

**DESARROLLO DE APLICACIONES WEB AVANZADO**

**LABORATORIO N° 10**

**Ciclo de vida y Routing**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Alumno(s):*** |  | | | | | | ***Nota*** |  | |
| ***Grupo:*** |  | | ***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 10:**

**Ciclo de Vida y Routing**

**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

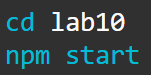
**Procedimiento:**

**Lab Setup**

1. **Instalación y configuración de ReactJS**
   1. Ejecuta el siguiente comando para preparar la instancia de ReactJS del presente laboratorio



* 1. 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. Fases del ciclo de vida

El ciclo de vida se puede dividir en 3 fases, el montado, actualización y desmontado del componente. Estas fases a su vez se dividen en varios métodos que puede tener el componente.

Tip: Cada método tiene un prefijo will o did dependiendo de si ocurren antes o después de cierta acción.

**Montado**

La primera fase ocurre solo una vez por componente cuando este se crea y monta en la UI. Esta fase se divide en 4 funciones.

constructor(props)

Este método se ejecuta cuando se instancia un componente. Nos permite definir el estado inicial del componente, hacer bind de métodos y definir propiedades internas en las que podemos guardar muchos datos diferentes, por ejemplo la instancia de una clase (un parser, un validador, etc.).

componentWillMount()

Este método se ejecuta cuando el componente se está por renderizar. En este punto es posible modificar el estado del componente sin causar una actualización (y por lo tanto no renderizar dos veces el componente). Es importante sin embargo evitar causar cualquier clase de efecto secundario (petición HTTP por ejemplo) ya que este método se ejecuta en el servidor y hacer esto puede causar problemas de memoria.

render()

En este momento de la fase de montado se van a tomar las propiedades, el estado y el contexto y se va a generar la UI inicial de este componente. Esta función debe ser pura (no puede tener efectos secundarios) y no debe modificar nunca el estado del componente.

Actualizar el estado en este punto puede causar un ciclo infinito de renderizados, ya que cada cambio al estado genera que el componente se renderice de nuevo (y vuelva a cambiar el estado).

componentDidMount()

Este último método de la fase de montado se ejecuta una vez el componente se renderizó en el navegador (este no se ejecuta al renderizar en el servidor) y nos permite interactuar con el DOM o las otras APIs del navegador (geolocation, navigator, notificaciones, etc.).

También es el mejor lugar para realizar peticiones HTTP o suscribirse a diferentes fuentes de datos (un Store o un WebSocket) y al recibir una respuesta, actualizar el estado. Cambiar el estado en este método causa que se vuelva a renderizar el componente.

**Actualización**

Esa fase puede ocurrir múltiples veces (o incluso ninguna), sucede cuando algún dato del componente (ya sea una propiedad, un estado o el contexto) se modifica y por lo tanto requiere que la UI se vuelva a generar para representar ese cambio de datos.

componentWillReceiveProps(nextProps)

Este método se ejecuta inmediatamente después que el componente reciba nuevas propiedades. En este punto es posible actualizar el estado para que refleje el cambio de propiedades, ya sea reiniciando su valor inicial o cambiándolo por uno nuevo.

Hay que tener en cuenta que React puede llegar a ejecutar este método incluso si las propiedades no cambiaron. Por eso es importante validar que las nuevas propiedades (nextProps) sean diferentes de las anteriores (this.props).

shouldComponentUpdate(nextProps, nextState)

Este método (el cual debe ser puro) se ejecuta antes de empezar a actualizar un componente, cuando llegan las nuevas propiedades (nextProps) y el nuevo estado (nextState).

Acá es posible validar que estos datos sean diferentes de los anteriores (this.props y this.state) y devolver true o false dependiendo de si queremos volver a renderizar o no el componente.

Los componentes creados al extender React.PureComponent implementan esta validación sin necesidad de que hagamos nada y de una forma que no afecte al rendimiento. El resto de componentes devuelven siempre true por defecto.

Hay que tener cuidado con este método ya que si nuestro componente tiene otros componentes con estado como hijos, devolver false acá puede impedir que estos sub-componentes no se actualicen al detectar un cambio.

componentWillUpdate(nextProps, nextState)

Una vez el método anterior devolvió true se ejecuta este método, acá es posible realizar cualquier tipo de preparación antes de que se actualice la UI.

Es importante tener encuenta que acá no se puede ejecutar this.setState para actualizar el estado. Si queremos actualizar el estado con base a un cambio de propiedades debemos hacerlo en componentWillReceiveProps.

render()

Al igual que en el montado acá se va a generar la UI, esta vez con los datos que hayan cambiado. Como antes este método debe ser puro.

componentDidUpdate(prevProps, prevState)

Esta última parte de la actualización de un componente ocurre justo después de que se renderiza en el DOM nuestro componente. Al igual que con componentDidMount() acá es posible interactuar con el DOM y cualquier API de los navegadores.

Aunque acá podríamos realizar una petición HTTP y actualizar el estado hay que tener cuidado, ya que de hacerlo podríamos causar un bucle infinito de actualizaciones y peticiones HTTP.

**Desmontado**

Esta última fase consiste en un solo método que se ejecuta antes de que un componente se elimine (desmonte) de la UI de nuestra aplicación.

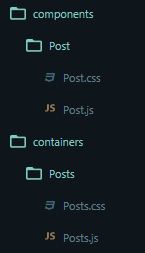
componentWillUnmount()

Este único método de la fase de desmontado nos permite realizar cualquier tipo de limpieza antes de remover el componente.

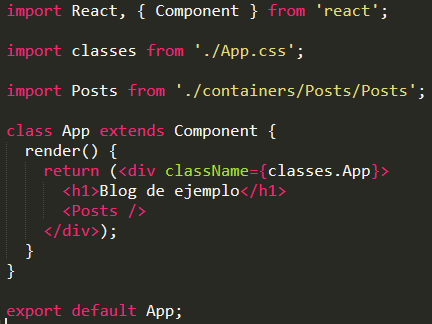
Acá es posible dejar de escuchar eventos de window, document o el DOM, desuscribirse de un WebSocket o Store o cancelar peticiones HTTP que hayan quedado pendientes.

Es importante hacer esta limpieza ya que si alguna petición pendiente se completa luego del desmontado, va a tratar de actualizar el estado y va a dar un error (y hasta un posible problema de memoria) ya que el componente no existe más.

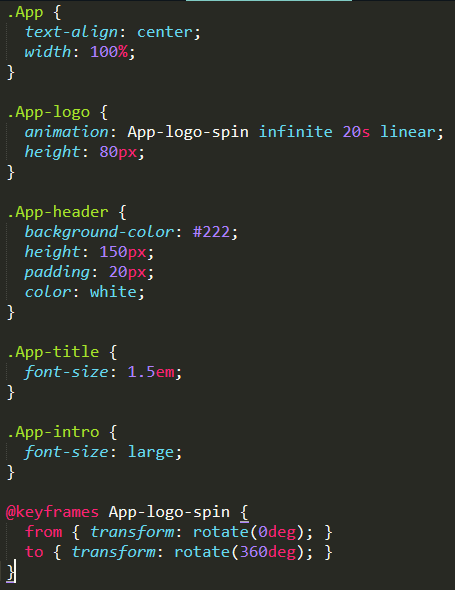
1. **Código de ejemplo actualización asíncrona**
   1. Crear la siguiente estructura de archivos y luego completar su contenido como se muestra en los siguientes archivos.



* 1. App.js



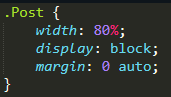
* 1. App.css



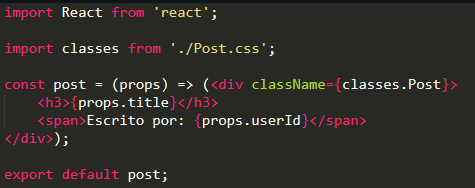
* 1. Posts.js



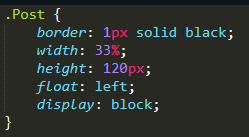
* 1. Posts.css



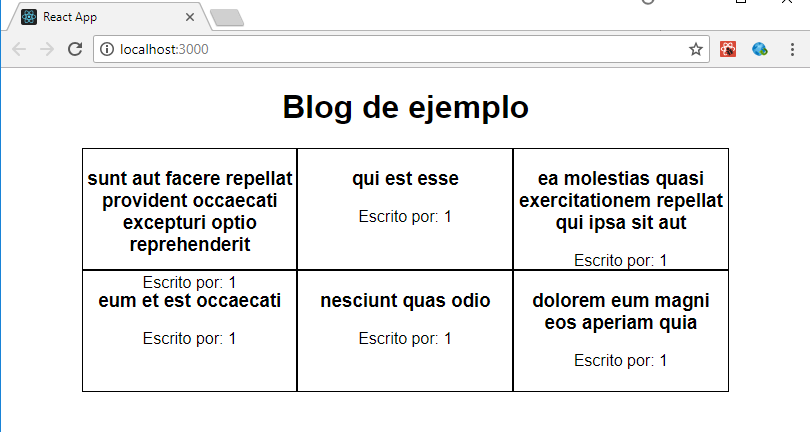
* 1. Post.js



* 1. Post.css



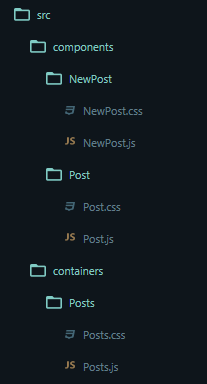
* 1. Adjunte una captura del resultado del código.



1. Routing:
   1. Instale los siguientes paquetes:

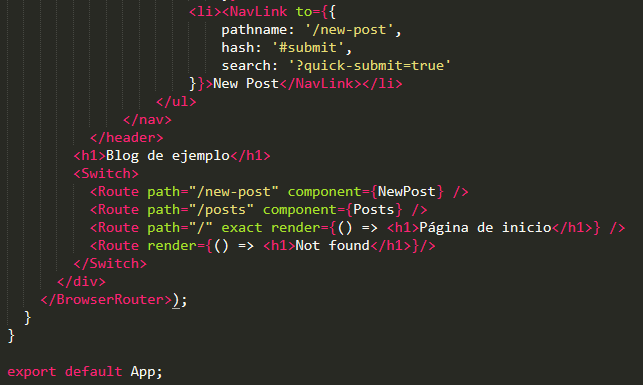


* 1. La estructura de carpetas será la siguiente:



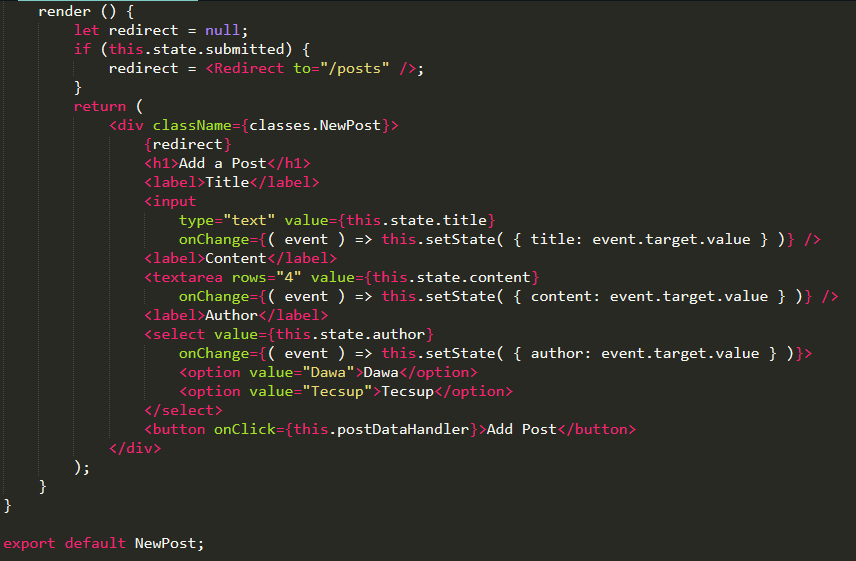
* 1. Ahora modificaremos los siguientes archivos:
  2. App.js



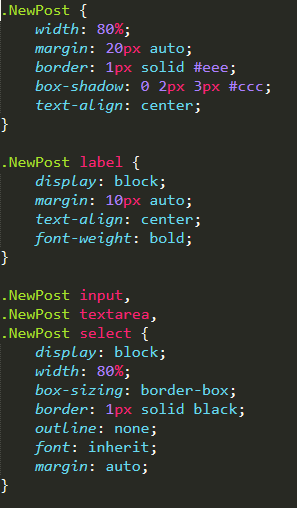


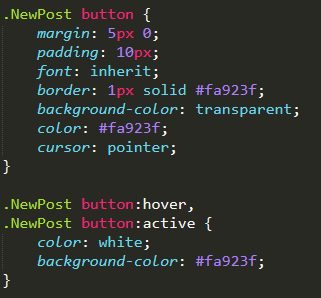
* 1. NewPost.js





* 1. NewPost.css





* 1. Adjunte capturas del comportamiento de la aplicación después de haber aplicado el enrutamiento. Fíjese en la cero recarga del navegador a pesar de navegar entre links y retroceder o avanzar con los controles del navegador.

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

**Tarea:**

Realizar el enrutamiento de su proyecto grupal:

* Crear las rutas y presentar con los componentes de grupo ya desarrollados en anteriores sesiones.

**Conclusiones:**

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

|  |
| --- |
|  |