Aquí cubriremos varios temas como:

1. Uso y configuración de Express
2. Servir contenido estático
3. Template engines
4. Handebars
   1. Helpers
   2. Parciales
   3. Variables
5. Despliegues en Heroku y Github
6. Hacer carpetas públicas en la web

**INICIAMOS PORYECTO**

Npm init

Archivo app.js

Vamos a pagina Node vamos a docs y vamos al método HTTP, es el que nos permite crear un webserver

const http = require('http');

//SE CREA EL SERVIDOR

http.createServer((req, res) => {

    res.write('Hola Mundo');

  res.end();

})

.listen(8080);

console.log('Escuchando el puerto 8080');

exportamos el paquete interno de Node http, creamos el servidor con el método createServer, que es un callbal de los request y de las res. (método write para escribir algo en el servidor y el método listen para especificarle un puerto de salida. Y basta con escribir node app en la consola.Y el método end para terminar de compilar lo que tiene write.

O podemos mandar un json, para crear nuestra API

 //probando mandar JSON

    res.writeHead(200, { 'Content-Type': 'application/json' });

    let salida = {

        nombre: 'Mario',

        edad: 26,

        url: req.url

    }

    res.write(JSON.stringify(salida));

    //   res.write('Hola Mundo');

    res.end();

/data/usuario/1 y funciona la peticion. Pero ya esta ese paquete que hace el trabajo por nosotros y es **EXPRESS**

**EXPRESS**

**Lo instalamos con npm install express –-save**

**Hacemos el archivo app y corremos en la consola con node app**

const express = require('express')

const app = express()

app.get('/', (req, res) => {

    res.send('Hello World')

})

app.listen(3000)

Vamos a hacer que envie json como en el ejercicio pasado

El método send ya automatiza el proceso de convertir los objetos en formato JSON

app.get('/', (req, res) => {

    //

    let salida = {

        nombre: 'Mario',

        edad: 26,

        url: req.url

    }

    res.send(salida);

})

Esto es como un middleware que filtra cualquier tipo de petición por la url ‘/’

**SERVIR CONTENIDO ESTATICO**

Creamos una capeta publica que es la que va a contener lo que va a ver el usuario final y dentro de esta carpeta hicimos un HTML, entonces tenemos que hacer uso de un middleware que haga que no importa que dirección mande el usuario, siempre va a ir a la capeta publica.

const app = express()

app.use(express.static(\_\_dirname + '/public'));

Todo lo que este en la capeta publica es lo que se va a mostrar, archivos css, html, js etc.

**CREAR UNA PEQUEÑA PAGINA WEB**

Descargamos el sourcefiles de boostrap4 ysamos de la caprta bin los css y los js

Y los enlacamos al html

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

Y usaremos los components que nos da boostrap por defecto.

Insertamos un jumbotron

<script src="assets/js/bootstrap.min.js"></script>

Y también el jquery

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>

    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN" crossorigin="anonymous"></script>

    <script src="assets/js/bootstrap.min.js"></script>

Y hacemos en el nav los links hacia las otras paginas

footer {

    background-color: #343a40;

    color: white;

    width: 100%;

    position: fixed;

    bottom: 0px;

    text-align: center;

    padding: 20px;

}

Hacemos el link para integrar el footer…

Ahora para no repetir todo el código del header y el footer express maneja una paqueteria que se llama Haldebars y hacer la pagina dinámica.

**HANDLEBARS**

Existe temples engine con los cuales express ya trabaja, es aquí donde entran los frameworks angular, react, vue.js ya entran y que esos frameworks ya los maneje de manera automica. Pero en este caso no estamos usando esos frameworks, lo que vamos a usar es “bigotes” o handlebars.

Nosotros vamos a trabajar con la librería **hbs npm --save**

Este paquete renderiza las paginas como hnadlebars

Colocamos el renderizador en el archivo del servidor con el método set.

app.use(express.static(\_\_dirname + '/public'));

//renderizador

app.set('view engine', 'hbs');

hicimos una capeta en la raíz del proyecto que se llama vamos donde vamos a tener todas vitas y los archivos son extenciones hbs

para que renderize la nueva vista usamos el método render el la pestaña / inicio

//home

app.get('/', (req, res) => {

    res.render('home')

})

Podemos poner las variables dinámicas en el html de la siguiente forma entre {{}}

 <footer>

        Mario Garcia - {{anio}}

    </footer>

Y las variables se las podemos poner en el método get.

app.get('/', (req, res) => {

    res.render('home', {

        nombre: 'Mario',

        anio: new Date().getFullYear()

    })

})

**USANDO PARCIALES CON HBS**

Son bloques de código html que nosotros podemos reutilizar y lo vamos a usar con un folder, no olvidar importarlo.

Ahora usamos el método del hbs registerPatials para mandarle el directorio completo de las vistas, en este caso hicimos una capeta nueva que se llama particals dentro de views.

const hbs = require('hbs');

//renderizador

hbs.registerPartials(\_\_dirname +'/views/parciales');

app.set('view engine', 'hbs');

vamos a crear el primer parcial

ahora como reconoce la ruta de los parciales vamos a integrarlo a la vista que queramos en la sección correspondiente en el html.

   {{> navbar}} > debe de estar pegando al { y deben de llamarse igual

Y asi lo hacemos con los demás parciales o componentes.

Para que nodemon detecte los cambios usamos que tipos de archivos debe de estar pendiente de esos cambios

Nodemon server -e js,hbs,html,css

El home quedo asi despyues de haber echo lo mismo con el header, navbar y el footer.

{{> header}}

    {{> navbar}}

    <div class="jumbotron">

        <h1 class="display-4">Hello, {{nombre}}</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>

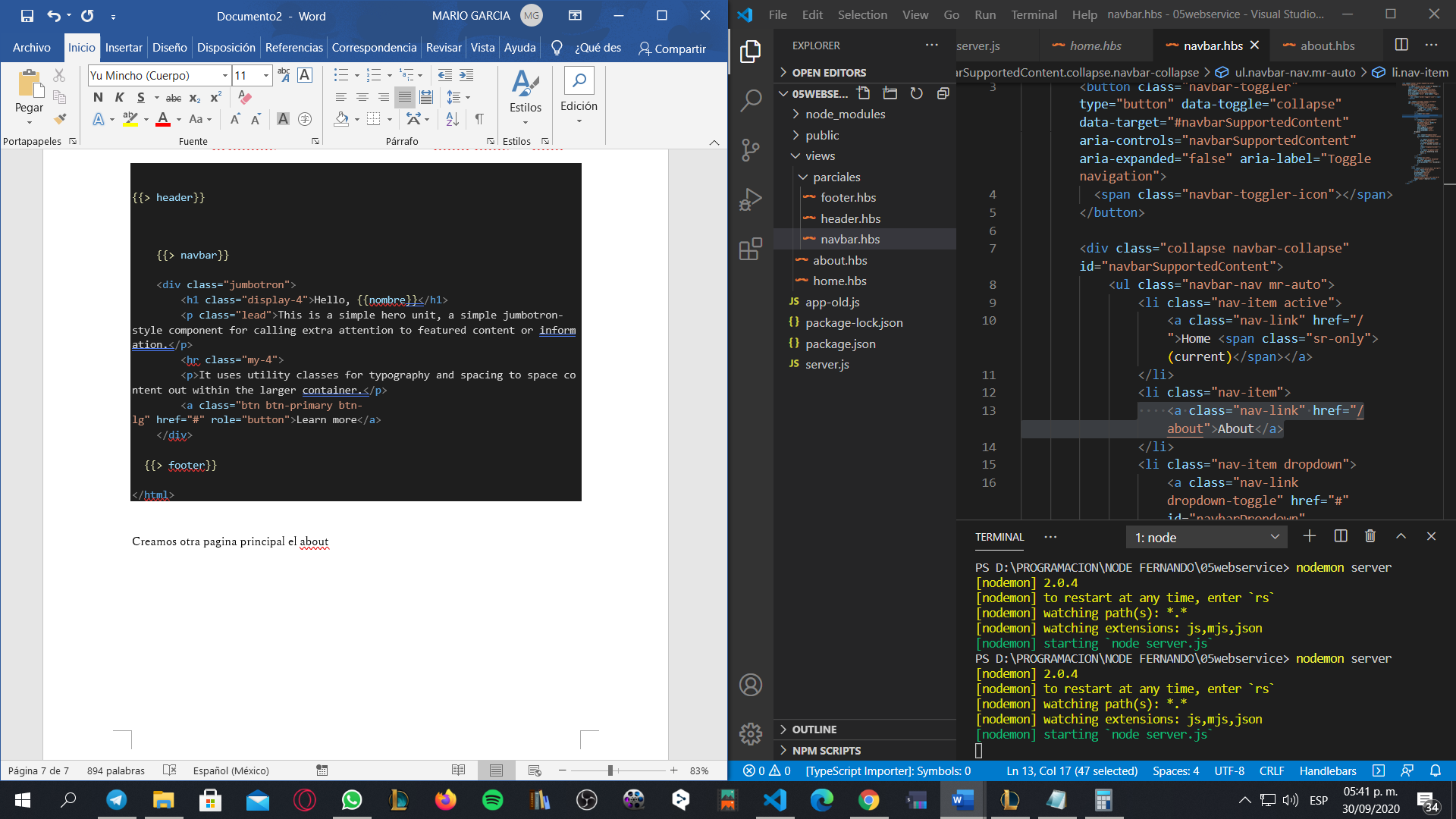
        <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>

    </div>

  {{> footer}}

</html>

Creamos otra pagina principal el about e hicimos los pasos de renderizar nuevamente



//about

app.get('/about', (req, res) => {

    res.render('about', {

        anio: new Date().getFullYear(),

    })

})

Para direccionar en el navbar al about se hace en el render

    <a class="nav-link" href="/about">About</a>

Y EL ARCHIVO DE ABOUTlo trabajamos igual que el home

**HELPERS**

Ahora bien el código que tenemos en el render ‘/’ y el ‘about’ es igual se parecen, y si quitamos el get full year por ejemplo se desaparecen.

El helper es una función que se usa en el template.

//renderizador

hbs.registerPartials(\_\_dirname + '/views/parciales');

app.set('view engine', 'hbs');

//helpers

hbs.registerHelper('getanio', () => {

    return new Date().getFullYear();

})

Se indica el nombre del helper y después una función que queremos que retorne la optenciond e al fecha, vamos al footer y usamos el helper

 Mario Garcia - {{getanio}}

Hicimos otro helper que capitaliza palabras

hbs.registerHelper('capitalizar', (texto) => {

    let palabras = texto.split(' ');

    palabras.forEach((palabra, idx) => {

        palabras[idx] = palabra.charAt(0).toUpperCase() + palabra.slice(1).toLowerCase();

    });

    return palabras.join(' ');

});

La forma de usarlo en el hbs con un espacio son las variables o los argumentos

 {{ capitalizar nombre}}

Para no tener todos los helpers en el archivo del serves creamos una carpeta nueva en la raíz del proyecto y el archivo helpers.js

const hbs = require('hbs');

//helpers

hbs.registerHelper('getanio', () => {

    return new Date().getFullYear();

})

hbs.registerHelper('capitalizar', (texto) => {

    let palabras = texto.split(' ');

    palabras.forEach((palabra, idx) => {

        palabras[idx] = palabra.charAt(0).toUpperCase() + palabra.slice(1).toLowerCase();

    });

    return palabras.join(' ');

});

Con su require por que este archivo no lo tiene

Y lo importo en el principal server.js

const hbs = require('hbs');

require('./hbs/helpers');

**SUBIENDO NUESTRA APLICACION A HEROKU**