

**DESARROLLO DE APLICACIONES WEB AVANZADO**

**LABORATORIO N° 11**

**Llamadas asíncronas en React**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Alumno(s):*** | ***CONDORI CONDORI, JOHNN BILL*** | | | | | | ***Nota*** |  | |
| ***Grupo:*** | ***B*** | | ***Ciclo:V*** | | | | | | |
| ***Criterio de Evaluación*** | | ***Excelente (4pts)*** | | ***Bueno (3pts)*** | ***Requiere mejora (2pts)*** | ***No acept. (0pts)*** | | | ***Puntaje Logrado*** |
| Entiende el ciclo de vida de los componentes | |  | |  |  |  | | |  |
| Utiliza React-router | |  | |  |  |  | | |  |
| Desarrolla aplicaciones web con ReactJS | |  | |  |  |  | | |  |
| Realiza con éxito lo propuesto en la tarea. | |  | |  |  |  | | |  |
| Es puntual y redacta el informe adecuadamente | |  | |  |  |  | | |  |

**Laboratorio 11:**

**Llamadas asíncronas**

**Objetivos:**

Al finalizar el laboratorio el estudiante será capaz de:

* Entender el funcionamiento del ciclo de vida
* Desarrollar aplicaciones web enfocadas a componentes
* Implementación correcta de Routing

**Seguridad:**

* Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
* No ingresar con líquidos, ni comida al aula de Laboratorio.
* Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

**Equipos y Materiales:**

* Una computadora con:
* Windows 7 o superior
* VMware Workstation 10+ o VMware Player 7+
* Conexión a la red del laboratorio
* Máquinas virtuales:
* Windows 7 Pro 64bits Español - Plantilla
* Instalador de node.js

**Procedimiento:**

**Lab Setup**

1. **Configuración inicial**
   1. El docente proveerá el archivo lab11.zip el cuál deberá descomprimir para su utilización en el presente laboratorio. Dicho laboratorio contiene un proyecto de React basado en el laboratorio 10 del presente curso, con ciertas mejoras en el CSS, el cual se encuentra comentado para que pueda usted analizarlo.
   2. Esto dejará listo nuestro entorno de trabajo, por lo que nos situaremos en la carpeta generada y posteriormente iniciaremos el proyecto con **npm start**







1. **Componentes de Jerarquía Alta (HOC o High Order Components)**
   1. Es común encontrar una carpeta llamada hoc en los repositorios de código de proyectos de React. hoc se refiere a High Order Components y suelen ser componentes que engloban a nuestros componentes básicos o ayudan a reciclar funcionalidad básica que podemos repetir una y otra vez.
   2. Crearemos la carpeta hoc dentro de la carpeta src. Dentro de ella crearemos la carpeta Auxiliary con el archivo Auxiliary.js con el siguiente contenido.



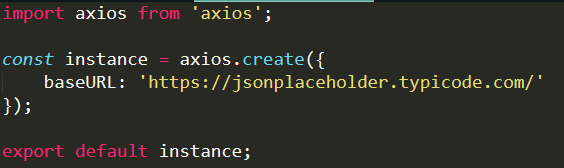
* 1. Ahora modificaremos el archivo Post.js. El archivo creado es un componente auxiliar que sirve como truco para no englobar nuestro componente Post en un div (como está actualmente). Recuerda que todo componente en React debe tener una etiqueta de inicio y una de fin, no pueden ser tags consecutivos sin un padre. Con este componente Auxiliary podemos evadir dicho inconveniente.



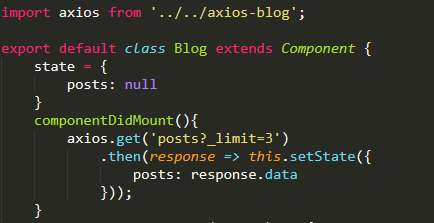
1. **Llamadas asíncronas en React**
   1. Utilizaremos axios para la comunicación asíncrona de nuestra aplicación. Para eso debemos detener el proyecto e instalarlo



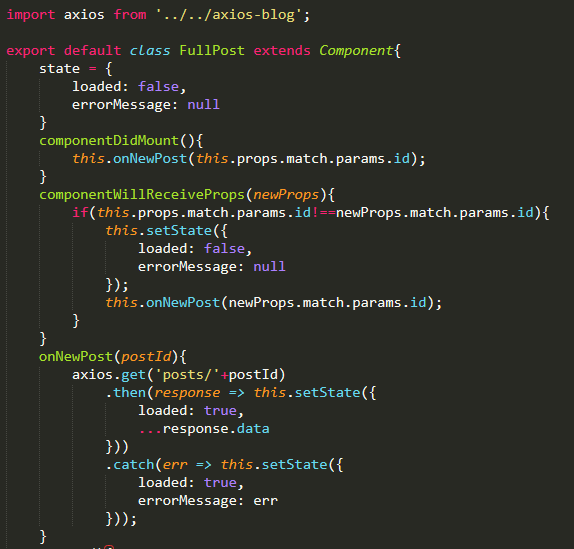
* 1. Crearemos el archivo axios-blog.js dentro de nuestra carpeta src



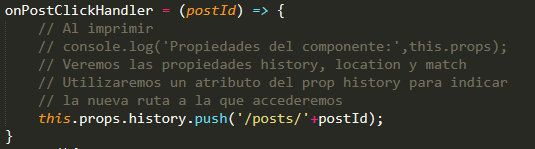
* 1. Modificaremos nuestro archivo Blog.js y veremos el resultado en el navegador.



* 1. Modificaremos también el archivo FullPost.js para poder obtener el contenido del post al que accedemos mediante la URL



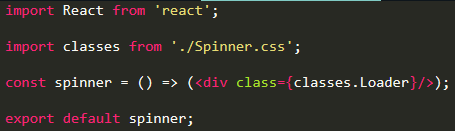
* 1. En esta última actualización vemos un cambio adicional importante. Hacemos uso de componentWillReceiveProps para poder verificar que cambia nuestro id de post. Al estar utilizando react-router-dom, nuestros componentes directamente afectados tendrán tres props inyectados: match, history y location. Utilizamos match para poder obtener los parámetros inyectados en la URL. Ahora crearemos ese efecto de navegación a través de Posts.
  2. En Blog.js cambiaremos el contenido de la función onPostClickHandler, que se disparará cuando el usuario le haga click a algunos de nuestros posts.



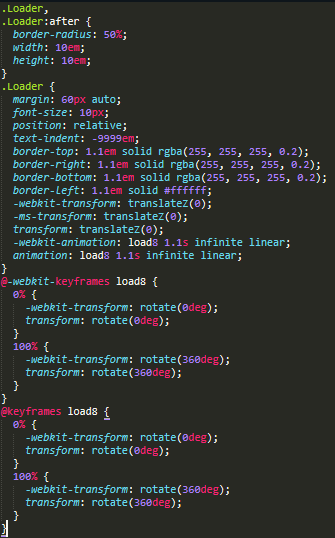
* 1. Hemos visto que la forma de obtener data a través de axios es sencilla gracias a la función get. Asi mismo, contamos con una función post, la que utilizaremos para insertar data. Modificaremos el archivo NewPost.js



1. **Indicadores de carga (loaders)**
   1. Nuestra aplicación funciona realizando llamadas asíncronas al servidor, pero no de una manera ideal, ya que al realizar la carga de data del servidor solamente vemos el mensaje Cargando y no es una manera elegante de presentarlo. Crearemos dentro de la carpeta UI la carpeta Spinner y a su vez, el archivo Spinner.js

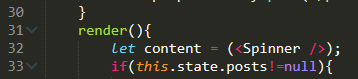


* 1. A su vez, crearemos el archivo Spinner.css con el siguiente contenido. Antes de empezar a escribirlo, lea el punto 4.3



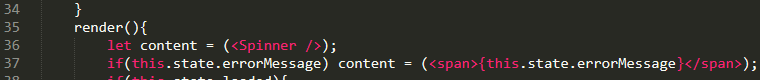
* 1. El spinner (o efecto de cargador) ha sido extraído de la página Single Element CSS Spinners [(https://projects.lukehaas.me/css-loaders/)](https://projects.lukehaas.me/css-loaders/) la cual tiene una galería amplia de efectos de Loaders. Yo escogí la que más me agradó, pero Usted puede escoger la que desee del conjunto mostrado en la página, o hace su propia búsqueda en Google con **css loaders**
  2. En caso seleccione algún ejemplo de la página citada, tendremos que hacer un pequeño cambio, que es reemplazar el nombre de clase que nos toque por **Loader** (por ejemplo, si al escoger su ejemplo en el código css encuentra **.spinner-loading**, esto se deberá reemplazar por **.Loader**) Recuerde que esto se realiza solamente para llevar coherencia con el código antes escrito, no es un requerimiento de React.
  3. Incluiremos el componente Spinner creado en nuestro archivo Blog.js para ver su funcionamiento.



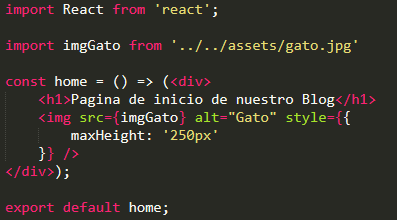


* 1. Al ver que funciona perfectamente, lo añadiremos a FullPost.js





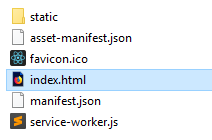
1. **Importación de imágenes en React**
   1. La importación de imágenes en React es realmente simple. Primero procederemos a crear una carpeta assets dentro de src y colocar dentro la imagen **gato.jpg**, provista por el docente. Nuevamente, esto no es algo obligatorio de React, sino que es una convención utilizada por la mayoría de programadores, además que es más fácil centralizar una sola carpeta de archivos extras.
   2. Modificaremos nuestro archivo Home.js para ver la inclusión de la imagen.



1. **Creación de archivos de producción**
   1. Ejecutaremos el siguiente comando para crear nuestros archivos de producción.



* 1. Al revisar nuestro proyecto, veremos que ahora contamos con la carpeta build, la cual contiene una versión optimizada de nuestro proyecto, lista para ser incluida en cualquier servidor web, sin importar su lenguaje o versión.



* 1. Para simular que estamos subiendo estos archivos estáticos a un servidor web real, instalaremos el paquete de npm **serve**, el cual nos permitirá emular dicho servidor



* 1. Al ingresar a <http://localhost:5000> veremos nuestro proyecto de React funcionando sin necesidad del servidor original de desarrollo. Funciona independiente y podemos verificar mediante las herramientas de navegador que ahora contiene solamente un archivo Javascript y su equivalente Css, ambos optimizados sin mostrar nada de nuestro código original.

1. **Finalizar la sesión**
   1. Apagar el equipo virtual
   2. Apagar el equipo

**Conclusiones:**

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.

|  |
| --- |
|  |