Introducción a Express (Parte II)

Integrando jQuery y Bootstrap

Competencia

- Construir un servidor utilizando Express para servir un sitio web estático con Bootstrap y jQuery integrados.

Introducción

Ya has aprendido a devolver sitios web de contenido estático, pero ¿Qué pasa si necesitamos utilizar librerías o frameworks, en las aplicaciones frontend que devolvemos en la ruta raíz? Podríamos ocupar sus cdn's, sin embargo, esto generaría una dependencia remota en nuestros proyectos, y si necesitamos que nuestra aplicación funcione de forma offline no sería alternativa viable.

En el desarrollo frontend aprendiste que también se pueden descargar los códigos fuentes e importarlos directamente en los documentos HTML, no obstante, ahora nos encontramos en un entorno Node y todas las rutas deben estar programadas en el servidor, de lo contrario no se reconocerían y devolvería un mensaje genérico.

Entonces ¿Cómo resolvemos este dilema? En este capítulo aprenderás a instalar y utilizar jQuery y Bootstrap de forma local a través de NPM y los middlewares de Express para definir rutas personalizadas y disponibilizar los códigos fuentes que se encuentran dentro de la carpeta node_modules, de esa forma importarlos en el index.html que devolvemos en la ruta raíz.

Importación de jQuery y Bootstrap

La librería jQuery y el framework de componentes CSS Bootstrap han de ser 2 de las tecnologías más comunes de aprender al iniciar en el desarrollo web, a pesar de tener muchos años desde su creación, siguen siendo opciones importantes a considerar en el desarrollo de sitios web modernos.

En el desarrollo Node disponemos de NPM por defecto como gestor de paquete, con el podemos instalar dependencias y paquetes a nuestro proyecto para hacerlo más escalable. En el desarrollo a nivel de cliente, la importación de jQuery y Bootstrap se maneja por cdn o descargando su código fuente. Para el caso del backend, tenemos a disposición los siguientes comandos:

jQuery:

```
npm install --save jquery
```

Bootstrap:

```
npm install --save bootstrap
```

¿Cuál es la diferencia de instalarlos de esta manera y no importarlos por cdn? La diferencia está en la estabilidad y la independencia de este código fuente, ya que al tenerlo instalado en nuestro proyecto, podremos usar sus métodos y clases de forma offline, evitamos la necesidad de conectarnos remotamente al cdn de estas tecnologías.

Entonces ¿Cuál es la diferencia de instalarlos como paquetes NPM, descargar el código y alojarlos en un directorio público? La diferencia está en la ubicación de los archivos fuentes, ya que al utilizar NPM las dependencias se instalan en la carpeta node_modules.

Ejercicio guiado: Importando Bootstrap y jQuery desde node_modules

Servir un sitio web estático que importe jQuery y Bootstrap, ambos instalados con NPM. El objetivo de este ejercicio será aprender a usar los middlewares y el método “static” para definir 2 rutas que permita importar los códigos fuente de nuestras dependencias desde la carpeta node_modules.

Consideraciones previas

Recordemos que ambas tecnologías deben ser importadas desde un Index.html con las etiquetas correspondientes, sus rutas (atributos href y src) y deben apuntar a sus códigos fuentes.

También, debemos preguntarnos ¿Cómo importar un archivo ubicado dentro de los node_modules a mi index.html? Esto no será necesario ni posible, recordemos que los servidores con Express, disponibilizan contenido a través de rutas o directorios "liberados" con el método "static" de la instancia de Express.

Lo anterior, nos plantea una última pregunta ¿Cómo haré para importar jQuery y Bootstrap desde mi sitio web HTML devuelto por Express, teniendo sus códigos fuentes dentro de una carpeta node_modules? La respuesta y la solución la encontramos con los "middlewares", puesto que podremos especificar que una ruta "X" será la referencia para acceder a un archivo o carpeta dentro del profundo árbol de archivos de node_modules.

Hablando de los node_modules, debemos estar conscientes de la ubicación de estos archivos fuentes en esta carpeta, para eso despliega los node_modules y busca las carpetas correspondientes a jQuery y Bootstrap, donde encontrarás que ambos tienen una subcarpeta llamada "dist" y dentro de esta, se encuentran los tan queridos archivos fuentes.

En la siguiente imagen te muestro la navegación dentro de los node_modules para acceder a bootstrap.css:

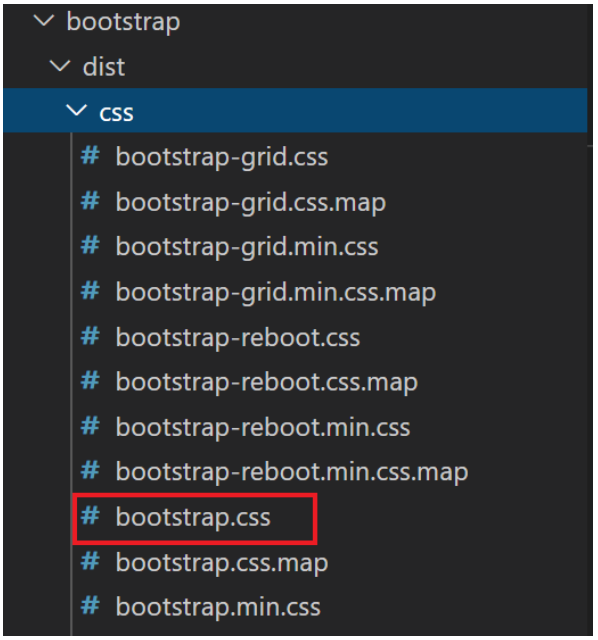

Imagen 1. Navegación en la carpeta node_modules para ubicar el archivo bootstrap.css.
Fuente: Desafío Latam

Ahora que sabemos dónde está nuestro framework de CSS, en la siguiente imagen te muestro la ubicación del jquery.js:

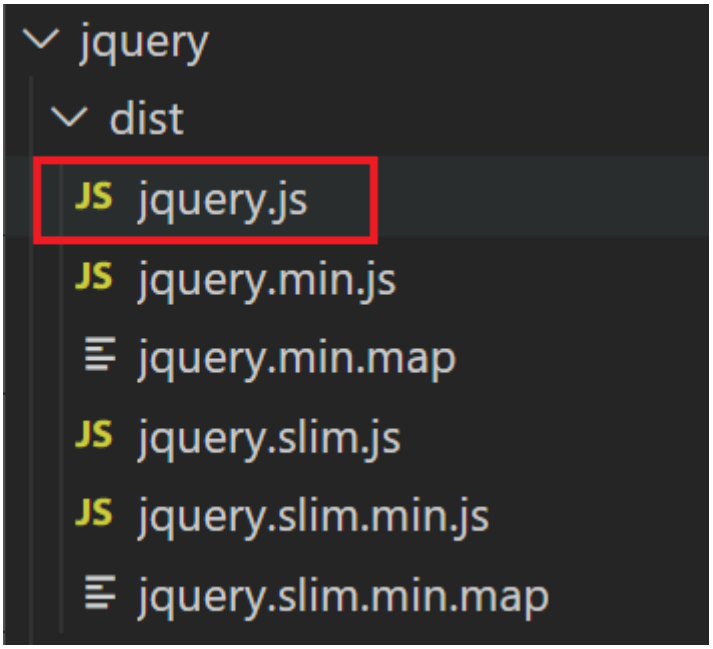

Imagen 2. Navegación en la carpeta node_modules para ubicar el archivo jquery.js.
Fuente: Desafío Latam

Teniendo clara de la ubicación de ambos archivos, sigue los siguientes pasos para descubrir cómo resolver este ejercicio:

- ***Paso 1:*** Crear un servidor con Express que escuche el puerto 3000.
- ***Paso 2:*** Crear un middleware que define una ruta **/bootstrap** y libere el contenido de la carpeta "css", de la dependencia de Bootstrap en el node_modules.
- ***Paso 3:*** Crear un middleware que define una ruta **/jquery** y libere el contenido de la carpeta "dist", de la dependencia de jQuery en el node_modules.
- ***Paso 4:*** Crear una ruta **GET /** que devuelva un archivo index.html.

```
// Paso 1
const express = require("express");
const app = express();
app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

// Paso 2
app.use('/bootstrap', express.static(__dirname +
'/node_modules/bootstrap/dist/css'))

// Paso 3
app.use('/jquery', express.static(__dirname +
'/node_modules/jquery/dist'))

// Paso 4
app.get("/", (req, res) => {
  res.sendFile(__dirname + '/index.html')
})
```

Con la lógica escrita en el servidor, solo falta probar la efectividad de nuestros middlewares, por ende necesitaremos crear nuestro index.html y escribir dentro de este algún componente de Bootstrap, junto con la instrucción con jQuery. Sigue los siguientes pasos para preparar nuestra aplicación cliente y observa en el navegador que estamos sirviendo, desde nuestro servidor con Express un sitio web estático con jQuery y Bootstrap integrados con NPM.

- ***Paso 1:*** Importar Bootstrap.css con una etiqueta “link”.
- ***Paso 2:*** Importar un Jumbotron para demostrar que estamos importando correctamente Bootstrap. Te dejo el link oficial para que tomes este componente
- ***Paso 3:*** Importar jQuery.js usando la etiqueta “script”.
- ***Paso 4:*** Ocupa el document ready de jQuery para cambiarle el color de fondo al Jumbotron de Bootstrap a negro y cambiarle el color de texto a blanco.

```
<!-- Paso 1 -->
<link rel="stylesheet" href="/bootstrap/bootstrap.css" />
<!-- Paso 2 -->
<div class="jumbotron">
  <h1 class="display-4">Hello, world!</h1>
  <p class="lead">
    This is a simple hero unit, a simple jumbotron-style component for
calling
    extra attention to featured content or information.
  </p>
  <hr class="my-4" />
  <p>
    It uses utility classes for typography and spacing to space content
out
    within the larger container.
  </p>
  <p class="lead">
    <a class="btn btn-primary btn-lg" href="#" role="button">Learn
more</a>
  </p>
</div>

<!-- Paso 3 -->
<script src="/jquery/jquery.js"></script>
<!-- Paso 4 -->
<script>
  $(document).ready(() => {
    $(".jumbotron").css("background", "black").css("color", "white");
  });
</script>
```

Visita la ruta raíz de tu servidor (http://localhost:3000/) con tu navegador preferido y deberás recibir lo que te muestro en la siguiente imagen:

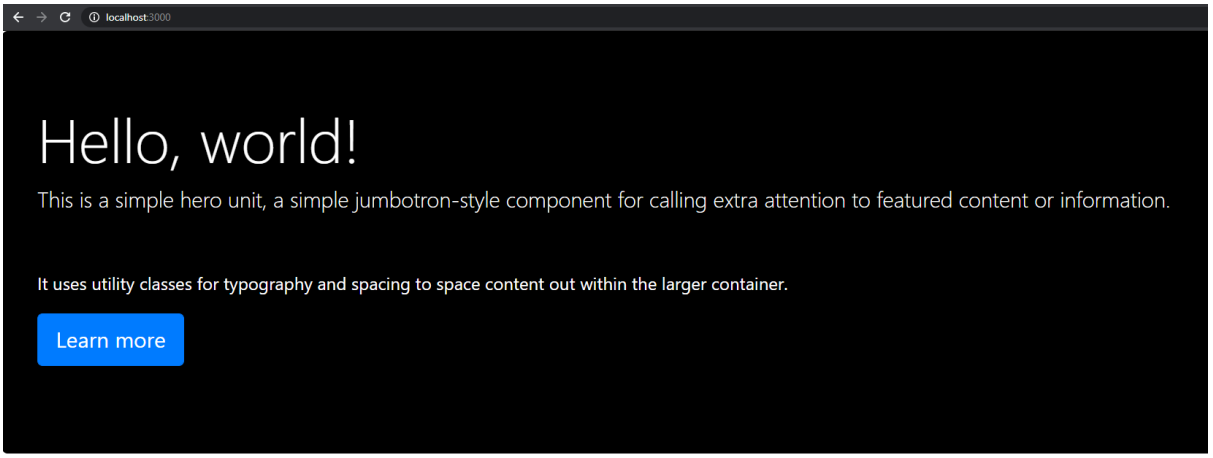

Imagen 3. Sitio web estático servido desde Express con jQuery y Bootstrap integrados.
Fuente: Desafío Latam

¡Maravilloso! Ahí está, lo logramos, ahora sabemos cómo importar jQuery y Bootstrap a los sitios web estáticos que servimos desde nuestro servidor con Express.

Ejercicio propuesto (1)

Desarrollar un servidor con Express que sirva en su ruta raíz un sitio web estático que muestre un navbar responsivo y un footer de Bootstrap. El componente navbar debe ser funcional, por lo que también se debe importar el documento JavaScript correspondiente a Bootstrap.

Handlebars y los motores de plantilla

Competencias

- Reconocer el uso de los motores de plantillas en el desarrollo de sitios web con contenido dinámico.
- Desarrollar un servidor con Express que renderice una plantilla usando Handlebars.

Introducción

Ya hemos visto cómo podemos gestionar las vistas, renderizando un archivo HTML en el cliente desde el backend con Express y su router, sin embargo, aún quedan muchas herramientas y funciones que aprovechar de nuestro framework. Lo próximo por aprender será un poco más avanzado, es momento de empezar a trabajar con contenido dinámico ¿Cómo haremos esto? Utilizando motores de plantillas para gestionar contenido y poder ocupar lógica JavaScript dentro de los templates.

En este capítulo, aprenderás qué son los motores de plantillas y cómo ocupar Handlebars para renderizar contenido de forma dinámica, logrando con esto un sistema más completo e interactivo para los usuarios de una aplicaciones web.

Motores de plantillas

¿Alguna vez te has visto en la necesidad de copiar y pegar código HTML? Muy probablemente sí, en especial cuando tenemos varios documentos HTML que necesitamos que compartan, por ejemplo, un menú de navegación y un footer.

En el desarrollo con Node tenemos la posibilidad de utilizar los motores de plantillas, los cuales, sirven como una herramienta que podemos integrar a nuestros servidores con Express ¿Pero qué hacen los motores de plantillas? Ofrecen renderizar y reutilizar código HTML como partes de un rompecabezas, simulando el principio de las funciones en la programación, lo cual, nos permite contar con una estructura que podemos volver a ocupar haciendo uso de su llamado. El objetivo principal de los motores de plantillas, es lograr una mejor organización del template en nuestros proyectos, en especial cuando tenemos varias vistas que comparten contenido.

Recordemos que con un código HTML puro no es posible hacer un mix con lógica y variables de JavaScript, en el desarrollo frontend aprendiste a utilizar el objeto “document” para hacer impresiones de datos en un elemento del DOM, e incluso recurrimos a jQuery para facilitar esto. Ahora trabajando en el backend, podemos optimizar este proceso ocupando los motores de plantillas y sus helpers para la inserción de lógica JavaScript en nuestros templates, logrando con esto la renderización de variables y/o condicionales dentro de nuestro código HTML, esto con el fin de facilitar el desarrollo de aplicaciones web para el lado del cliente, convirtiendo el contenido en dinámico.

Existen varios motores disponibles en NPM, siendo los más conocidos Pug, Ejs, Handlebars, ECT, Hogan y Chibi. En esta lectura aprenderemos a utilizar Handlebars.

Handlebars js

Es un motor de plantillas destacado por su velocidad y facilidad de uso. Permite la creación y reutilización de parciales, los cuales no son más que porciones o pedazos de HTML que pueden ser importados y reutilizados, además, cuenta con Helpers que nos ayudarán a renderizar variables y condicionales dentro de nuestro código HTML. Más adelante en la lectura, veremos en detalle el uso de los parciales y helpers, por ahora iniciaremos con la instalación e integración de este motor de plantillas a nuestro servidor desarrollado con Express.

Instalación

En NPM contamos con un paquete que ya viene configurado para ser integrado con Express, incluso su nombre es **express-handlers** y podemos instalarlo con el siguiente comando:

```
npm install --save express-handlebars
```

Integración y configuración

Ahora que tenemos nuestro motor de plantillas instalado, podemos proceder con su integración y configuración en donde especificaremos los directorios donde estarán ubicados nuestros archivos de extensión “handlebars”. Así es, no estaremos creando documentos puros HTML, sino que usaremos archivos cuya extensión se llaman igual que el motor de plantillas, no obstante, debes saber que esto pudiéramos cambiarlo a nuestro gusto y lo revisaremos a continuación.

Antes de iniciar con el desarrollo de nuestro primer ejercicio guiado con motor de plantilla, sigue los siguientes pasos:

1. Crea una carpeta nueva y abrela con tu editor preferido.
2. Instala Express y el express-handlebars.
3. Crea una carpeta llamada “views”. Esta carpeta debe llamarse de esta manera porque por defecto handlebars la buscará con ese nombre.
4. Dentro de la carpeta “views”, crea un archivo llamado **main.handlebars**.

Con lo anterior listo, ha llegado el momento de crear un servidor con Express e integrar handlebars.

Ejercicio guiado: Estrenando handlebars

Desarrollar un servidor con Express que utilice handlebars para renderizar una vista con una cabecera HTML que diga “Hola mundo! Usando ahora motores de plantillas =D”. Sigue los siguientes pasos para resolver este ejercicio de forma progresiva:

- ***Paso 1:*** Crear un servidor con Express e incluir el paquete express-handlebars en tus importaciones iniciales.

```
const express = require("express");
const app = express();
const exphbs = require("express-handlebars");

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});
```

- ***Paso 2:*** Ocupar un método de la instancia “app” llamado “set”. Este método lo que hace es definir una configuración para ser utilizada por Express, en nuestro caso la usaremos para integrar handlebars como motor de plantillas de la siguiente manera:

```
app.set("view engine", "handlebars");
```

- ***Paso 3:*** Configurar el motor de plantilla, para esto debemos usar el método “engine”, el cual define el motor de plantillas que utilizaremos en nuestro servidor con Express. Este método recibe los siguientes parámetros:

    - **Primer parámetro:** Define la extensión de los archivos que handlebars identificará como “vistas” o layouts para nuestra aplicación, en nuestro caso será el mismo nombre del motor, es decir: “handlebars”.
    - **Segundo parámetro:** Recibe la instancia de express-handlebars que importamos al inicio del código. Esta instancia es realmente un método que recibe un objeto de configuración en el que definiremos la propiedad “layoutsDir”, y cuyo valor será la ruta del directorio donde tendremos las vistas o layouts que utilizaremos en nuestra aplicación.

```
app.engine(
  "handlebars",
  exphbs({
    layoutsDir: __dirname + "/views",
  })
);
```

Con este código estamos diciendo que las vistas usadas en nuestra aplicación estarán almacenadas en la carpeta “views”.

- ***Paso 4:*** Crear la ruta raíz del servidor, como respuesta usaremos el método "render" del objeto response, el cual recibe los siguientes parámetros:

    - **Primer parámetro:** Define el nombre de la vista que queramos renderizar, en nuestro caso es "main".
    - **Segundo parámetro:** Recibe un objeto con el que podemos especificar el layout que queremos que se renderice, no obstante, por defecto handlebars buscará un archivo llamado "main.handlebars" por lo que no será necesario incorporarlo. Además, no es necesario colocar la extensión pues automáticamente es reconocida gracias a la configuración previa.

```
app.get("/", (req, res) => {
  res.render("main");
});
```

Es una configuración un poco enredada, debes saber que no todas las tecnologías que aprendemos se configuran en solo 1 paso, habrán frameworks o paquetes que requieran de este tipo de configuraciones para su correcto funcionamiento, lo importante es recordar que con la práctica es que lograremos digerir e interiorizar completamente los conocimientos, sólo a través del ensayo y error es que conseguimos el dominio de una herramienta.

Bien, tenemos escrito todo lo necesario a nivel de servidor, pero ¿Qué sucede con el archivo de extensión handlebars? Es momento de darle contenido. Utiliza el siguiente código HTML para este archivo:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Document</title>
</head>
<body>
   <h1>Hola mundo! Usando ahora motores de plantillas =D</h1>
</body>
</html>
```

Es momento de probar todo lo que hicimos, así que levanta tu servidor y consulta la dirección http://localhost:3000/, deberás recibir lo que te muestro en la siguiente imagen:

Imagen 4. Renderizando mi primera plantilla con handlebars.
Fuente: Desafío Latam

¡Excelente! con esto has logrado renderizar tu primera plantilla con handlebars desde un servidor con Express.

Ejercicio propuesto (2)

Desarrollar un servidor con Express que renderice una plantilla con un formulario HTML a través del motor de plantillas handlebars.

Parciales

Competencia

- Construir un servidor con Express que utilice páginas web parciales para el desarrollo de componentes reutilizables.

Introducción

En el capítulo anterior aprendimos cómo renderizar plantillas con Handlebars, una vez configurado e integrado dentro de un servidor con Express. En este capítulo, aprenderás a utilizar los parciales de handlebars para empezar a crear diferentes componentes y así reutilizar estructuras para diferentes vistas.

Veremos un ejemplo basado en la necesidad más común al momento de desarrollar sitios web que contienen varias vistas ¿Adivinas cuál es? Así es, la reutilización del menú de navegación.

Aprendiendo a reutilizar parciales o componentes podrás desarrollar con mucha más comodidad, al no tener un solo documento HTML con cientos o miles de líneas de código, sino diferentes archivos que representan diferentes piezas de la aplicación.

Parciales

Handlebars al igual que los demás motores de plantillas permite la reutilización de plantillas a través de partials. Los partials son plantillas normales de handlebars, que otras plantillas pueden llamar directamente y así formar un rompecabezas, en donde podemos tener una estructura principal y llamar a las piezas que conformarán la vista.

De forma predeterminada los partials son buscados en una carpeta llamada "partials" dentro de la carpeta "view", no obstante, se puede configurar esta ruta y definir el nombre que quieras para la carpeta que contendrán las piezas de nuestra aplicación ¿Pero en dónde podemos definir el directorio en el que escribiremos nuestros parciales? Por supuesto, en el mismo método engine que usamos en el capítulo anterior.

Importando Parciales

Los parciales (al igual que una función en programación) son estructuras o porciones de código que podemos reutilizar a través de su llamado, pero ¿Cómo las importamos? La forma de invocar un partial dentro de una vista es con las doble llaves (**{{ }}**) y con el signo de "mayor" (>). Por ejemplo, si quisiéramos importar un parcial de nombre "Menu" en un layout, usamos la siguiente sintaxis:

```
{{> Menu }}
```

Envío de parámetros a los partials

Se ha mencionado que el concepto de "parciales" es comparable al concepto de "funciones", por el hecho de ser estructuras reutilizables, pero incluso además de eso, también comparten la cualidad de poder recibir parámetros, por ejemplo, podemos definir una estructura que renderice de la misma manera pero un contenido diferente.

La forma de enviar variables o "argumentos" a un parcial, es la que te muestro en la siguiente sintaxis, en donde tenemos un parcial llamado "Card" y le estamos enviando 3 variables: nombre, imagen y descripción:

```
{{> Cards
     nombre="Inteligencia Artificial"
     imagen="img/inteligencia-artificial.jpg"
     descripcion="Es la inteligencia llevada a cabo por máquinas."
 }}
```

El resultado de este ejemplo sería como lo que muestro en la siguiente imagen:

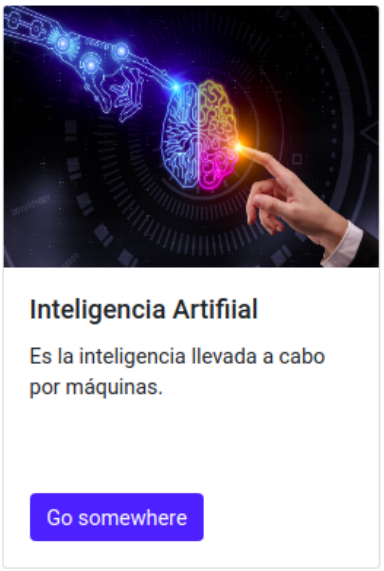

Imagen 5. Demostración gráfica del uso de parámetros enviados a partials.
Fuente: Desafío Latam

Como puedes notar, la información dentro de la tarjeta es la que estamos definiendo como parámetros del parcial. Ahora que entendemos que se pueden importar parciales y pasarle datos para que sean renderizados dentro de su estructura, procedamos con algún ejercicio y pongamos esto en práctica.

Ejercicio guiado: Menú compartido

Desarrollar un servidor con Express que disponibilice 3 rutas para renderizar 3 vistas diferentes: Inicio, Galería y Contactos. El objetivo será utilizar los parciales de handlebars para que las 3 vistas compartan el mismo menú de navegación.

Consideraciones previas

Necesitamos realizar lo siguientes puntos antes de comenzar con el ejercicio:

1. Crear una carpeta “componentes” dentro de la carpeta “views” y crea un archivo llamado “Menu.handlebars” con el siguiente código:

```
<ul style="background: black; padding: 5px">
    <li><a href="/"> Inicio </a></li>
    <li><a href="/Galeria"> Galeria </a></li>
    <li><a href="/Contactos"> Contactos </a></li>
    <li>Número: {{numero}} </a></li>
</ul>

<style>
    li { display: inline-block }
    a, li { color: white }
</style>
```

Como puedes ver, queremos en el último ítem de la lista imprimir el valor de un parámetro, eso se parece bastante a la importación de un componente, pero la diferencia está en que en la renderización de variables no se usa el símbolo “mayor” (>).

2. Crear un archivo llamado “Inicio.handlebars” dentro de la carpeta “views” con el siguiente código:

```
{{> Menu numero="1"}}
<h1>Esta es la página de Inicio</h1>
```

3. Crear un archivo llamado “Galeria.handlebars” dentro de la carpeta “views” con el siguiente código:

```
{{> Menu numero="2" }}
<h1>Esta es la página de Galería</h1>
```

4. Crear un archivo llamado “Contactos.handlebars” dentro de la carpeta “views” con el siguiente código:

```
{{> Menu numero="3"}}
<h1>Esta es la página de Contactos</h1>
```

Como puedes notar, estamos pasándole al parcial “Menu” un parámetro “numero” en cada template con números del 1 al 3 respectivamente, esto lo hacemos para comprobar el paso de parámetros a un parcial o componente. Ahora que tenemos nuestras vistas y componente creado, sigue los siguientes pasos para la creación de nuestro servidor:

- ***Paso 1:*** Crear un servidor con Express que integre handlebars como motor de plantillas.

- ***Paso 2:*** Usar el método "engine" para definir el objeto de configuración de handlebars, especificando que la ruta para las vistas será **/views** y la ruta en donde se encontrarán los parciales o componentes será **/views/componentes.**

- ***Paso 3:*** Crear una ruta raíz **GET /** que utilice el método render, recibiendo como primer parámetro "Inicio" y como segundo un objeto, que contenga la propiedad "layout" y cuyo valor será "Inicio". Este objeto será el que defina que se utilizará el archivo **Inicio.handlebars** para renderizar esta vista.

- ***Paso 4:*** Crea una ruta raíz **GET /Contactos** que utilice el método render, recibiendo como primer parámetro "Contactos" y como segundo un objeto, que contenga la propiedad "layout" y cuyo valor será "Contactos".

- ***Paso 5:*** Crea una ruta raíz **GET /Galeria** que utilice el método render, recibiendo como primer parámetro "Galeria" y como segundo un objeto, que contenga la propiedad "layout" y cuyo valor será "Galeria".

```
// Paso 1
const express = require("express");
const app = express();
const exphbs = require("express-handlebars");

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

app.set("view engine", "handlebars");

// Paso 2
app.engine(
  "handlebars",
  exphbs({
    layoutsDir: __dirname + "/views",
    partialsDir: __dirname + "/views/componentes/",
  })
);

// Paso 3
app.get("/", function (req, res) {
  res.render("Inicio", { layout: "Inicio" });
});
```

```
// Paso 4
app.get("/Contactos", function (req, res) {
  res.render("Contactos", { layout: "Contactos" });
});

// Paso 5
app.get("/Galeria", function (req, res) {
  res.render("Galeria", { layout: "Galeria" });
});
```

Probemos esto, levanta el servidor e ingresa a las siguientes direcciones una por una:

- http://localhost:3000/
- http://localhost:3000/Galeria
- http://localhost:3000/Contactos

Deberás ver lo que te muestro en las siguientes imágenes:

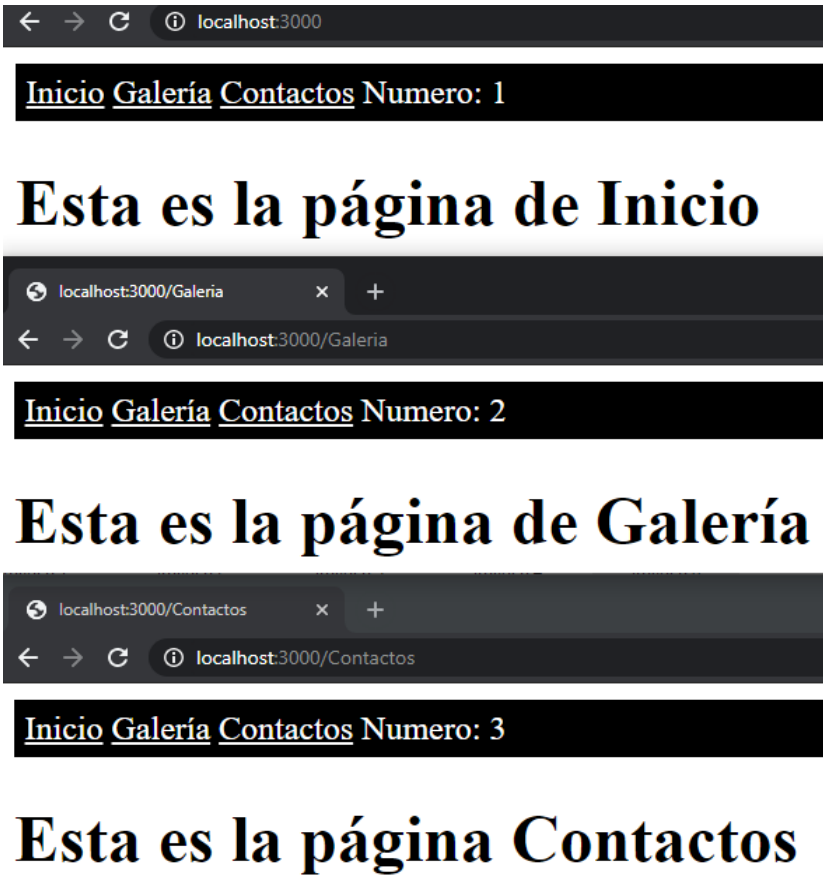

Imagen 6. Renderización de vistas que comparten el mismo parcial.
Fuente: Desafío Latam

Como puedes ver, se está renderizando el componente “Menu” en las 3 vistas y de esta manera hemos evitado tener que copiar y pegar el mismo código, además, hemos comprobado el paso de parámetro al parcial con valores diferentes en cada vista.

Ejercicio propuesto (3)

Desarrollar un servidor con Express que renderice una plantilla con handlebars para mostrar una etiqueta <img> cuyo src sea enviado como parámetro en su importación desde una vista principal.

Helpers

Competencia

- Construir un servidor con Express que utilice helpers para la renderización dinámica de datos.

Introducción

En el capítulo anterior aprendiste cómo separar tu sitio web en varios pedazos gracias a los parciales, además cómo renderizar vistas con formato handlebars en diferentes rutas desde servidor. Ahora es momento de aprender la segunda pero no menos importante razón por la cual utilizamos motores de plantillas, los helpers.

Los helpers son funciones que podemos ocupar dentro de nuestras plantillas, para renderizar elementos pertenecientes a un arreglo dentro de los parámetros de una vista y/o condicionar contenido según un criterio booleano.

En este capítulo, aprenderás a ocupar los helpers principales de handlebars como lo son: if, unless y each. Con estos helpers podrás mostrar contenido de forma dinámica, es decir, que la información que mostramos en nuestros sitios web dependerá de los parámetros que definamos para una vista o un parcial.

Helpers

Los Helpers son funciones ofrecidas por handlebars para la reproducción de datos que recibimos en los parciales, bien sea por parámetro definido en su llamado desde otro parcial o por definición desde el render. La forma de importarlos es muy parecida a la de un parcial, pero la diferencia es el símbolo, en este caso es un hash o también conocido como "gato" (#).

Entre los helpers de este motor de plantilla tenemos:

- **if:** Funciona únicamente con la condición booleana, es decir, evalúa si el dato es true o false y su sintaxis es la siguiente:

```
{{#if nombre }}
<h3>Bienvenido: {{ nombre }} </h3>
{{/if }}
```

Como puedes notar, no se usan paréntesis para definir la condición ni llaves para delimitar el bloque de renderización, sino que se habilita el inicio del helper, seguido de las instrucciones que queremos mostrar y para cerrar el bloque correspondiente al "if", se ocupa la misma sintaxis pero con el slash "/" seguido del nombre del helper.

- **Else:** Debe existir siempre dentro del bloque de renderización del if y su sintaxis es la siguiente:

```
{{#if nombre }}
<h3>Bienvenido: {{ nombre }} </h3>

{{else}}

<h3>Aún no ha ingresado ningún usuario</h3>

{{/if }}
```

- **Unless:** Parecido al if, este helper sólo muestra contenido si el dato en evaluación es "false", por ejemplo, cuando se intenta ingresar a un sistema y el usuario enviado a la validación no se encuentra. Con el siguiente ejemplo, te muestro como poder enviar parámetros o variables directamente a una vista desde el render:

```
app.get('/', function (req, res) {
 res.render('Home', {
   isLogged: false
 })
})
```

Como puedes ver, es igual a la definición que usamos para especificar el layout de una vista, sin embargo, la propiedad "layout" se puede considerar como una palabra reservada de handlebars, por lo que definir una propiedad con cualquier otro nombre será interpretado por el mismo motor de plantilla como un parámetro para ese parcial.

La renderización dentro del parcial para el "unless" sería como te muestro a continuación:

```
{{#unless isLogged}}
 <p>El usuario no está registrado</p>
{{/unless}}
```

- **Each:** Es el iterador de arreglos que podemos usar dentro de las plantillas, su sintaxis incluye la palabra reservada "this" para indicar cada elemento del arreglo, muy parecido al iterador each de jQuery o al forEach de vanilla js.

A continuación te muestro un ejemplo en el que estamos iterando un arreglo que contiene como elementos cadenas de texto con el nombre de los 3 lenguajes principales del frontend. En este ejemplo, te mostraré el envío de parámetros desde el método render dentro de la ruta en el servidor:

```
app.get("/", function (req, res) {
  res.render("Inicio", {
    layout: "Inicio",
    lenguajesFrontend: ["HTML", "CSS", "Javascript"],
  });
});
```

¿Qué estoy haciendo aquí? Estoy pasando un parámetro a la vista "Inicio" al agregar una propiedad "lenguajesFrontend", la cual contiene un arreglo de Strings.

Ahora te muestro como sería el código dentro de la vista:

```
<ul>
   {{#each lenguajesFrontend}}

   <li>{{this}}</li>

   {{/each}}
</ul>
```

Observa con detención, estoy integrando la etiqueta <li> de una lista desordenada dentro del bloque "each" de handlebars, ¿Por qué de esta manera? Porque así logramos que este elemento se repita por cada iteración del arreglo lenguajesFrontend e imprima el valor del elemento con el "this".

El resultado de este ejemplo sería como lo que muestro en la siguiente imagen:

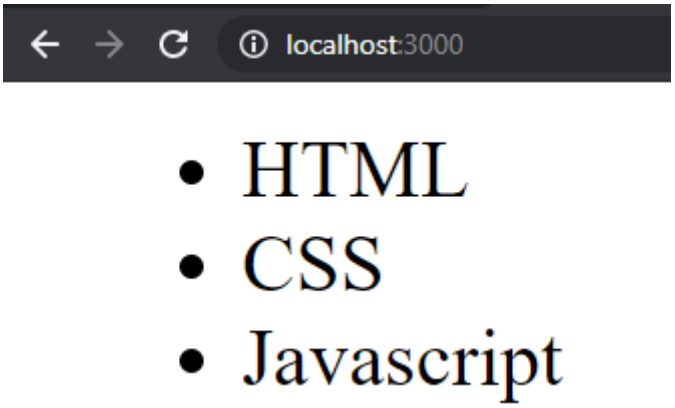

Imagen 7. Renderización de una vista con el helper iterador each.
Fuente: Desafío Latam

Como se puede ver, se están mostrando los 3 elementos del arreglo dentro de cada etiqueta <li>. Ahora veamos lo aprendido en un ejercicio diferente y un poco más completo para poner a prueba nuestros nuevos conocimientos.

Ejercicio guiado: Ver mis ventas

Agregar a nuestro servidor una ruta **GET /ventas** que use los helpers each, if y unless para mostrar las ventas de un usuario, enviadas como parámetro en el método render.
Sigue los siguientes pasos para resolver este ejercicio:

- ***Paso 1:*** Crear una ruta **GET /ventas** en el servidor.
- ***Paso 2:*** Utilizar el método render para renderizar una vista "Ventas" que pase como parámetro las propiedades "usuario" con el nombre de usuario y un arreglo de números llamado "ventas".

```
// Paso 1
app.get("/ventas", function (req, res) {
  // Paso 2
  res.render("Ventas", {
    layout: "Ventas",
    usuario: "Maria José",
    ventas: [14990, 42490, 22500],
  });
});
```

- ***Paso 3***: Crear un archivo en la carpeta "views" que se llame "Ventas.handlebars" y que contenga el helper "if" para validar que existe valor en el parámetro usuario.

- ***Paso 4:*** Utilizar el helper "each" para iterar el arreglo "ventas" e imprimir sus elementos dentro de una lista desordenada.

- ***Paso 5:*** Utilizar el helper "unless" para imprimir un mensaje que indique que se debe iniciar sesión en caso de que el parámetro "usuario" sea "false".

```
{{!-- Paso 3 --}}
{{#if usuario}}
<h1>
    Bienvenid@: {{usuario}}
</h1>
<h3>Tus ventas hoy fueron las siguientes:</h3>
<ul>
    {{!-- Paso 4 --}}
    {{#each ventas}}
    <li>${{this}}</li>
    {{/each}}
</ul>
{{/if}}

{{!-- Paso 5--}}
{{#unless usuario}}

<h1>Inicie sesión para ver sus ventas</h1>

{{/unless}}
```

Ahora levanta el servidor y en caso de darle valor a la propiedad “usuario” obtendrás algo similar a la siguiente imagen:

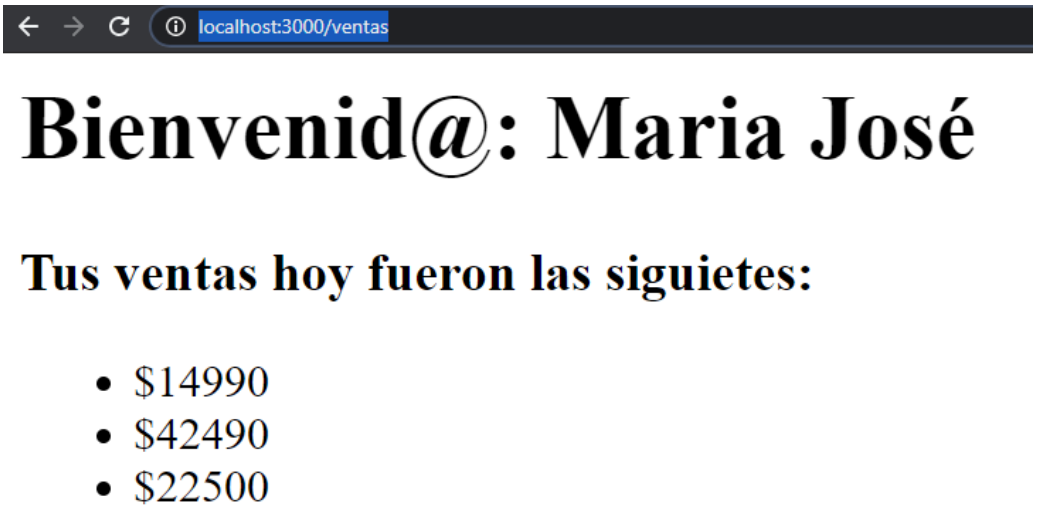

Imagen 8. Renderización de una vista, caso de usuario con valor.
Fuente: Desafío Latam

Como puedes ver, hemos recibido la renderización del bloque if y dentro la iteración del arreglo “ventas” gracias al helper “each”.

¿Qué sucedió con el bloque unless? Recordemos que este bloque mostrará contenido si el parámetro que definimos es “false”. Para comprobarlo cambia el valor de la propiedad “usuario” dentro del método “render” por “false” y si vuelves a consultar la ruta **/ventas** verás lo que te muestro en la siguiente imagen:

Imagen 9. Renderización de una vista, caso de usuario sin valor o con valor false.
Fuente: Desafío Latam

Excelente, ahora el contenido que mostramos es dinámico y dependerá del valor de la propiedad usuario y del arreglo ventas. Considera que esto es muy escalable y podemos mezclarlo con la obtención de parámetros o payload enviados desde una aplicación cliente, y de esta manera construir sistemas cada vez más complejos y dinámicos.

Ejercicio propuesto (4)

Desarrollar una ruta que renderice una vista “Pendientes”, que itere un arreglo de tareas por hacer con el helper “each” de handlebars y en caso de no haber tareas imprimir un mensaje que diga “No hay tareas pendientes por hacer”.

Devolviendo un sitio web de contenido dinámico

Competencia

- Construir un servidor utilizando Express para servir un sitio web con contenido dinámico utilizando el motor de plantillas handlebars.

Introducción

Es momento de consolidar toda la información que has recibido en esta lectura. En este capítulo realizaremos un ejercicio en el que crearemos un servidor con express y handlebars para mostrar una vista con 8 botones de bootstrap, los cuales al ser presionados cambiarán el color de un spinner de forma dinámica.

Al finalizar este ejercicio estarás listo para empezar a construir sitios web con contenido dinámico utilizando los parciales y helpers de Handlebars.

Ejercicio guiado: Spinner de colores

Desarrollar un servidor que reciba en su ruta raíz un parámetro con el nombre de un color según las variantes de colores de bootstrap. Este parámetro será enviado a través del render a una vista "Inicio", la cual importará 2 parciales:

- **Botones:** Contendrá el helper "each" para iterar un arreglo de colores.
- **Spinner:** Contendrá el componente spinner de Bootstrap, en donde su variante de color será igual a un parámetro "color" enviado desde el render.

El objetivo será programar que al ser presionado un botón de color "success", por ejemplo, se consulte al servidor pasando como parámetro de la URL esta variante de color y ¿Cómo redireccionaremos al usuario luego del click en el botón? Usaremos las etiquetas de ancla, concatenando de forma dinámica el color en el atributo "href".

La siguiente imagen es una referencia de lo que debemos lograr al finalizar este capítulo:

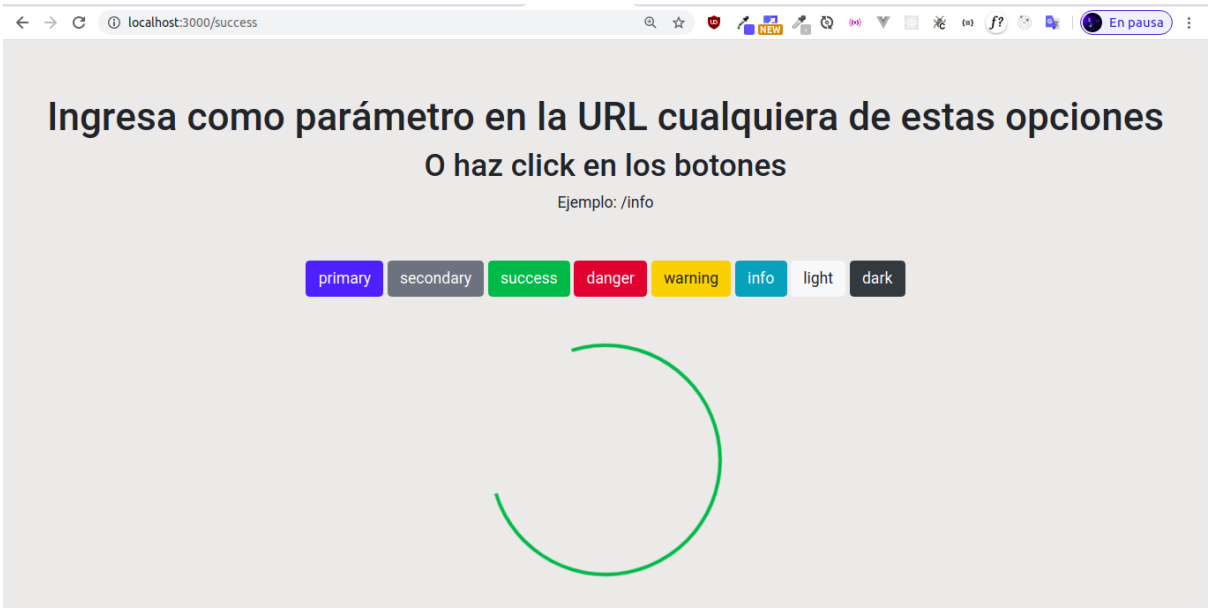

Imagen 10. Renderización de parciales y helpers.
Fuente: Desafío Latam

Sigue los siguientes pasos para desarrollar este ejercicio progresivo que estará dividido en 2 partes: Plantillas y Servidor.

Parte I: Plantillas

- ***Paso 1:*** Crear una plantilla dentro de la carpeta “views” llamado “Inicio.handlebars” que contenga el siguiente código:

```
<link rel="stylesheet" href="/css/bootstrap.min.css">

<div class="contain d-flex flex-column text-center
justify-content-center align-items-center">
  <h1>Ingresa como parámetro en la URL cualquiera de estas opciones</h1>
  <h2>O haz clic en los botones</h2>
  <span>Ejemplo: /info</span>
  <div class="m-5">
    {{> Botones }}
  </div>

  <div>
    {{> Spinner}}
  </div>
</div>
<style>
  * {
    margin: 0;
  }

  .contain {
    background: #ece9e9;
    height: 100vh;
  }
</style>
```

En esta plantilla lo que hacemos es:

  - Importar Bootstrap.
  - Diseñar la maqueta con código HTML y clases de Bootstrap.
  - Importar 2 parciales: **Botones** y **Spinner.**

- ***Paso 2:*** Crear una plantilla dentro de la carpeta “componentes” llamada “Botones” y escribe el siguiente código.

```
{{#each colores}}
<a href="/{{this}}"> <button class="btn btn-{{this}}">{{this}}</button>
</a>
{{/each}}
```

En esta plantilla estamos iterando un arreglo llamado “colores” que será pasado como parámetro dentro del método render, es importante que sepas que aunque este parámetro se está pasando a la vista “Inicio”, todos los parciales hijos también podrán acceder a este dato.

Nota que además estamos renderizando el valor del “this”, concatenandolo en el atributo href de la etiqueta ancla para hacer la consulta de forma dinámica al servidor y mostrando los botones en sus diferentes variantes.

- ***Paso 3:*** Crear una plantilla dentro de la carpeta “componentes” llamada “Spinner” y escribe dentro el siguiente código:

```
<div class="spinner">
  <div class="spinner-border text-{{color}}" style="width: 15rem;
height: 15rem;">
  </div>
</div>
```

Esta plantilla estará mostrando el componente spinner de Bootstrap, pero la variante de color será un parámetro “color” enviado desde el render en el servidor.

Parte II: Servidor

- ***Paso 1:*** Crear un servidor con Express e integra handlebars definiendo en su configuración las rutas para el consumo de vistas y componentes.

- ***Paso 2:*** Crear un middleware que publique la carpeta “css” de Bootstrap localizada dentro de los node_modules.

- ***Paso 3:*** Crear una ruta raíz que reciba un parámetro “color”.

- ***Paso 4:*** Utilizar destructuring para extraer la propiedad color de los parámetros de la consulta con el objeto request.

- ***Paso 5:*** Ocupar el método render para renderizar la plantilla Inicio y pasar como parámetros un arreglo con todas las variantes de colores de Bootstrap y una propiedad “color” cuyo valor será el recibido como parámetro en la ruta.

```
// Paso 1
const express = require("express");
const app = express();
const exphbs = require("express-handlebars");

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

app.engine(
  "handlebars",
  exphbs({
    layoutsDir: __dirname + "/views",
    partialsDir: __dirname + "/views/componentes/",
  })
);

app.set("view engine", "handlebars");

// Paso 2
app.use("/css", express.static(__dirname +
"/node_modules/bootstrap/dist/css"));

// Paso 3
app.get("/:color", function (req, res) {
  // Paso 4
  const { color } = req.params;
  // Paso 5
  res.render("Inicio", {
    layout: "Inicio",
    colores: [
      "primary",
      "secondary",
      "success",
      "danger",
      "warning",
      "info",
      "light",
      "dark",
    ],
    color: color,
  });
});
```

Ha llegado el momento de la verdad, levanta el servidor y consulta la siguiente ruta: http://localhost:3000/danger, deberás ver lo que te muestro en la siguiente imagen:

Imagen 11. Cambio de color del spinner según parámetro de la URL.
Fuente: Desafío Latam

Muy bien, se está renderizando todo sin problema, sin embargo, falta probar lo más importante, que es el cambio dinámico del color del spinner, para esto haz click en el botón "primary" y deberás recibir lo que te muestro en la siguiente imagen:

Imagen 12. Cambio dinámico del color del spinner luego de presionar un botón.
Fuente: Desafío Latam

¡Felicitaciones! Lo has logrado, con este ejercicio terminado estás listo para empezar a construir sistemas web más completos que gestionen contenido de forma dinámica desde el servidor con handlebars.

No está demás mencionar que con esta arquitectura puedes mezclar la programación con JavaScript para el lado del cliente y del servidor, logrando con esto la unión de ambos mundos y la expansión de funcionalidades en tus aplicaciones.

Ejercicio propuesto (5)

Desarrollar un servidor con Express que tenga integrado handlebars y Bootstrap para la renderización de una plantilla “Inicio” que importe un parcial “Table”.

En este parcial deberás usar el helper “each” para mostrar la información de diferentes usuarios pasados como parámetros en el render de la ruta raíz.

Resumen

En esta lectura aprendimos a utiliza el motor de plantillas handlebars para la renderización de contenido dinámico, distribuido en parciales que utilizan helpers para la impresión de datos, abordando los siguientes contenidos:

- Cómo importar los códigos fuentes de Bootstrap y jQuery dentro de un sitio web estático servido con express.
- Qué son los motores de plantilla y por qué utilizarlos en el desarrollo de sitios web con contenido dinámico.
- Cómo integrar y configurar Handlebars en un servidor con Express para la renderización de una vista.
- Qué son los parciales y cómo importarlos dentro de una vista renderizada por Handlebars.
- Definición y uso de los Helpers realizando un embebido de lógica de JavaScript dentro de un parcial.
- Creación de un sitio web de contenido dinámico que utiliza los parciales y helpers para cambiar el color de un spinner de Bootstrap.

Solución de los ejercicios propuestos

1. Desarrollar un servidor con Express que sirva en su ruta raíz un sitio web estático que muestre un navbar responsivo y un footer de Bootstrap.

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

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

app.use('/bootstrap', express.static(__dirname +
'/node_modules/bootstrap/dist/css'))
app.use('/jquery', express.static(__dirname +
'/node_modules/jquery/dist'))
app.use('/BootstrapJs', express.static(__dirname +
'/node_modules/bootstrap/dist/js/'))

app.get("/", (req, res) => {
  res.sendFile(__dirname + '/index.html')
})
```

```
<link rel="stylesheet" href="/bootstrap/bootstrap.css" />
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="#navbarNavAltMarkup"
    aria-controls="navbarNavAltMarkup"
    aria-expanded="false"
    aria-label="Toggle navigation"
  >
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
    <div class="navbar-nav">
      <a class="nav-item nav-link active" href="#"
        >Ejemplo de NavBar <span class="sr-only">(current)</span></a
```

```
      >
    </div>
  </div>
</nav>

<footer class="footer">
  <div class="container">
    <span class="text-muted">Place sticky footer content here.</span>
  </div>
</footer>

<style>
  .footer {
    position: absolute;
    bottom: 0;
    width: 100%;
    height: 60px;
    background-color: black;
  }
</style>
<script src="/jquery/jquery.js"></script>
<script src="/BootstrapJs/bootstrap.js"></script>
```

2. Desarrollar un servidor con Express que renderice una plantilla con un formulario HTML a través del motor de plantillas handlebars.

```
const express = require("express");
const app = express();
const exphbs = require("express-handlebars");

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

app.set("view engine", "handlebars");

app.engine(
  "handlebars",
  exphbs({
    layoutsDir: __dirname + "/views",
  })
);
```

```
app.get("/", (req, res) => {
  res.render("main");
});
```

```
<!DOCTYPE html>
<html>

<head>
   <meta charset="utf-8">
   <title>Document</title>
</head>

<body>
   <form>
      <label>Nombre:</label>
      <input type="text"> <br>
      <label>Contraseña:</label>
      <input type="text"> <br> <br>
      <button>Iniciar sesión</button>
   </form>
</body>

</html>
```

3. Desarrollar un servidor con Express que renderice una plantilla con handlebars para mostrar una etiqueta <img> cuyo src sea enviado como parámetro en su importación desde una vista principal.

```
const express = require("express");
const app = express();
const exphbs = require("express-handlebars");

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

app.set("view engine", "handlebars");

app.engine(
  "handlebars",
  exphbs({
```

```
    layoutsDir: __dirname + "/views",
    partialsDir: __dirname + "/views/componentes/",
  })
);

app.get("/", function (req, res) {
  res.render("Inicio", { layout: "Inicio" });
});
```

```
{{> Imagen
src='https://i.pinimg.com/originals/66/9c/df/669cdfa60c802602c9fe2bd6dd2
b1537.jpg'}}
<h1>Esta es la página de Inicio</h1>
```

```
<img src="{{src}}">
```

4. Desarrollar una ruta que renderice una vista “Pendientes” que itere un arreglo de tareas por hacer con el helper “each” de handlebars y en caso de no haber tareas, imprimir un mensaje que diga “No hay tareas pendientes por hacer”.

```
app.get("/pendientes", function (req, res) {
  const tareas = ["Ir al médico", "Subir el cerro", "Hacer aseo"];
  res.render("Pendientes", {
    layout: "Pendientes",
    tareas: tareas.length > 0 ? tareas : false,
  });
});
```

```
{{#if tareas}}
<h3>Tus tareas pendientes son:</h3>
<ul>
    {{#each tareas}}
    <li>{{this}}</li>
    {{/each}}
</ul>
{{/if}}

{{#unless tareas}}

<h5>No tiene tareas pendientes por hacer</h5>
```

```
{{/unless}}
```

5. Desarrollar un servidor con Express que tenga integrado handlebars y Bootstrap para la renderización de una plantilla “Inicio” que importe un parcial “Table.

   En este parcial deberás usar el helper “each” para mostrar la información de diferentes usuarios enviados como parámetros en el render de la ruta raíz.

```
const express = require("express");
const app = express();
const exphbs = require("express-handlebars");

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

app.set("view engine", "handlebars");

app.use("/css", express.static(__dirname +
"/node_modules/bootstrap/dist/css"));

app.engine(
  "handlebars",
  exphbs({
    layoutsDir: __dirname + "/views",
    partialsDir: __dirname + "/views/componentes/",
  })
);

app.get("/usuarios", function (req, res) {
  res.render("Usuarios", {
    layout: "Usuarios",
    usuarios: [
      {
        nombre: "Luis",
        direccion: "Avenida Libertador",
        telefono: "+56978994561",
      },
      {
        nombre: "Francisco",
        direccion: "Avenida Luis del valle Garcia",
```

```
        telefono: "+56935761594",
      },
      {
        nombre: "Diana",
        direccion: "Avenida via al Sur",
        telefono: "+56913467946",
      },
    ],
  });
});
```

```
<link rel="stylesheet" href="/css/bootstrap.min.css">

<div class="container">
    <h1 class="text-danger text-center">Tabla de usuarios</h1>

    {{> Tabla}}
</div>
```

```
<table class="table">
    <thead>
        <th>Nombre</th>
        <th>Dirección</th>
        <th>Teléfono</th>
    </thead>
    <tbody>
        {{#each usuarios}}
        <tr>
            <td>{{this.nombre}}</td>
            <td>{{this.direccion}}</td>
            <td>{{this.telefono}}</td>
        </tr>
        {{/each}}
    </tbody>
</table>
```