DigitalHouse > Coding School

Programación Web Full Stack

El ciclo de vida de los componentes

Práctica integradora

Objetivo

Hasta ahora hemos visto cómo renderizar componentes a partir de un estado inicial y de una serie de datos originales, pero, como bien sabemos, las aplicaciones necesitan responder ante **eventos** producidos por el usuario —clics, escritura en cajas de texto, etc—.

Es por ello que es conveniente en ciertos casos valernos de los **componentes con estado**. Estos permiten mantener datos propios a lo largo del tiempo e implementar comportamientos en sus diferentes métodos del ciclo de vida.

El **ciclo de vida** no es más que una serie de estados por los cuales pasa todo componente a lo largo de su existencia. Básicamente se pueden clasificar en tres etapas: de montaje o inicialización, actualización y destrucción. Dichas etapas tienen correspondencia en diversos métodos, que nosotros podemos implementar para realizar acciones concretas cuando estos van sucediendo.

Con React, podemos usar cualquier biblioteca AJAX. Algunas de las más populares son **Axios**, **jQuery AJAX**, y **window.fetch** —la cual es soportada de manera nativa en la mayoría de los navegadores modernos—, y de esa forma lograr la **integración con APIs**.

En esta ejercitación tendremos la responsabilidad de poner en práctica todos los elementos anteriormente mencionados. Es decir, ampliaremos la funcionalidad de nuestro

tablero administrativo, creando componentes con estado, contemplando el ciclo de vida de los mismos, integrándonos con nuestras APIs creadas en nuestro proyecto de Node.js y Express, para finalmente incorporar eventos con el fin de lograr la interacción del usuario con el componente.

¡Éxitos! 😎👍✨

Requisitos:

Utilizaremos de base los siguientes proyectos:

- **Proyecto creado con React**: recordemos instalar todas las dependencias del proyecto, ejecutando la instrucción **npm install**. 😉

  Para activar el servidor debemos ejecutar el comando **npm start** y finalmente en el navegador ejecutar: http://localhost:3000/.

- **Proyecto creado con Express**: recordemos instalar todas las dependencias del proyecto, ejecutando la instrucción **npm install**. 😉 Además, aprovecharemos la base de datos **movies_db** (no olvidemos activar el servicio de MySQL en nuestro equipo). De esta manera, todo funcionará correctamente.

  Para activar el servidor debemos ejecutar el comando **npm test** y finalmente en el navegador ejecutar: http://localhost:3001/.

Micro desafío - Paso 1:

- Hacer que el componente **/components/GenresInDb.js** sea ahora un componente stateful para poder conectarse con nuestro back-end a través de una API, la cual ya está armada y su endpoint es: **/api/genres**. Esta debe exponer de forma dinámica los géneros de las películas:
    - No nos olvidemos de importar de **React** el **Component**:

```
import React, {Component} from 'react';
```

    - Además, debemos importar el componente responsable de renderizar de forma dinámica los datos:

```
import Genre from './Genre';
```

    - Para crear un componente con estado, recordemos la sintaxis inicial:

```
class GenresInDb extends Component{...}
```

    - Una de las primeras cosas que debemos hacer es crear el método **constructor**:

```
constructor(){
        super()
        this.state ={
            genresList : []
        }
    }
```

    - Ahora debemos establecer la conexión con nuestro back-end con **React**, podemos usar cualquier biblioteca AJAX, recomendamos utilizar

windows.fetch. Esto lo debemos programar en el momento en que el componente se monte, valiéndonos del método **componentDidMount()**:

```
componentDidMount(){

        fetch('/api/genres')

        .then(respuesta =>{

            return respuesta.json()

        })

        .then(genres =>{

            //console.log(genres)

            this.setState({genresList: genres.data})

        })

        .catch(error => console.log(error))

    }
```

- Ahora debemos proceder a renderizar lo que se le mostrará al usuario de forma dinámica:

```
render(){

  return (

    <React.Fragment>

      {/*<!-- Aquí va todo el contenido estático que tenemos en
Genres in DB -->*/}

      {/*<!-- Aquí es donde vamos a enviar los datos obtenidos
al componente Genre para que renderice de forma dinámica
-->*/}

      <div className="card-body fondoCaja">

          <div  className="row">
```

```
          {this.state.genresList.map((genre,index)=>{
           return  <Genre  {...genre}  key={index} />
           })}
          </div>
       </div>
    </React.Fragment>
    )
    }
```

- Ya solo resta crear el componente **/component/Genre.js**. Podemos crearlo como un componente sin estado el cual recibirá las **props** enviadas desde el componente **/component/GenresInDb.js** y renderizar los datos:

```
import React from 'react';
function Genre(props){
    return (
        <React.Fragment>
            <div className="col-lg-6 mb-4">
             <div className="card bg-dark text-white shadow ">
                    <div className="card-body">
                        {props.name}
                    </div>
             </div>
            </div>
        </React.Fragment>
    )
```

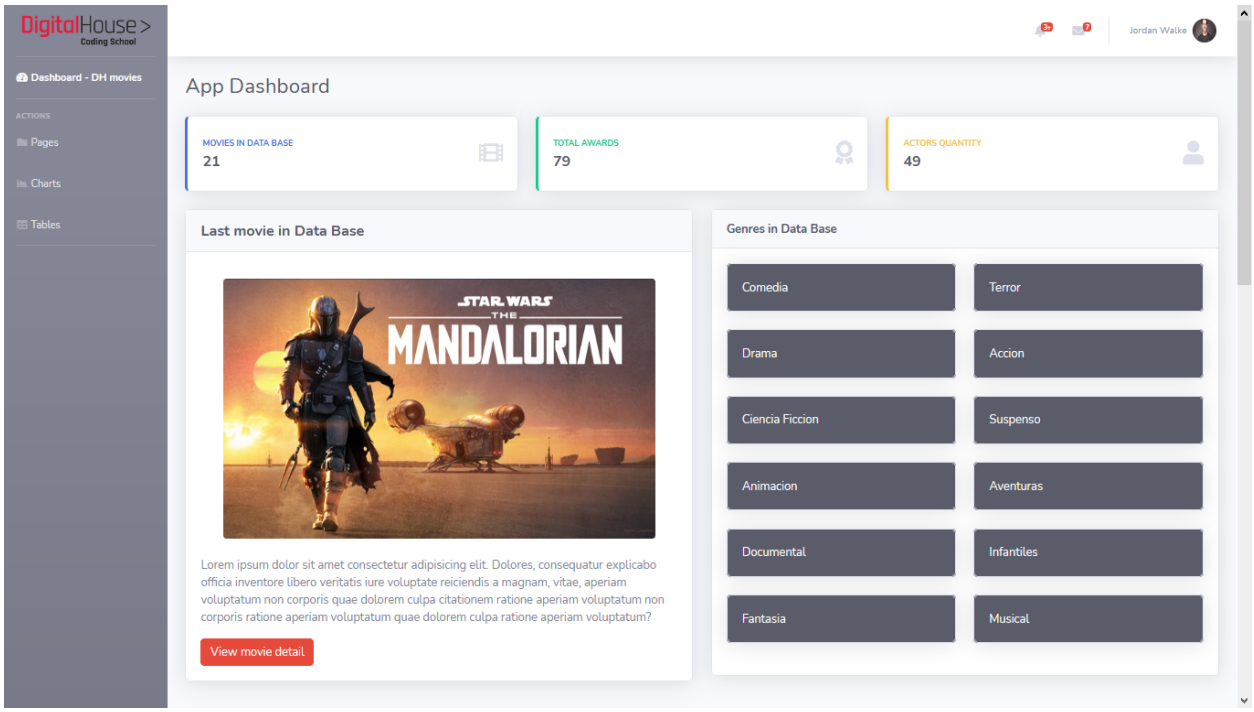
```
}

export default Genre;
```

- Para lograr que todo funcione correctamente, debemos escribir una línea de código más, para establecer una comunicación entre nuestra aplicación en React y nuestra aplicación del lado del back-end. Existen varias formas de efectuar dicha comunicación, considerando elementos de seguridad por medio de *token*, sin embargo, nosotros vamos a establecer dicha comunicación de una forma muy sencilla por medio del uso del proxy, por lo cual solo debemos hacer una pequeña actualización en nuestro archivo **package.json**. Con esto lograremos que nuestra aplicación de React se comunique con nuestra aplicación desarrollada con Node.js y Express:

```
"proxy": "http://localhost:3001"
```

Si logramos terminar todo lo indicado, al ejecutar:

Deberíamos ver lo siguiente:



Micro desafío - Paso 2:

- Ahora, debemos ingresar al componente **/components/GenresInDb.js** y crear un método que maneje el evento **onMouseOver** de forma tal que al pasar el mouse sobre el elemento **<h6>**:

```
<h6 className="m-0 font-weight-bold text-gray-800">Genres in Data Base</h6>
```

El card-body (el cual encierra a los géneros de las películas) debe cambiar el fondo, aplicando la siguiente clase de bootstrap: **bg-secondary**.

```
<div className="card-body fondoCaja">

   <div  className="row">

   </div>

</div>
```

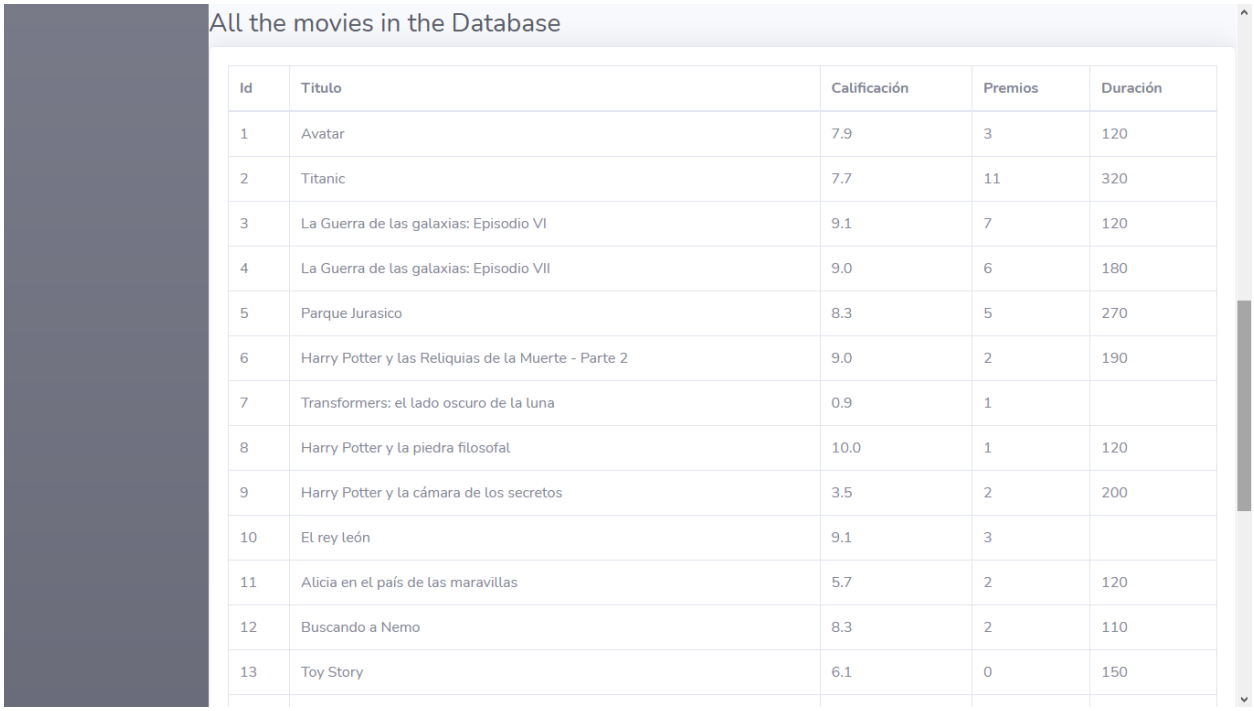
Micro desafío - Paso 3:

- Hacer que el componente **/components/Movie.js** sea un componente stateful para poder conectarse con nuestro back-end a través de una API, la cual ya está armada y su endpoint es: **/api/movies**. Esta debe exponer de forma dinámica la siguiente información de la películas:

```
<thead>

    <tr>

      <th>Id</th>

      <th>Título</th>
```

```
      <th>Calificación</th>
      <th>Premios</th>
      <th>Duración</th>
    </tr>
</thead>
```

- Hacer que el componente **/component/MoviesList.js**, el cual podemos crear como un componente sin estado, reciba las **props** enviadas desde el componente **/component/Movies.js** y se encargue de renderizar los datos recibidos. Al final, debemos lograr algo así:



Conclusión

Con esta práctica pudimos comprobar la versatilidad que nos ofrece **React** por medio de la creación de los componentes con estado, aplicando el ciclo de vida de los mismos, así como la integración con nuestras APIs y el comportamiento que deseamos que ocurra en nuestras páginas haciendo uso de los eventos.

¡Hasta la próxima!