



Clase 10. Programación Backend

PUG & EJS



OBJETIVOS DE LA CLASE

- Entender qué es un motor de plantillas y su implementación en el backend.
- Conocer el motor de plantillas Handlebars: sintaxis y uso, e integrarlo a Express.
- Conocer el motor de plantillas Pug: sintaxis y uso, e integrarlo a Express.
- Conocer el motor de plantillas Ejs: sintaxis y uso, e integrarlo a Express.

CRONOGRAMA DEL CURSO

Clase 4



Motores de Plantillas

Clase 5



Pug & Ejs

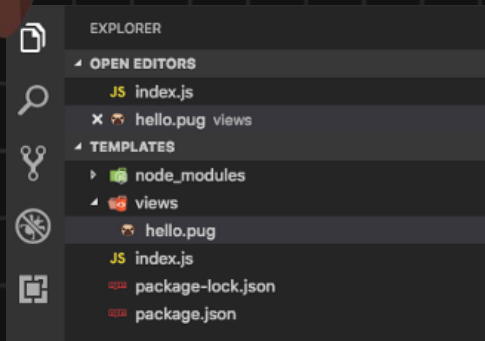
Clase 6



Websockets

Pug

```
hello.pug — templates
JS index.js  hello.pug x
1  html
2    head
3      title='Mi primera plantilla Pug JS'
4    body
5      h1=mensaje
```





¿Qué es Pug?

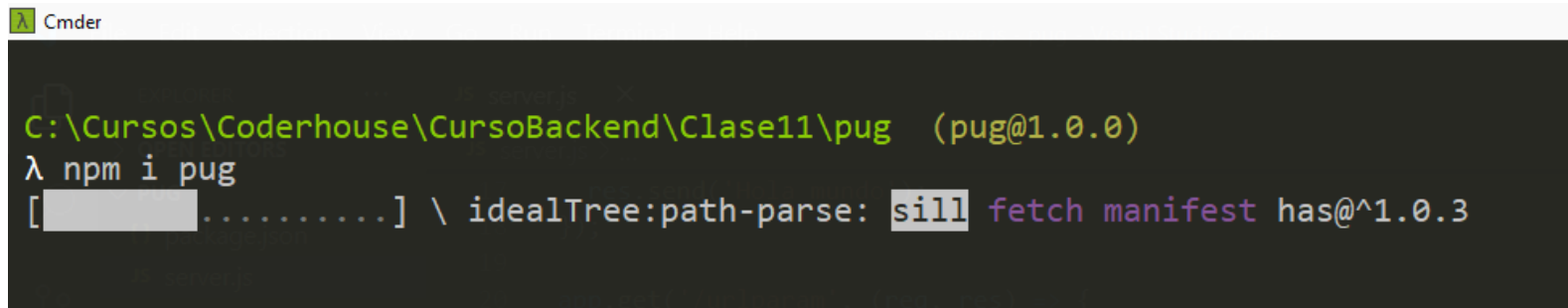


- Pug JS es un **motor de plantillas** que nos permite **utilizar archivos estáticos** como plantillas, enviar **valores** para **reemplazar variables** dentro de las mismas y **transformar** estos archivos en **páginas HTML** que se envían al cliente.
- Express permite trabajar con muchos motores de plantillas, entre los que se encuentra Pug JS.
- Pug es muy **fácil de implementar**, solo bastará un par de líneas de código para indicarle a *express* que use Pug JS como motor de plantillas.

Instalar Pug JS

Para esto abriremos la terminal, nos posicionamos en la ruta de nuestra aplicación y lo instalaremos con ayuda de npm con el siguiente comando

```
npm install pug
```



```
λ Cmder  
C:\Cursos\Coderhouse\CursoBackend\Clase11\pug (pug@1.0.0)  
λ npm i pug  
[REDACTED] .....] \ idealTree:path-parse: sill fetch manifest has@^1.0.3
```



Configuración

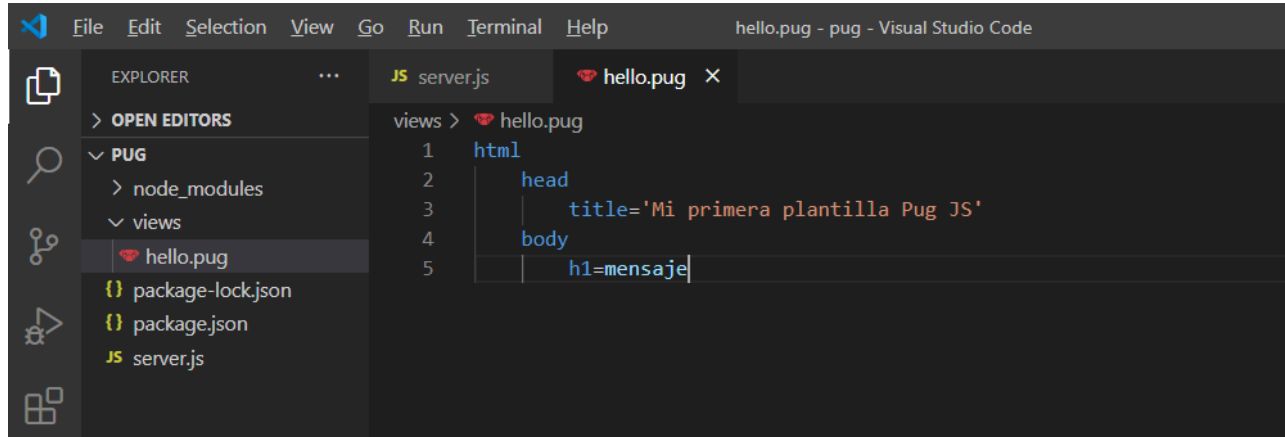


- El primer paso es crear un directorio en carpeta en la raíz de nuestro proyecto para guardar las plantillas que se utilizarán en la aplicación. En la imagen (siguiente slide) se puede apreciar el nuevo directorio creado “views” (*panel lateral izquierdo*).
- Ahora necesitamos indicarle a express que “views” será nuestro directorio de plantillas. Y también indicar cuál será el motor de plantillas que se utilizará (en este caso Pug JS). Lo configuramos con:

- `app.set('views', './views');`
- `app.set('view engine', 'pug');`

Crear nuestra primera plantilla

Una vez que se ha configurado e instalado correctamente Pug js solo queda **crear** nuestra **primera plantilla** y mostrarla al cliente. Para ello crearemos el archivo hello.pug (.pug es la extensión de las plantillas) y la mostraremos al ingresar en la url:
localhost:8080/hello

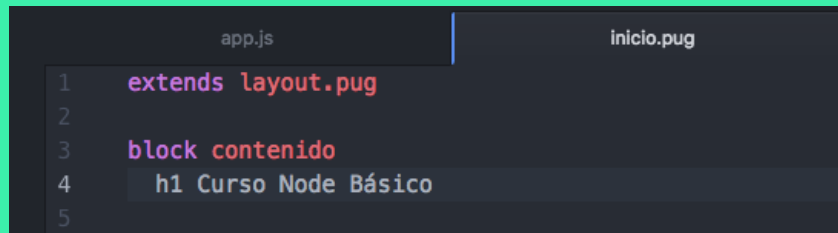


The screenshot shows the Visual Studio Code interface with the following details:

- Explorer Panel:** Displays the file structure with folders `PUG` and `views`. The file `hello.pug` is selected under the `views` folder. Other files listed are `package-lock.json`, `package.json`, and `server.js`.
- Editor Panel:** Shows the content of `hello.pug` with the following code:

```
1  html
2    head
3      title='Mi primera plantilla Pug JS'
4    body
5      h1=mensaje
```
- Terminal Panel:** Empty.
- Run and Debug Panel:** Empty.
- Output Panel:** Empty.

Pug: sintaxis



```
app.js | inicio.pug
1 extends layout.pug
2
3 block contenido
4   h1 Curso Node Básico
5
```

- Pug JS utiliza su **propia sintaxis** para declarar atributos html sin necesidad de abrir y cerrar etiquetas. En cambio se usa la **tabulación** para indicar que una etiqueta pertenece o está dentro de otra.

Para mayor información visitar el sitio web oficial de pugjs:

<https://pugjs.org/api/getting-started.html>

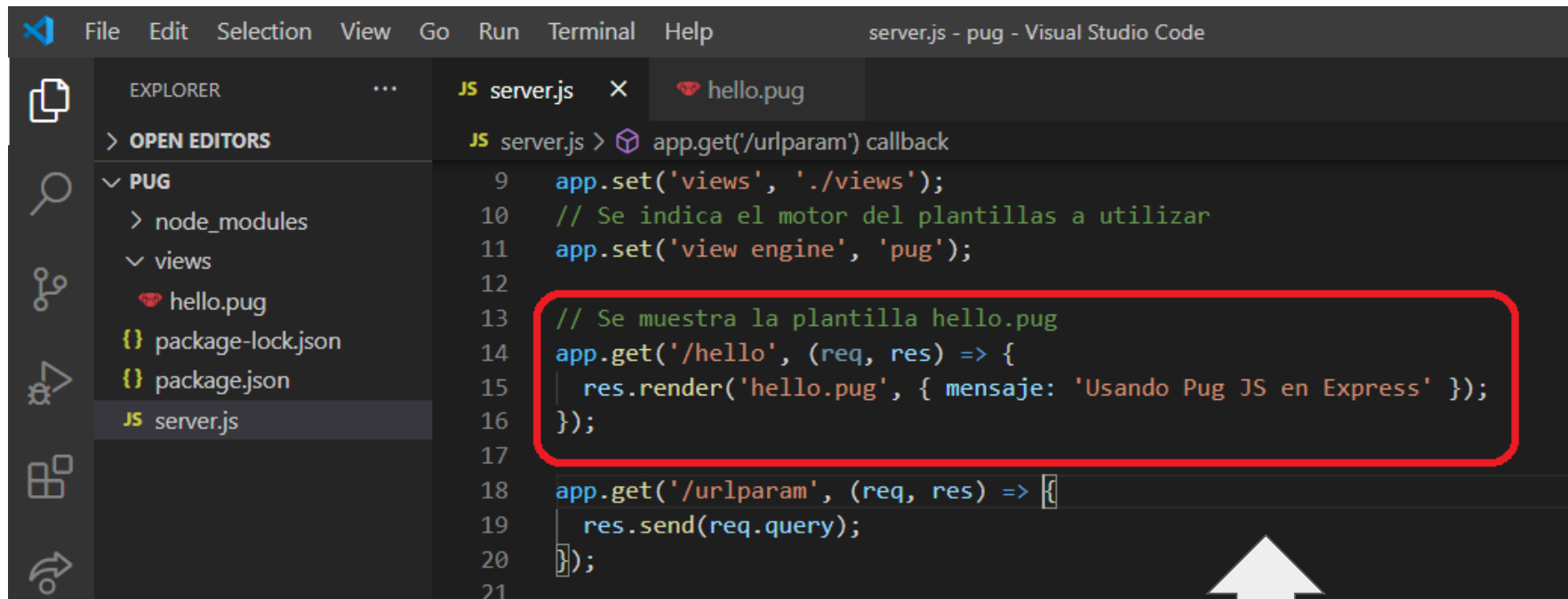
- En la línea 5 de **hello.pug** se está declarando una **variable** con nombre “**mensaje**”. Esta será **reemplazada** con el valor que se enviará al momento de transformar de formato .pug a HTML

Siguiente paso

- El siguiente paso será **modificar la ruta “/hello”** que actualmente está mostrando al usuario el mensaje “Hola mundo”. Lo reemplazamos **con la plantilla que creamos**.
- Para ello usaremos la función “**render**” que está disponible en el objeto res (response). Esta función recibe **dos parámetros**: el primero es el **nombre** de la plantilla a mostrar y el segundo un **objeto** con los valores a reemplazar.

```
res.render(view: string, options?: Object)
```

A continuación se puede ver el código final:-



```
server.js - pug - Visual Studio Code

EXPLORER
  > OPEN EDITORS
  PUG
    > node_modules
    > views
      hello.pug
      package-lock.json
      package.json
      JS server.js

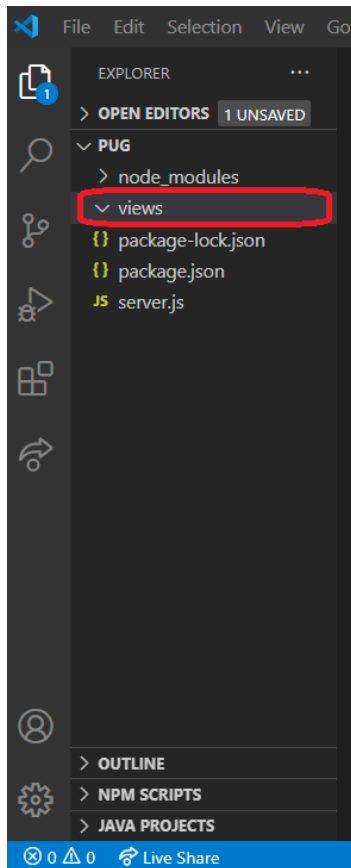
JS server.js
  app.get('/urlparam') callback
  9 app.set('views', './views');
  10 // Se indica el motor del plantillas a utilizar
  11 app.set('view engine', 'pug');
  12
  13 // Se muestra la plantilla hello.pug
  14 app.get('/hello', (req, res) => {
  15   res.render('hello.pug', { mensaje: 'Usando Pug JS en Express' });
  16 });
  17
  18 app.get('/urlparam', (req, res) => {
  19   res.send(req.query);
  20 });
  21
```

localhost:8080/hello

Usando Pug JS en Express

CODER HOUSE

Implementación de pug en express



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

app.use('/static', express.static(__dirname + '/public'));
app.use(express.urlencoded({ extended: false }));
app.use(express.json());

// Se indica el directorio donde se almacenarán las plantillas
app.set('views', './views');
// Se indica el motor de plantillas a utilizar
app.set('view engine', 'pug');

app.get('/hello', (req, res) => {
  res.send('Hola mundo');
});

app.get('/urlparam', (req, res) => {
  res.send(req.query);
});

app.post('/urljson', (req, res) => {
  res.send(req.body);
});

const PORT = 8080;
app.listen(PORT, () => console.log(`Servidor iniciado en el puerto ${PORT}`));
```





PUG

Vamos a practicar lo aprendido hasta ahora

Tiempo: 10 minutos

CODER HOUSE



- 1) Realizar un servidor que reciba por query params (mediante la ruta get '/datos') el valor que debe representar una barra de medición (usando el tag de html *meter*).
- 2) Asimismo recibirá además los valores mínimos y máximos permitidos y el título que se pondrá por arriba de la barra, en un elemento h1 en color azul (debe permitir formato HTML).

Ejemplo de petición:

`http://localhost:8080/datos?min=10&nivel=15&max=20&titulo=<i>Medidor</i>`

- 1) Como respuesta a este request, el servidor devolverá al frontend una plantilla armada con los datos recibidos.
- 2) Utilizar pug integrado a express, manejando una plantilla común y una particular con la representación requerida.



Mi plantilla de medidor en Pug JS x



localhost:8080/datos?min=5&nivel=15&max=20&titulo=<i>Medidor</i>

Medidor

5  20

EJS

<%= EJS %>

Effective JavaScript templating.



CODER HOUSE

<%= **EJS** %>

¿Qué es EJS?

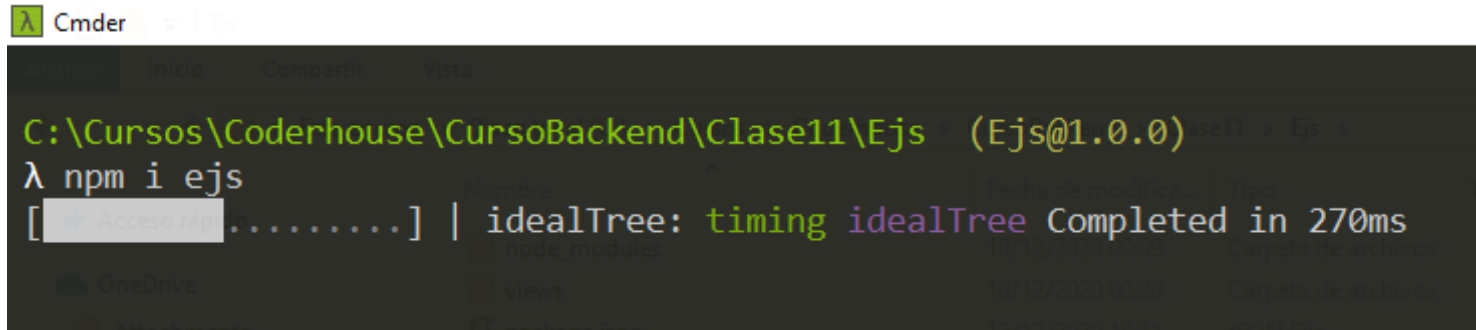


- EJS se encuentra entre los **motores de visualización temáticos más populares para node.js y express** con 5k estrellas en github y más de 8 millones de descargas por semana en npm.
- EJS significa **plantillas de JavaScript incrustadas** y podemos usarlo tanto en el **lado del servidor** como en el **del cliente**. En esta presentación, nos centraremos en el lado del servidor.
- EJS es **fácil de configurar** y podemos incluir las partes repetibles de nuestro sitio (parciales) y pasar los datos a nuestras vistas.

Instalar EJS

Para esto abriremos la terminal, nos posicionamos en la ruta de nuestra aplicación y lo instalaremos con ayuda de npm con el siguiente comando

```
npm install ejs
```



A screenshot of a Windows Command Prompt window. The title bar shows 'Cmder' with a green icon. The window has a dark background. The command prompt shows the current directory as 'C:\Cursos\Coderhouse\CursoBackend\Clase11\Ejs' and the version of EJS as '(Ejs@1.0.0)'. The command 'npm i ejs' has been entered. The output shows a progress bar for the installation, followed by the text '| idealTree: timing idealTree Completed in 270ms'. The window also shows standard Windows taskbar elements like 'Inicio', 'Compartir', 'Vista', and 'OneDrive'.

```
C:\Cursos\Coderhouse\CursoBackend\Clase11\Ejs (Ejs@1.0.0)
λ npm i ejs
[Accessing .....] | idealTree: timing idealTree Completed in 270ms
```



Configuración



- Configuramos EJS como el motor de visualización de nuestra aplicación Express usando `app.set('view engine', 'ejs')`
- Creamos una carpeta de vistas: **views**
- EJS enviará una vista al usuario usando *res.render()*. Es importante tener en cuenta que *res.render()* buscará la vista en una carpeta **views**.
- Por ejemplo, si definimos *pages/index* dentro de views, ***res.render('pages/index')*** buscará en *views/pages/index*.

Implementación de EJS en express

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

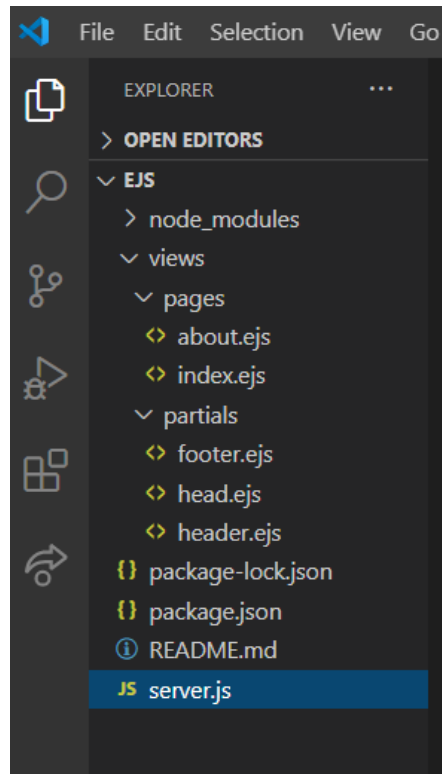
// configuramos EJS como motor de plantillas
app.set('view engine', 'ejs');

// index page
app.get('/', function(req, res) {
  var mascots = [
    { name: 'Sammy', organization: "DigitalOcean", birth_year: 2012},
    { name: 'Tux', organization: "Linux", birth_year: 1996},
    { name: 'Moby Dock', organization: "Docker", birth_year: 2013}
  ];
  var tagline = "No programming concept is complete without a cute animal mascot.";

  res.render('pages/index', {
    mascots: mascots,
    tagline: tagline
  });
});

// about page
app.get('/about', function(req, res) {
  res.render('pages/about');
});

app.listen(8080);
console.log('8080 is the magic port');
```





Sintaxis básica (etiquetas)



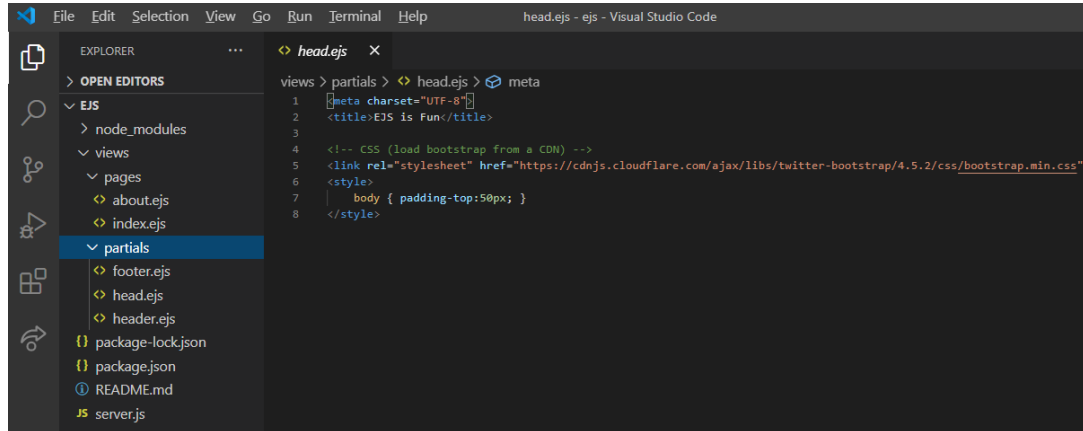
- `<%=` Incrusta en la plantilla el valor tal cual está
- `<%-` Incrusta en la plantilla el valor renderizado como HTML
- `<%` 'Scriptlet': Admite instrucciones en JS para declaración de variables y control de flujo

Ejemplo

```
<% const estilo = "color:crimson;"; %>
<% if (message) { %>
  <h2 style=<%=estilo%>><%= message.name %></h2>
<% } %>
```

Creando nuestras plantillas Parciales

- Al igual que muchas aplicaciones que creamos, hay mucho **código que se reutiliza**. En EJS llamamos a estos códigos ***parciales***
- En el ejemplo que mostramos a continuación, los definimos dentro de la **carpeta 'partials'**



```
File Edit Selection View Go Run Terminal Help head.ejs - ejs - Visual Studio Code

EXPLORER
  > OPEN EDITORS
  > EJS
    > node_modules
    > views
      > pages
        > about.ejs
        > index.ejs
        > partials
          > footer.ejs
          > head.ejs
          > header.ejs
        package-lock.json
        package.json
        README.md
        JS server.js

views > partials > head.ejs > meta
1 [meta charset="UTF-8"]
2 <title>EJS is Fun</title>
3
4 <!-- CSS (load bootstrap from a CDN) -->
5 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/twitter-bootstrap/4.5.2/css/bootstrap.min.css">
6 <style>
7   body { padding-top: 50px; }
8 </style>
```

Ejemplo: Plantillas Parciales

views/partials/head.ejs

```
<meta charset="UTF-8">
<title>EJS Is Fun</title>

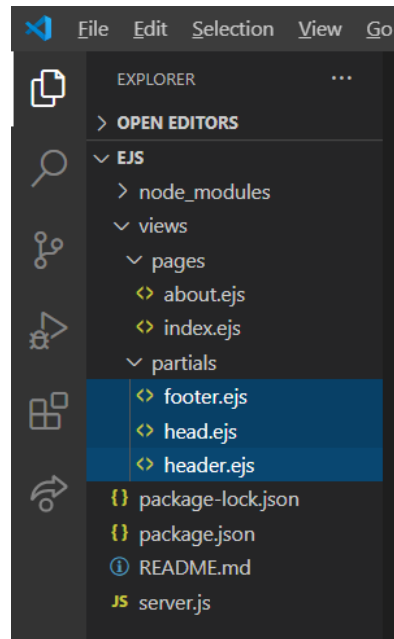
<!-- CSS (load bootstrap from a CDN) -->
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.2/css/bootstrap.min.css">
<style>
  body { padding-top:50px; }
</style>
```

views/partials/header.ejs

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="/">EJS Is Fun</a>
  <ul class="navbar-nav mr-auto">
    <li class="nav-item">
      <a class="nav-link" href="/">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/about">About</a>
    </li>
  </ul>
</nav>
```

views/partials/footer.ejs

```
<p class="text-center text-muted">© Copyright 2020 The Awesome People</p>
```



Añadiendo los parciales de EJS a Vistas

Ahora tenemos nuestros parciales definidos. Lo único que debemos hacer es **incluirlos en nuestras vistas**.

- Utilizamos `<%- include('RELATIVE/PATH/TO/FILE') %>` para integrar un parcial de EJS en otro archivo.
- El guión `<%-` en lugar de solo `<%` es para indicar a EJS que **renderice HTML sin formato**.
- La ruta al parcial es **relativa** al archivo actual

Ejemplo: Plantillas Parciales en Vista

views/pages/index.ejs

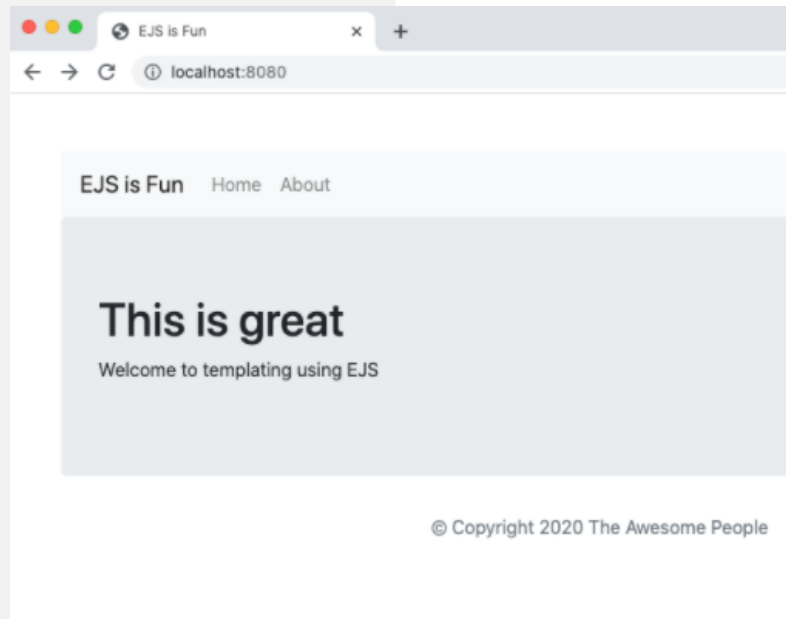
```
<!DOCTYPE html>
<html lang="en">
<head>
  <%- include('../partials/head'); %>
</head>
<body class="container">

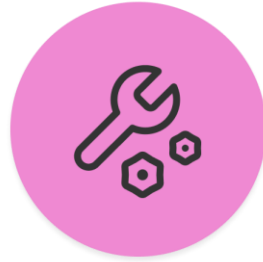
<header>
  <%- include('../partials/header'); %>
</header>

<main>
  <div class="jumbotron">
    <h1>This is great</h1>
    <p>Welcome to templating using EJS</p>
  </div>
</main>

<footer>
  <%- include('../partials/footer'); %>
</footer>

</body>
</html>
```





EJS

Realizar el mismo ejercicio que en el desafío anterior, utilizando
ejs.

Tiempo: 10 minutos

CODER HOUSE



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

EJS: Incorporando Datos en Vistas

Pasando datos a Vistas

Vamos a definir una variable y una lista para pasar a nuestra página de inicio. Volvamos al archivo `server.js` e incorporemos lo siguiente dentro de la ruta **`app.get('/')`**

server.js

```
// index page
app.get('/', function(req, res) {
  var mascots = [
    { name: 'Sammy', organization: "DigitalOcean", birth_year: 2012},
    { name: 'Tux', organization: "Linux", birth_year: 1996},
    { name: 'Moby Dock', organization: "Docker", birth_year: 2013}
  ];
  var tagline = "No programming concept is complete without a cute animal mascot.";

  res.render('pages/index', {
    mascots: mascots,
    tagline: tagline
  });
});
```

Ejemplo: datos en Vistas

Renderizar una variable única en EJS

- *Para utilizar una de las variables pasada, usamos directamente el nombre de la misma. En este caso: **<%= tagline %>***

views/pages/index.ejs

```
...  
<h2>Variable</h2>  
<p><%= tagline %></p>  
...
```

Ejemplo: datos en Vistas

Iterar sobre datos contenidos en una variable

- *Aquí utilizamos código JS. Por ejemplo, podemos usar `.forEach`:*

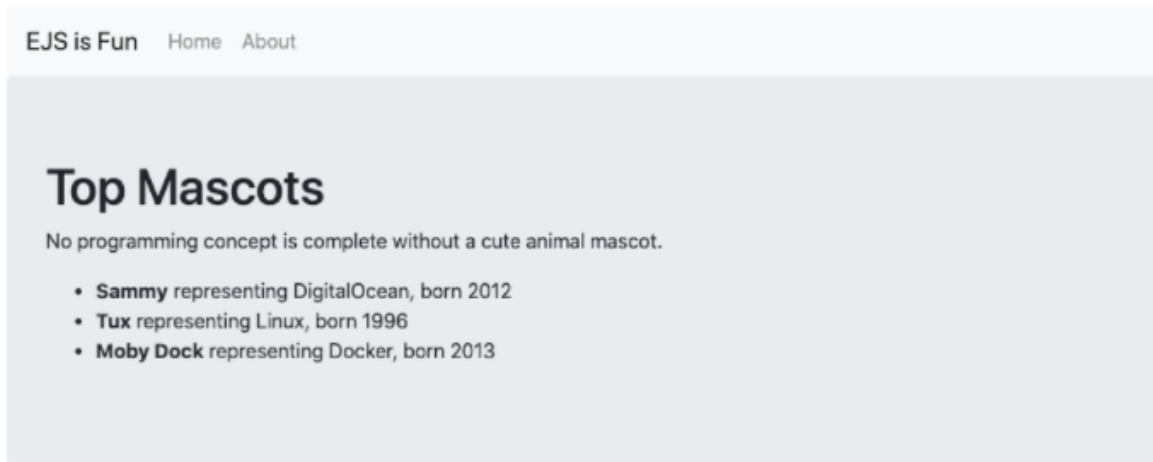
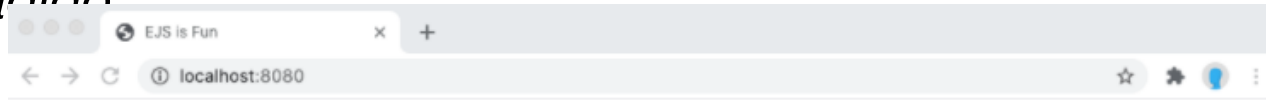
```
views/pages/index.ejs

...
<ul>
  <% mascots.forEach(function(mascot) { %>
    <li>
      <strong><%= mascot.name %></strong>
      representing <%= mascot.organization %>, born <%= mascot.birth_year %>
    </li>
  <% }); %>
</ul>
...
```

- *Alternativamente, se puede utilizar también: `for ... of`*

Ejemplo: datos en Vistas

Podemos ver en nuestro navegador la información que hemos añadido



EJS: Incorporando Datos en Parciales

Pasando datos a un parcial en EJS

El parcial EJS tiene acceso a todos los datos que la vista principal.

Aclaración: si hacemos referencia a una variable en un parcial, debe definirse en cada vista que utilice el parcial o arrojará un error.

- También podemos definir y pasar variables a un parcial EJS en la **sintaxis include**, pasándolos como segundo argumento:

views/pages/about.ejs

```
...  
<header>  
  <%- include('../partials/header', {variant:'compact'}); %>  
</header>  
...
```

Pasando datos a un parcial en EJS

Si deseamos referenciar una variable en un parcial que puede no definirse siempre, y darle un valor predeterminado, podemos hacerlo de esta forma:

views/partials/header.ejs

```
...  
<em>Variant: <%= typeof variant != 'undefined' ? variant : 'default' %></em>  
...
```

*En la línea anterior, el código EJS se renderiza el valor de **variant** si está definido y de **default** si no lo está.*



FORMULARIO + HISTORIAL

Tiempo: 15 minutos



- 1) Desarrollar un servidor basado en node.js, express y ejs que disponga de un **formulario** en su ruta raíz (creado con una plantilla de ejs) para ingresar los siguientes datos de una persona: nombre, apellido y edad.
- 2) La información será enviada mediante el **método post** al endpoint '/personas
- 3) Representar por debajo del mismo formulario los **datos históricos** ingresados más el actual en forma de tabla. En el caso de no encontrarse información mostrar el mensaje '**No se encontraron datos**' en lugar de la tabla.

Se sugiere el uso de bootstrap para los estilos de las plantillas. Ejemplos a continuación:



Mi plantilla de Ingreso de Datos x +

localhost:8080

Ingrese datos

Nombre

Apellido

Edad

Enviar

Historial

Nombre	Apellido	Edad
Daniel	Sánchez	52
Ana	Mei	23
Diego	Suarez	34

Mi plantilla de Ingreso de Datos x +

localhost:8080

Ingrese datos

Nombre

Apellido

Edad

Enviar

Historial

no se encontraron datos



Motores de Plantillas

Incorporando Handlebars

Formato: link a un repositorio en Github con los tres proyectos en carpetas separadas. No incluir los node_modules.

Desafío
entregable



>> Consigna:

- 1) Utilizando la misma API de productos del proyecto entregable de la clase anterior, construir un web server (no REST) que incorpore:
 - a) Un formulario de carga de productos en la ruta raíz (configurar la ruta '/productos' para recibir el POST, y redirigir al mismo formulario).
 - b) Una vista de los productos cargados (utilizando plantillas de handlebars) en la ruta GET '/productos'.
 - c) Ambas páginas contarán con un botón que redirija a la otra.

Incorporando Handlebars

Formato: link a un repositorio en Github con los tres proyectos en carpetas separadas. No incluir los node_modules.

Desafío
entregable



>> Consigna:

- 2) Manteniendo la misma funcionalidad reemplazar el motor de plantillas handlebars por **pug**.
- 3) Manteniendo la misma funcionalidad reemplazar el motor de plantillas handlebars por **ejs**.
- 4) Por escrito, indicar cuál de los tres motores de plantillas prefieres para tu proyecto y por qué.

Incorporando Handlebars

Formato: link a un repositorio en Github con los tres proyectos en carpetas separadas. No incluir los node_modules.

Desafío
entregable



>> Aspectos a incluir en el entregable:

- Realizar las plantillas correspondientes que permitan recorrer el array de productos y representarlo en forma de tabla dinámica, siendo sus cabeceras el nombre de producto, el precio y su foto (la foto se mostrará como un imagen en la tabla)
- En el caso de no encontrarse datos, mostrar el mensaje: 'No hay productos'.

>> Sugerencias:

- Utilizar iconfinder (https://www.iconfinder.com/free_icons) para obtener la url de las imágenes de los productos (click derecho sobre la imagen -> copiar dirección de la imagen)

Incorporando Handlebars

Formato: link a un repositorio en Github con los tres proyectos en carpetas separadas. No incluir los node_modules.

Desafío
entregable



>> Opcional:

- Utilizar bootstrap para maquetar la vista creada por dicho motor de plantillas y el formulario de ingreso de productos.

Ejemplos a continuación





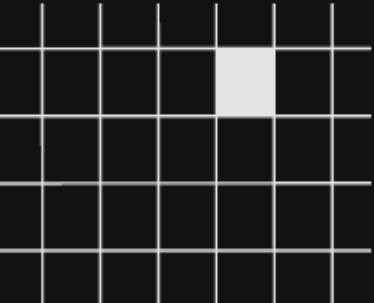
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Motor de plantillas Pug
 - Motor de plantillas EJS
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN

CODER HOUSE