# Ejercicios del Tema 3

**Crea un nuevo proyecto de React mediante la consola, como ya has visto en el tema anterior.**

EJERCICIO 1

En este ejercicio vamos a *simular* una característica tan utilizada en otra serie de frameworks como Angular, el **doble binding**.

Vamos a crear un campo de texto para poder introducir datos y queremos que esos datos se vean reflejados en *tiempo real* en otros espacios de nuestra aplicación.

Para ello vamos a capturar los eventos de teclado sobre el campo de texto y aplicar diferentes operaciones sobre la propiedad **state** de nuestro componente.

El primer paso será generar el nuevo componente con el campo de texto y el evento **KeyPress:**

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

class ComponenteEventos extends Component{

  constructor(props){
    super(props);
  }

  render(){
    return (
      <div>
        <input type="text" onKeyPress={this.manejaPulsacionTeclado}
      </div>
    );
  }

}

export default ComponenteEventos;
```

 

El siguiente paso será la implementación del método **manejaPulsaciónTeclado** en el que vamos a recoger la tecla pulsada y la agregaremos sobre una propiedad contenida dentro de **state** de tipo texto.

```
 manejaPulsacionTeclado(event){
   console.log(event);
   let texto = this.state.texto + event.key;
   this.setState({ texto: texto});
}
```

Recuerda que para poder trabajar con this (y que haga referencia a nuestro componente) dentro del método que captura el evento tenemos que enlazarlo dentro del constructor.

Aparte, aprovecharemos el constructor para inicializar el valor de texto.

```
 constructor(props){
   super(props);
   this.state = {
     texto: ""
   };
   this.manejaPulsacionTeclado = this.manejaPulsacionTeclado.bind(this);
 }
```

De esta manera conseguimos que nos identifique el texto que estamos introduciendo por teclado y podríamos mostrarlo en cualquier punto de la vista con la que estamos trabajando:

```
render(){
   return (
     <div>
       <input type="text" onKeyPress={this.manejaPulsacionTeclado} />
       {this.state.texto}
     </div>
   );
 }
```

 

Podemos comprobar que el evento con el que estamos trabajando captura correctamente la mayoría de las teclas. Podríamos encontrar algún problema a la hora de borrar el texto ya que no estamos capturando la pulsación de la tecla de borrado.

Para poder interactuar con la tecla de borrado, debemos capturar el evento **KeyDown** y filtrar para el caso específico del borrado:

```
 render(){
    return (
      <div>
        <input type="text" onKeyPress={this.manejaPulsacionTeclado}
onKeyDown={this.manejaKeyDown} />
        {this.state.texto}
      </div>
    );
  }


 manejaKeyDown(event){
    console.log(event.keyCode);
    if(event.keyCode === 8){
      var texto = this.state.texto;
      var str = texto.substring(0, texto.length - 1);
      this.setState({texto: str});
    }
  }
```

No olvides enlazar el nuevo método que hemos creado para que podamos trabajar con la instancia de **this** dentro del mismo.

Una vez implementado el evento, ya podemos interactuar con el campo de texto y ver reflejados los cambios dentro de nuestra vista.

**Ejercicio 2**

En este ejercicio vamos a trabajar con los eventos de ratón.

Nuestro objetivo será detectar dichos eventos sobre una parte de nuestra vista y actuar en consecuencia haciendo modificaciones sobre el color de fondo.

Partimos del siguiente componente:

 

```
import React, { Component } from 'react';
import './MouseColor.css';

class MouseColor extends Component{

  constructor(props){
    super(props);
    this.state = {
      color: {
        red: 123,
        green: 234,
        blue: 221
      }
    };
  }

  render(){

    return (
      <div className="contenedor" style={{backgroundColor:
'rgb(200,'+this.state.color.green+','+this.state.color.blue+')'}}>

      </div>
    );
  }

}

export default MouseColor;
```

Como se puede observar, hemos inicializado una seri de colores dentro de state para posteriormente usarlos en la definición de los estilos de nuestro componente.

Habría que generar también la hoja de estilos CSS asociada a este componente.

Nuestro siguiente paso será generar el método que se encargue de modificar la gama de colores en función de las coordenadas donde se encuentre el ratón.

 

```
manejoMovimientoRaton(event){
   this.setState({
     color: {
       red: event.clientX,
       green: event.clientY,
       blue: event.clientX
     }
   });
 }
```

Una vez creado el método, no podemos olvidar capturarlo sobre el **div** principal de nuestro componente, haciendo uso del evento **onMouseMove**

Aparte, como en el ejercicio anterior, tenemos que enlazar este método con this, para poder usar state.

Una vez tengamos todo creado, si pasamos el ratón por encima del componente, deberíamos ver cómo cambia el color de fondo (no olvides darle un tamaño al div del componente para así poder visualizarlo).

**AMPLIACIÓN**

- Detecta cuando el ratón entra dentro del componente y asigna un color de fondo aleatorio.
- Haz lo mismo cuando el ratón salga del componente.
- Detecta el evento de doble click sobre el componente y bloquea los cambios de color hasta que no se haga de nuevo doble click

 