

Práctica 06 - Web Server

Wilson Aguilar
PLATAFORMAS WEB

24 de junio de 2020

1. Configuraciones iniciales

En esta práctica nos centraremos en la creación de un sitio web dinámico haciendo uso de nodejs y algunas librerías que nos ofrece esta tecnología.

Primero creamos un nuevo proyecto de node con el comando:

```
npm init -y
```

En comparación con otros lenguajes de programación como Java o PHP donde tenemos que hacer uso de algún programa externo que haga de servidor para que pueda servir nuestros archivos, en node js lo podemos hacer de una manera mucho mas sencilla. NodeJs nos da la posibilidad de crear el servidor mediante la programación de una manera sencilla, para ello tenemos el módulo nativo http que trae node y otra es hacer uso de alguna librería que nos facilite la creación del servidor.

1.1. Módulo http de node

El módulo http de node nos permite levantar un servidor web con varias líneas de código. Podemos enviar al cliente html, archivos, etc.

Para ello empezamos creando un archivo con la siguiente configuración:



```
1  const http = require('http');
2
3  const port = process.env.PORT || 3000
4
5  http
6    .createServer((req, res) => {
7      // res.write('<h1>Hola mundo</h1>')
8      res.writeHead(200, {
9        'Content-Type': 'application/json'
10     })
11     let response = {
12       name: 'Will',
13       lastname: 'AG',
14       age: 22,
15       uri: req.url
16     }
17     res.write(JSON.stringify(response));
18     res.end();
19   })
20   .listen(port);
```

Figura 1: Servidor web con el modulo http de node

En este caso lo que mandamos al cliente son datos con formato Json. Observamos que la creación del servidor es muy sencilla pero el problema viene al momento de manejar rutas y tratar de tener contenido dinámico. Para ello tenemos muchas librerías o paquetes que nos facilitan el trabajo, entre ellas, algunas librerías famosas son:

- Express
- Nest
- Koa

1.2. Express

Express es probablemente la librería más famosa para la creación de aplicaciones web. Esta librería nos hace el trabajo mucho más fácil debido a que tiene varias cosas ya configuradas y no debemos preocuparnos por resolver problemas muy triviales.

Express está basado en el módulo http nativo de node, por lo que se lo puede integrar con este fácilmente. Para crear el servidor creamos un archivo `server.js` con el siguiente código.

A screenshot of a code editor with a dark background and light-colored text. The code is written in JavaScript and uses ES6 syntax (const, arrow functions). It imports the 'express' module, creates an Express application, sets a port (defaulting to 3000 if not specified in the environment), and listens on that port. A log message is included to confirm the server is running.

```
1  const express = require('express');
2  const app = express();
3
4  const port = process.env.PORT || 3000;
5
6
7  app.listen(port, () => {
8    console.log(
9      `Server is running at http://localhost:${port}`
10   );
11 });
```

Figura 2: Servidor con express.

En este punto tenemos nuestro servidor corriendo en el puerto definido (en este caso 300).

2. Contenido estático

Ahora lo que haremos es crear algunas páginas web en html para que nuestro servidor pueda enviar esas páginas cuando algún cliente lo solicita. La manera más común de hacerlo es crear un directorio con los archivos que se van a visualizar, y los recursos que esta va a utilizar. Para ello en la raíz de nuestro proyecto creamos un directorio público donde van a estar todos los archivos que va a poder ser accedidos desde la parte del cliente.

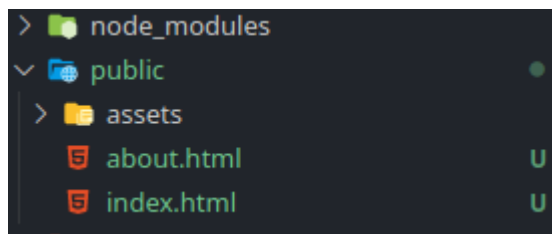


Figura 3: Directorio con archivos publicos.

En esta carpeta public van a estar los archivos html y recursos que va a mostrar el servidor. Después de crear este directorio le debemos decir a express el path completo donde se encuentra el directorio. Nos dirigimos al archivo principal y agregamos lo siguiente:



Figura 4: Definiendo carpeta publica.

3. Motor de platillas

Realizar un sitio web haciendo uso solo de html puede ser muy tedioso ya que por cada página tenemos que repetir la misma estructura, es decir vamos a tener que escribir lo mismo una y otra vez. Los motores de plantillas nos permiten la reutilización del código html ya escrito anteriormete y dividirlos como plantillas que podemos reutilizar cuando sea necesario, esto nos ayuda a escribir solo el contenido necesario.

El motor de plantillas utilizado en la practica es handle bars, pero tambien existen otros que podriamos utilizar como son:

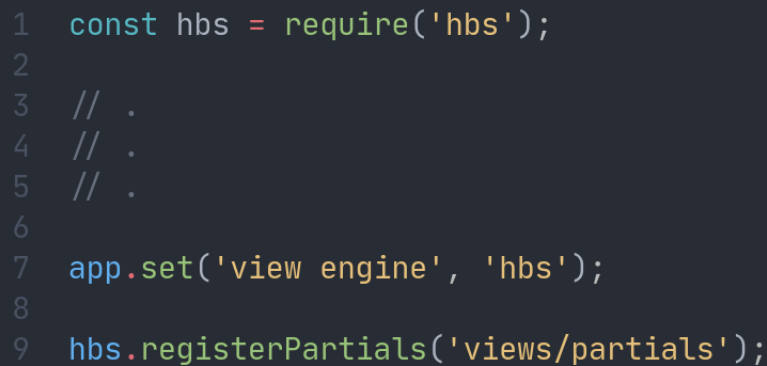
- Pug
- Ejs

3.1. Handlebars

Handlebars es un motor de plantillas que nos permite hacer más dinámico el sitio web haciendo uso de variables, funciones dentro del html. Para hacer uso de handlebars en nuestro proyecto de node primero lo instalamos haciendo uso de:

```
npm install hbs
```

Ahora en nuestro archivo principal importamos la librería y realizamos la siguiente configuración.



```
1  const hbs = require('hbs');
2
3  // .
4  // .
5  // .
6
7  app.set('view engine', 'hbs');
8
9  hbs.registerPartials('views/partials');
```

Figura 5: Configuración de handlebars.

En esta configuración lo que decimos es que hbs va a ser el motor de vistas. También utilizamos hbs para registrar los partials.

Para poder hacer uso de handlebars creamos una carpeta que se llame views y dentro de ella irán archivos con la extensión.hbs que usen html como si se tratara de cualquier otra página.

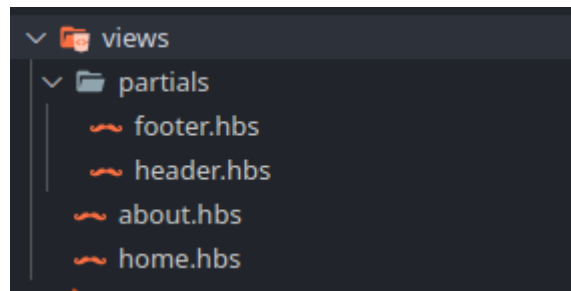


Figura 6: Configuración de handlebars.

3.2. Partials

Los partials vienen a ser nuestras plantillas, el código html que escribimos y que puede ser reutilizado. En este caso tenemos 2 partials el header y el footer ya que se van a repetir en todas las páginas que tenemos.

Para registrar los partials en nuestro archivo principal usamos el siguiente comando:

```
hbs.registerPartials('views/partials');
```

Para usar los partils lo hacemos de la siguiente manera:

```

1  {{>header }}
2  <!-- Content -->
3  <div class="jumbotron">
4    <h1 class="display-4">
5    >Static Web Page with Express andNodejs</h1>
6    <p class="lead">Plataformas Web</p>
7    <hr class="my-4">
8    <p>My name is : {{name}}</p>
9    <p>
10      {{capitalize "proBANDO teXTO paRa capITaliZAR"}}
11    </p>
12    <p>
13      Lorem ipsum, dolor sit amet consectetur adipisicing
14      elit. Repudiandae pariatur dolorum vitae recusandae
15      minus, consequatur fuga voluptate ipsa fugit exce
16      pturi soluta assumenda maxime totam error maiores corpori
17      s!
18      Atque, reprehenderit repellat.
19    </p>
20  </div>
21  {{>footer}}

```

Figura 7: Uso de partials

Simplemente usamos `{{>partial }}`. Y en esa sección se agregara el código que tenemos en nuestro partial.

3.3. Helpers

Los helpers son funciones que podemos llamar directamente desde los templates, es decir podemos llamar una funcion sencilla que nos permita capitalizar un string u obtener el año actual.

Para crear los helpers simplemente creamos un archivo aparte con la siguiente configuración:




```
1  const hbs = require('hbs');
2
3  hbs.registerHelper('getAnio', () => {
4      return new Date().getFullYear();
5  });
```

Figura 8: Registro de nuevos helpers.

Para que los helpers queden completamente registrados, simplemente hacemos un `require()` en el archivo principal, esto ayuda a que el archivo con las configuraciones cargue desde el principal.

Para usarlo simplemente entre llaves ponemos el nombre de la función. Por ejemplo `{>capitalize }`



```
1  <footer>
2      &copy;Wilson Aguilar - {{ getAnio }}
3  </footer>
```

Figura 9: Uso de helpers.

4. Subiendo a producción

Para subir nuestra aplicación a internet vamos a usar la plataforma heroku, ya que es un servicio muy sencillo de usar.

Primero nos creamos una cuenta en heroku y después creamos una aplicación dentro de heroku.

Con la aplicación lista nos dirigimos a nuestro proyecto e inicializamos un repositorio de git.

```
git init
git commit -m "first commit"
```

Ahora lo que hacemos es iniciar sesión con el heroku CLI, para ello ejecutamos el comando `heroku login`. Nos llevará a una ventana del navegador para iniciar sesión.

Después agregamos el repositorio remoto de heroku a nuestro repositorio local para posteriormente subirlo ahí. Ejecutamos:

```
heroku git:remote -a nombre-aplicacion-heroku
git push -a heroku master
```

Con esto ya tendremos nuestro proyecto subido a heroku. ahora simplemente abrimos la url de nuestra aplicación en heroku o escribimos el comando `heroku open` para abrir en una pestaña del navegador.

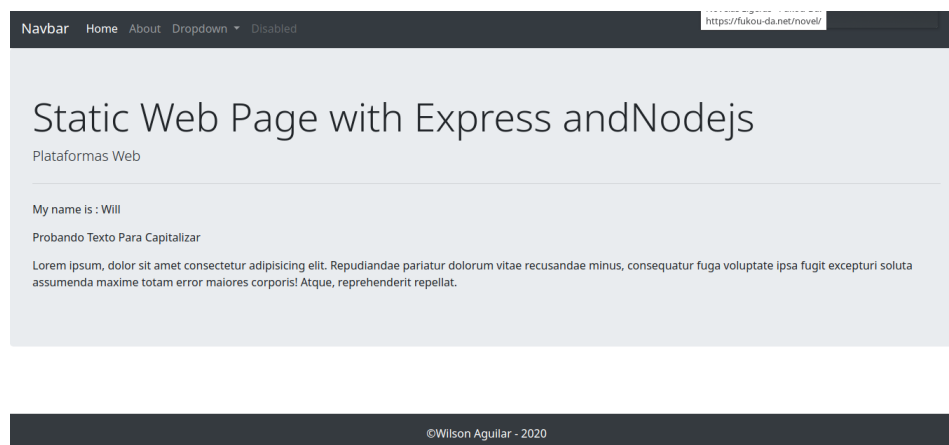


Figura 10: Sitio web desplegado a heroku.

5. Repositorio en Github

El repositorio se encuentra en el siguiente enlace:

<https://github.com/WilsonAG/plataformas-web/tree/master/node/06-webserver>