Ejercicios del Tema 2

Crea un nuevo proyecto de React mediante la consola, como ya has visto en el tema anterior. Vamos a proceder a realizar una aplicación que cambie con el tiempo, similar al ejemplo que daba la hora actual.

Ejercicio 1

En este primer ejercicio vamos a desarrollar un componente que nos permita visualizar la fecha y la hora dentro de nuestra aplicación.

Para ello, lo primero que vamos a hacer es el fichero representando nuestro componente. Para hacerlo lo más completo posible, vamos a implementar el componente a partir de una clase. La estructura básica podría ser:

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

class FechaHora extends Component{

  constructor(props){
    super(props);
  }

  render(){
    return (
      <div>
           Componente FechaHora
      </div>
    );
  }
}

export default FechaHora;
```

Debemos tener en cuenta:

- Es importante que la clase generada para nuestro componente personalizado extienda de la clase **Component** de ReactJS para heredar todas sus funcionalidades.

- Debemos implementar el constructor para inicializar todas las propiedades de la clase con la que estamos trabajando.
- De igual manera, debemos implementar el método **render,** mediante el cual podemos definir qué datos vamos a mostrar cuando utilicemos este componente.

Para poder utilizar el componente, dentro de nuestro fichero **index.js** debemos importarlo y agregarlo al renderizado general de nuestra aplicación:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';

import FechaHora from './FechaHora';

ReactDOM.render(
  <FechaHora />,
  document.getElementById('root')
);
```

El resultado de nuestra aplicación en este momento debería ser "**Componente FechaHora**"

Vamos a tratar de mejorar el componente separando las funcionalidades de la fecha y la hora en métodos para así poder actuar independientemente.

El primer paso sería completar el constructor de la clase **FechaHora** para así poder inicializar los datos de las propiedades con las que vamos a trabajar, en este caso, la fecha actual.

```
constructor(props){
   super(props);
   this.state = {
     date : new Date()
   };
}
```

De esta manera conseguimos que, cuando mostramos nuestro componente, en la propiedad **date** tengamos almacenadas la fecha y la hora actual.

El siguiente paso será la definición de los métodos para mostrar tanto la fecha como la hora actual. Podemos especificar una estructura para cada uno de los elementos:

```
getFecha(){
   return (
     <div className="fecha">
       <p>Fecha: {this.state.date.toLocaleDateString()}</p>
     </div>
   );
 };

 getHora(){
   return (
     <div className="hora">
       <p>Hora: {this.state.date.toLocaleTimeString()}</p>
     </div>
   );
 };
```

A través de la propiedad **className** estamos definiendo qué estilos vamos a poder asignar a cada elemento de nuestro componente.

Para ello, podemos crear un fichero **FechaHora.css** y especificar algunos estilos

```
div{
  width: 150px;
  height: 30px;
  text-align: center;
}

p{
  margin: 0px;
}

.fecha{
  background-color: #D4A190;
}

.hora{
  background-color: #90C3D4;
}
```


No olvides importar dentro del fichero **FechaHora.js**, la hoja de estilos que acabamos de crear para que se puedan aplicar sobre el componente que estamos creando.

Al igual que hemos hecho en el tema, necesitamos actualizar los datos de la fecha en los segmentos de tiempo que nos interese. En este caso, como estamos controlando un reloj, actualizaremos los datos cada segundo.

Este tipo de actualizaciones debemos hacerlas en el momento de cargar los datos de nuestro componente y asegurarnos de terminarlos cuando desaparezca el componente.

Para ello podemos implementar los métodos **componentDidMount** y **componenteWillUnmount**.

```
componentDidMount() {
   this.timerID = setInterval(
     () => this.tick(),
     1000
   );
 }

 componentWillUnmount() {
   clearInterval(this.timerID);
 }

 tick() {
   this.setState({
     date: new Date()
   });
 }
```

Necesitamos modificar el método **render()** de nuestro componente para hacer las llamadas sobre los métodos que acabamos de generar

```
render(){
  return (
    <div>
      {this.getFecha()}
      {this.getHora()}
    </div>
  );
}
```

Para ampliar nuestro componente, vamos a pasarle por parámetro el intervalo de tiempo que vamos a usar para actualizar nuestra hora.

Para ello, vamos a definir una nueva propiedad dentro del objeto state, a partir de la propiedad que usemos en el momento de definir el componente. Podemos inicializarlo en el constructor.

```
constructor(props){
   super(props);
   this.state = {
     date : new Date(),
     interval: parseInt(this.props.interval)
   };
 }
```

Posteriormente podemos utilizar este intervalo a la hora de definir el timer

```
componentDidMount() {
   this.timerID = setInterval(
     () => this.tick(),
     this.state.interval
   );
 }
```

Esto nos permite la posibilidad de utilizar varios componentes de tipo **FechaHora** con comportamientos diferentes

```
ReactDOM.render(
  <div>
    <FechaHora interval="5000" />
    <FechaHora interval="10000" />
    <FechaHora interval="1000" />
  </div>
    ,
  document.getElementById('root')
);
```


AMPLIACIÓN

Agrega una nueva propiedad dentro del componente que te permita mostrar u ocultar un mensaje encima de la fecha.

Para poder mostrar u ocultar el mensaje, puedes hacerlo a través la propiedad **display** de CSS. (los valores pueden ser ‘none’ y ‘block’)

