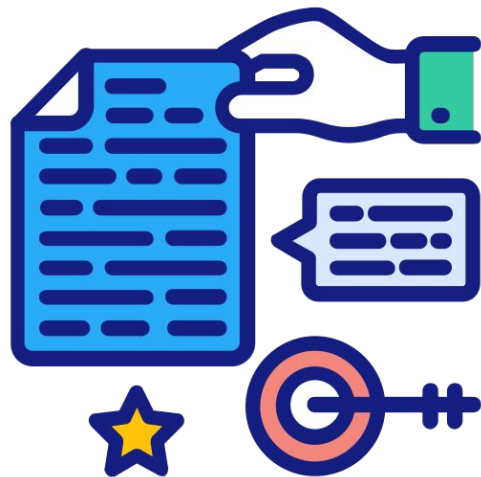


## Diferencias entre SSR y plantillas web

Aunque sí hay diferencias entre ambos:

- SSR está atado al uso de Frameworks JS como los mencionados anteriormente
- Los Motores de Plantillas son más laxos pero limitados por lo cual podemos prescindir de un Framework JS para realizar esta tarea

**Entonces, los Motores de Plantilla, sirven para realizar SSR aunque están pensados para procesos simples y no para trabajos más pesados o con mucha lógica de por medio.**



# Motores de plantillas

## Plantillas web

# Plantillas web

**Veamos, a continuación, algunas ventajas de porqué implementarlos:**

VENTAJAS	FUNDAMENTOS
<b>Facilitan la creación de web dinámicas</b>	Combinan plantillas predefinidas con datos dinámicos para generar así páginas web dinámicas. Esto hace más fácil y rápido crear páginas web personalizadas y dinámicas.
<b>Separan la lógica del negocio</b>	Separan la lógica ( <i>el HTML y la estructura de la página</i> ) del negocio ( <i>el código JavaScript que maneja los datos y las interacciones</i> ). Esto hace más fácil mantener y modificar el código.
<b>Reutilización de plantillas</b>	Permiten definir bloques reutilizables que se pueden utilizar en diferentes páginas web. Esto hace más fácil y rápido crear páginas web consistentes y con un aspecto similar.
<b>Mayor eficiencia</b>	Suelen tener un rendimiento mejor que la generación manual de HTML, ya que sus técnicas de caché y compilación permiten generar HTML de manera rápida y eficiente.
<b>Compatibilidad con diferentes sistemas</b>	Suelen ser compatibles con diferentes sistemas y frameworks, lo que permite utilizarlos en proyectos de <i>Node.js</i> , <i>Express</i> , <i>Meteor</i> , <i>Ruby on Rails</i> , <i>Django</i> , entre otros.

## Plantillas web

En Node.js existen varios motores de plantillas que se pueden utilizar para generar HTML dinámico en el servidor. Algunos de los más populares, son:

- EJS
- Handlebars
- Pug
- Mustache

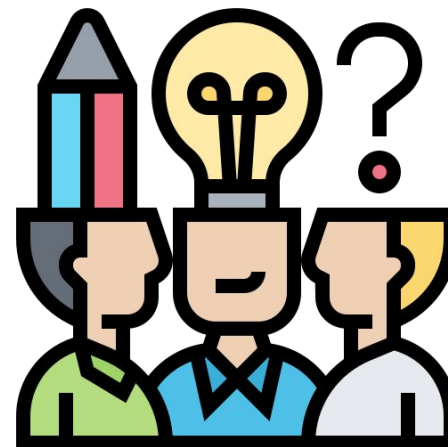
**Cada uno tiene sus propias ventajas y desventajas, y el mejor para utilizar, siempre depende de las necesidades puntuales del proyecto.**



## Cual elegir

**De todos los motores de plantillas disponibles, elegiremos EJS para trabajar. Este es el elegido porque ofrece una estructura fácil y práctica para intercalar HTML con JavaScript en el servidor.**

Entre el resto de motores de plantillas, algunos de ellos no incluyen esta funcionalidad, por lo tanto, se hace más complejo generar HTML dinámico a partir de un array de elementos u objetos en JS.



# Motores de plantillas

## Crear el proyecto base

(prácticas en clase)



## Crear el proyecto base

Para poder abordar un ejemplo, crearemos un proyecto Node.js desde cero, al cual llamaremos **motor-de-plantillas**.

- Instalaremos **Express, path** y **EJS**
- Configuraremos un webserver básico con Express que contenga dos rutas: (“/” y “/productos”)
- Definimos el **script start** utilizando el concepto de NPM automatizado aprendido la clase anterior



## Crear el proyecto base

**Nuestro webserver base deberá lucir más o menos como el siguiente snippet de código.**

Puedes incluir alguna otra característica que consideres necesaria, como por ejemplo, controlar peticiones a rutas inexistentes.

Esto también podrá controlarse aprovechando el Motor de Plantillas.

```
Express JS

const express = require('express');
const app = express();
const path = require('path');
const PORT = 3008

app.get('/', (req, res) => {

});

app.get('/productos', (req, res) => {

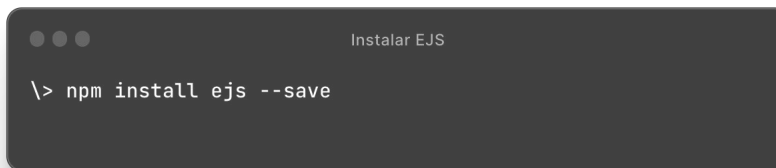
});

app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```



## Crear el proyecto base

**Para instalar EJS utilizamos, como siempre, la línea de comandos junto a NPM.**

A dark-themed terminal window with three window control buttons (red, yellow, green) in the top-left corner. The title bar reads "Instalar EJS". The command prompt shows the command `\> npm install ejs --save` entered.

```
Instalar EJS

\> npm install ejs --save
```

Finalizado el proceso, ya podremos utilizarlo desde nuestro webserver previamente estructurado.

# Motores de plantillas

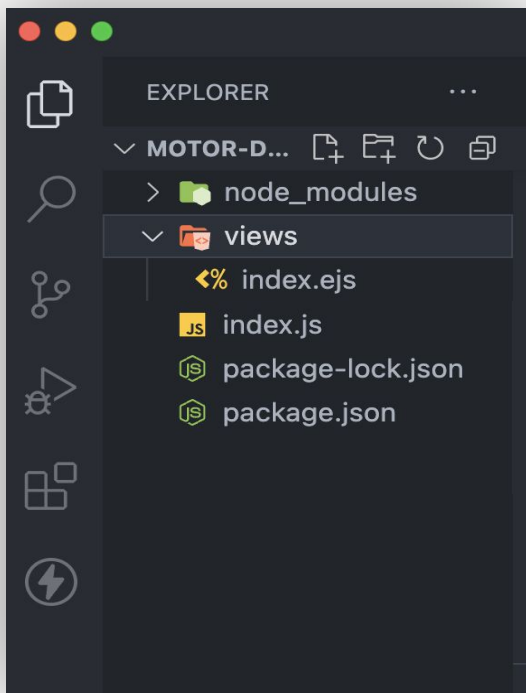
## Configuración

## Configuración

Con el archivo **index.js** ya creado y configurado, debemos inicializar el Motor de plantillas elegido. Para ello, debemos configurarlo utilizando el método **.set()** de nuestro servidor Express.

```
EJS  
  
app.set('view engine', 'ejs');  
app.use(express.static('views'));
```

# Configuración



Más allá de que gran parte del sitio se generará de manera dinámica, siempre serviremos un sitio estático y, es por ello, que usaremos el método **express.static()**.

Para alojar el contenido EJS, crearemos una subcarpeta llamada **views**. Aquí se alojan todas las vistas que conformarán la estructura del sitio web.

Dentro de esta carpeta, crearemos un documento HTML, aunque usaremos la extensión de archivo EJS (**index.ejs**).

# Configuración



Para tener una experiencia con el motor de plantillas más cercana al desarrollo frontend, sumaremos algunas herramientas “*que le den color*” a nuestra estructura HTML; por ejemplo, sumando un **favicon** al proyecto.

Te invitamos a descargar algún ícono de tu agrado de algún repositorio online como, por ejemplo, [www.flaticon.com](http://www.flaticon.com)

Guarda el mismo, con el nombre **favicon.png**, en la carpeta **views** del proyecto que estamos trabajando.

Luego sumaremos el microframework CSS Milligram, aunque este lo utilizaremos mediante su CDN.

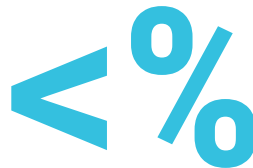
# Motores de plantillas

## Sintaxis

# Sintaxis

Veamos entonces, cómo es la estructura de un archivo EJS:

Tal como dijimos anteriormente, las plantillas de EJS se parecen a las páginas HTML normales, pero pueden incluir código JavaScript entre etiquetas `<% %>` para acceder a los datos.



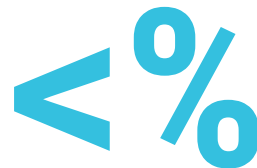
Para tener un punto de comparación, estas etiquetas especiales se comportan de forma similar al uso de *Template Literals* en JavaScript moderno: `${variable}`.

# Sintaxis

Su sintaxis se divide en dos partes:

- Uso de marcadores de posición para la inserción de datos
- Combinación de HTML y JavaScript en las plantillas de EJS

En sí, EJS toma en parte la filosofía que podemos encontrar en Librerías o Frameworks Frontend, pero de una forma más ligera. Además, nos ayuda a simplificar el HTML en gran medida, logrando así una performance efectiva para cargar documentos HTML resultantes con un muy buen tiempo de respuesta.





# Sintaxis

Aquí tenemos un ejemplo básico de la sintaxis de EJS.

Esta estructura va definida en el archivo EJS, y podemos ver que en la misma se intercalan los tags HTML, y dentro de las etiquetas EJS definimos una variable o propiedad JS, la cual desplegará dentro del nodo texto del documento HTML, el valor que tenga asignado.

```
Sintaxis

<h1><%= title %></h1>
<p><%= message %></p>
```

# Sintaxis

Si contamos con un array de objetos el cual, por ejemplo, tiene un listado de productos y queremos visualizar el mismo en el documento HTML, EJS nos permite agregar la estructura de etiquetas necesarias para poder iterar dicho array mediante un **ciclo for convencional**.

```
Sintaxis

<ul>
  <% for (let i = 0; i < products.length; i++) { %>
    <li><%= products[i].name %></li>
  <% } %>
</ul>
```

# Sintaxis

Todos los parámetros que le indicaremos a la plantilla EJS, deben estar generados dentro de una constante la cual se denomina **data**, bajo un formato de objeto literal. Desde allí, el Motor de plantillas se ocupará de armar la estructura HTML final, renderizando correctamente el contenido.

```
EJS

const data = {
  title: 'Mi sitio web con EJS',
  message: 'Bienvenido a mi sitio web generado a partir de un
motor de plantillas.'
};
```

# Sintaxis

**Veamos a continuación los diferentes marcadores de posición con los que cuenta EJS.**

MARCADOR	DESCRIPCIÓN
<code>&lt;%= %&gt;</code>	Este marcador se utiliza para insertar contenido en la plantilla. Cualquier expresión JavaScript válida se puede utilizar dentro de los marcadores, y su resultado se mostrará en la plantilla. Por ejemplo, si tienes una variable <b>title</b> con el valor " <i>Mi sitio web</i> ", puedes insertarla en la plantilla con <code>&lt;%= title %&gt;</code> .
<code>&lt;%- %&gt;</code>	Este marcador se utiliza para insertar contenido sin escapar en la plantilla. Esto significa que cualquier HTML o JavaScript incluido en los marcadores se mostrará tal cual, sin ser procesado por el navegador. Este marcador de posición es útil si necesitas insertar código HTML o JavaScript generado dinámicamente en la plantilla.
<code>&lt;% %&gt;</code>	Este marcador se utiliza para incluir código JavaScript en la plantilla. El código dentro de los marcadores se ejecutará cuando se procese la plantilla, pero no se mostrará en la salida final. Esto te permite realizar operaciones o lógica más complejas en la plantilla, como ciclos, condiciones, etc.

# Sintaxis

Para controlar el flujo de ejecución y generar contenido dinámicamente, EJS contiene una serie de comandos específicos que nos facilitan estas tareas.

## COMANDOS DE EJES

**`<% if (condition) { %> ... <% } %>`**: Permite ejecutar un bloque de código si se cumple una condición determinada. Puedes utilizar cualquier expresión JS como condición.

**`<% for (var i = 0; i < items.length; i++) { %> ... <% } %>`**: Permite ejecutar un bloque de código varias veces, en función del número de elementos en un arreglo o variable.

**`<% while (condition) { %> ... <% } %>`**: Permite ejecutar un bloque de código mientras se cumpla una condición determinada, usando cualquier expresión JavaScript como condición.

**`<% switch (variable) { case 'value': %> ... <% break; } %>`**: Permite ejecutar un bloque de código en función del valor de una variable, sobre una estructura switch similar a la de JS.

**`<%- include('filename') %>`**: Permite incluir otro archivo EJS dentro de la plantilla actual. Puedes utilizar esto para reutilizar bloques de código en varias plantillas.

# **Motores de plantillas**

## **Implementación**

# Implementación

Veamos entonces, cómo es la estructura de un archivo **EJS** implementando los marcadores de posición. Abrimos el archivo en cuestión y definimos los tags HTML básicos correspondientes al **<head>** del documento HTML.

```
index.ejs x
views > index.ejs > ul
1 <title>Mi motor de plantillas EJS</title>
2 <link rel="shortcut icon" href="/favicon.png" />
3 <meta charset="utf-8">
4
```

# Implementación

Seguido a dicha estructura, definimos los tags HTML correspondientes a **<body>**.

**En cada tag** donde sea necesario representar contenido a partir de JS, **intercalamos los marcadores de posición**, según lo que hayamos definido en el **objeto data**.

```
index.ejs  X
ews > <% index.ejs > link
5      <div class="container">
6          <h1><%= title %></h1>
7          <p><%= message %></p>
8          <button class="button">ver productos</button>
9      </div>
10     <footer>
11         <div class="footer">
12             <p><b>Copyright 2023 - Quien lo programa</b> 🤖 </p>
13         </div>
14     </footer>
```



# Implementación

```
EJS

<meta charset="utf-8">
<title>Mi motor de plantillas EJS</title>
<link rel="shortcut icon" href="/favicon.png" />

<div class="container">
  <h1><%= title %></h1>
  <p><%= message %></p>
  <button class="button">ver productos</button>
</div>
<footer>
  <div class="footer">
    <p><b>Copyright 2023 - Quien lo programa</b> 🤖 </p>
  </div>
</footer>
```

Como vemos en el ejemplo contiguo, en la estructura de un documento HTML creado a partir de EJS es posible definir los tags HTML que separan el **<head>** de **<body>**, sin agregar estos tags contenedores.

# Implementación

```
app.get('/', (req, res) => {  
  // res.send("Hello world!");  
  const data = {  
    title: 'Mi sitio web con EJS',  
    message: 'Bienvenido a mi sitio web generado a partir de  
un motor de plantillas.'  
  };  
  //aquí renderizamos el contenido  
});
```

En el archivo **index.js** correspondiente, dentro del apartado que escucha las peticiones en el raíz del sitio, definimos la estructura de la objeto **data** con **una propiedad y su valor, correspondiente a cada marcador de posición** a utilizar en la plantilla EJS.

# Implementación

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

En el bloque de código, justo antes del cierre del método **.get()**, sumamos al método **render()** correspondiente al objeto **response** (*res*).

Este método espera como **primer parámetro**, el archivo EJS que debe **renderizar**, y como **segundo parámetro**, el objeto **data** con la **estructura correspondiente** a los **marcadores de posición**.

# Implementación

Si en alguna parte de nuestro archivo JS tenemos un **array de objetos o elementos (1)** el cual se debe renderizar en un documento HTML, **lo podemos referenciar como una propiedad del objeto data (2).**

**Así podremos cargarlo de manera dinámica en el documento HTML/EJS correspondiente.**

```
EJS
const computerProducts = [{name: 'Notebook Lenovo', price: 720},
                           {name: 'Macbook Air 13', price: 1250},
                           {name: 'Tablet Droid 10.1', price: 350}]
```

```
EJS
app.get('/productos', (req, res) => {
  // res.send("Hello PRODUCTS!");
  const data = {
    title: 'Mi sitio web con EJS',
    message: 'Bienvenido a mi sitio web generado a partir de
un motor de plantillas.',
    products: computerProducts
  };
  res.render('productos', data);
});
```

# **Motores de plantillas**

## **Dinamizar contenido**

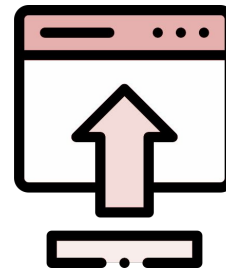
## Dinamizar contenido

Verifiquemos el último comando específico explicado en la diapositiva de Sintaxis de EJS, este se basa en la función **include()**.

Dicha función tiene un papel importantísimo en EJS, ya que nos permite generar un único encabezado **<head>** y un pie de página **<footer>** común a todas las plantillas EJS.

Esto nos permite, por ejemplo, definir un único archivo **head.ejs** y un archivo **footer.ejs** ambos con el contenido HTML predeterminado en estas secciones.

Luego, el resto de las plantillas EJS solo tendrán la estructura **body HTML**, y **agregaremos los apartados head y footer común a todas, mediante la función include()**.



# Dinamizar contenido

**Definimos una plantilla EJS con el nombre head.ejs.** Dentro de la misma, agregamos el contenido HTML correspondiente al encabezado del documento homónimo. Si usamos fuentes web y/o un framework CSS, aquí es donde debemos definir también los respectivos CDN.

```

<!DOCTYPE html>
<html lang="es">
  <meta charset="utf-8">
  <title>Mi motor de plantillas EJS</title>
  <link rel="shortcut icon" href="/favicon.png" />
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Roboto:300,300italic,700,700italic">
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/milligram/1.4.1/milligram.min.css">
</body>

```

## Dinamizar contenido

Luego hacemos lo propio con el archivo **footer.ejs**. En este caso, solo insertamos el contenido del pié de página, ya que el cierre del tag **<body>** podemos definirlo por cada documento *'cuerpo'* creado con EJS.



EJS

```
<div class="container">
  <p><b>Copyright 2023 - Quien programa</b> 🤖 </p>
</div>
```



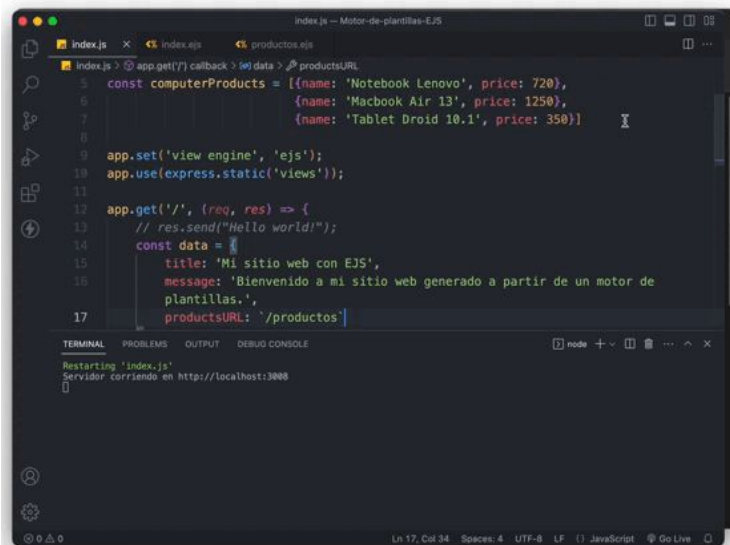
## Dinamizar contenido

Finalmente, agregamos la función **include()** en el archivo **index.ejs** y en cualquier otro archivo asociado, para referenciar en éstos los encabezados y pié para cada página.

```
productos.ejs

<%- include('head') %>
<div class="container">
  <h1><%= title %></h1>
  <p><%= message %></p>
  <ul>
    <% for (let i = 0; i < products.length; i++) { %>
      <li><%= products[i].name %> -
        ($ <%= products[i].price %>)</li>
    <% } %>
  </ul>
</div>
<%- include('footer') %>
</body>
</html>
```

# Dinamizar contenido

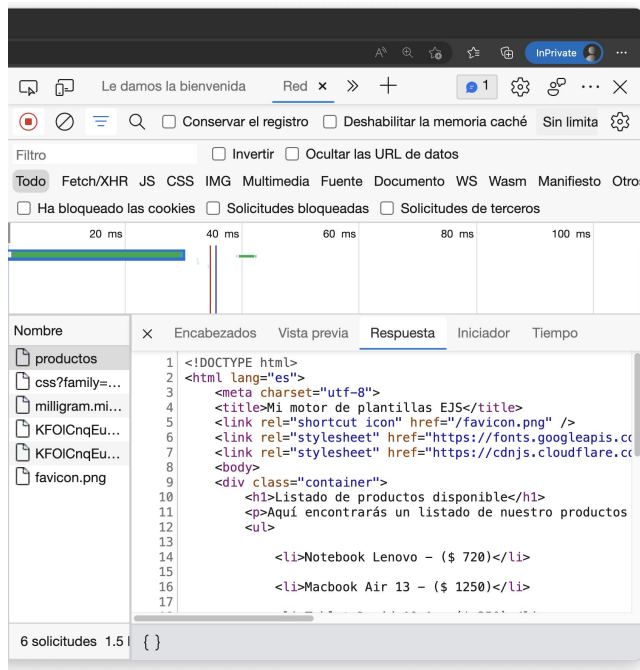


```
index.js > app.get('/', (req, res) => {  
  // res.send("Hello world!");  
  const data = {  
    title: 'Mi sitio web con EJS',  
    message: 'Bienvenido a mi sitio web generado a partir de un motor de  
    plantillas.',  
    productsURL: '/productos'  
  }  
  res.render('index', data);  
});  
app.set('view engine', 'ejs');  
app.use(express.static('views'));  
app.get('/', (req, res) => {  
  // res.send("Hello world!");  
  const data = {  
    title: 'Mi sitio web con EJS',  
    message: 'Bienvenido a mi sitio web generado a partir de un motor de  
    plantillas.',  
    productsURL: '/productos'  
  }  
  res.render('index', data);  
});  
app.set('view engine', 'ejs');  
app.use(express.static('views'));
```

Restarting 'index.js'  
Servidor corriendo en http://localhost:3000

Con ambas rutas definidas en nuestro proyecto Node.js, estamos listos para probar el servidor web renderizando contenido mediante el **Motor de plantillas EJS**.

# Dinamizar contenido

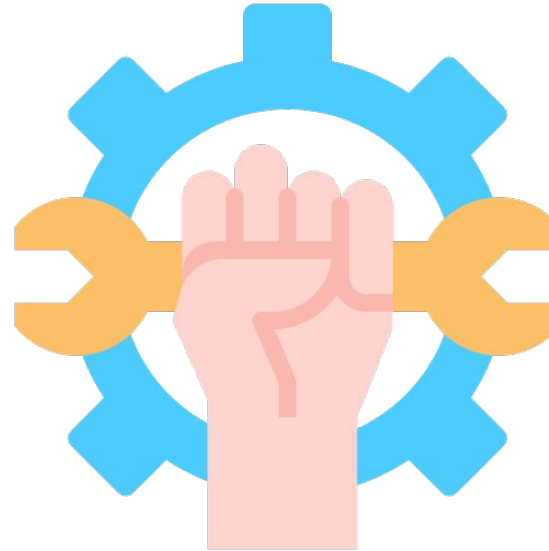


También podemos validar con DevTools que el documento descargado en el navegador web generó toda su estructura HTML, tal como si fuese un documento HTML único.

# Ventajas de uso de un Motor de plantillas

Como podemos apreciar, EJS es una herramienta poderosa para implementar en el desarrollo de aplicaciones backend, y, si debemos enumerar algunas de sus ventajas, destacamos:

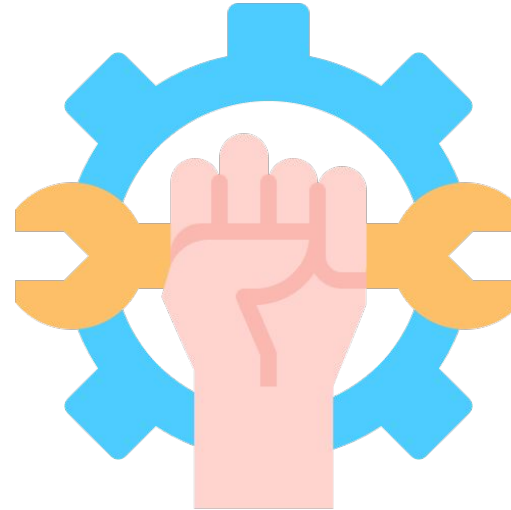
- Separación de responsabilidades
- Reutilización de código
- Flexibilidad
- Mayor velocidad
- Facilidad de mantenimiento



# Desventajas de uso de un Motor de plantillas

Aunque los motores de plantillas tienen muchas ventajas, también hay algunas desventajas a tener en cuenta:

- Curva de aprendizaje
- Sobrecarga
- Tamaño del paquete
- Seguridad



# Desafío

Crearemos una plantilla EJS creando una tabla HTML dinámica. En la misma, cargaremos un array de productos, creando una celda por cada una de las filas del array.

Utilizaremos una estructura EJS para ello. Tomaremos como base el proyecto de **webserver** construido durante el espacio de práctica de esta clase.



# Prácticas

La estructura del servidor web base debe ser similar a la siguiente:

- “/”
- “/productos”
- “/\*”

Crearemos dos plantillas **HTML/EJS**. Una de ellas será servida a través de la URL principal del sitio, la otra a través de la URL productos.

Debemos definir un encabezado **<head>** común para ambos documentos EJS, e insertar este **<head>** en ambos documentos utilizando la función **include()** de EJS.

El documento **productos.ejs** deberá contener una tabla HTML. En el apartado **<tbody>** debemos generar cada fila **<tr>** y cada celda **<td>** HTML, a partir de la información contenida en el array productos, utilizando para esto el ciclo for de EJS.

Agregaremos un control de errores para las rutas inexistentes creando una plantilla EJS dedicada. Podremos estilizarla de acuerdo a nuestro parecer, e integrar en el desarrollo de cada plantillas, algún framework CSS como Bootstrap, Milligram o cualquier otro.



# Muchas gracias.



Ministerio de Economía  
**Argentina**

Secretaría de  
Economía del Conocimiento

*primero  
la gente*