# **Proyecto Hoteles: Filtros (Parte 2)**

Ahora que tenemos nuestros filtros en la interfaz, debemos comenzar a definir su comportamiento y de esta forma permitir que los mismos puedan modificar los datos de la aplica

Para lograr esto lo primero que debemos hacer es lograr que los datos del objeto que contiene los filtros sean modificables por React durante la ejecución. Esto se logra creando un estado (state) en un componente de la aplicación.

Hasta el momento hemos creado tres componentes principales: Hero, Filters y App que contiene a ambos. El estado de los filtros debe ser accesible tanto por el componente Hero como por el componente Filters. Por esta razón el estado debe estar en un componente padre que pueda brindarle información y comportamiento a ambos. En este caso, el componente App.

## **Componente**App

En este punto deberíamos tener el componente App creado con una sintaxis similar a la siguiente.

function App() {

const filters = {

dateFrom: today, // Proviene del archivo data.js

dateTo: new Date(today.valueOf() + 86400000),

country: undefined,

price: undefined,

rooms: undefined

}

return (

<div>

<Hero filters={ filters } />

<Filters filters={ filters } />

</div>

)

}

Esta metodología nos permitió probar los componentes Hero y Filters en la interfaz pero no nos permite hacer cambios en los datos del objeto filters.

Para poder crear un estado dentro de este componente, lo primero que debemos hacer es convertir la declaración del componente App para que deje de ser una function y pase a ser una class que extiende de la clase [React.Component](https://reactjs.org/docs/state-and-lifecycle.html" \l "converting-a-function-to-a-class).

Una vez que tenemos nuestra clase App creada debemos agregarle un [constructor](https://reactjs.org/docs/state-and-lifecycle.html" \l "adding-local-state-to-a-class) en donde declararemos el estado del componente, que deberá contener el atributo filters con la misma estructura que antes.

this.state = {

filters: {

dateFrom: today,

dateTo: new Date(today.valueOf() + 86400000),

country: undefined,

price: undefined,

rooms: undefined

}

}

Por último es importante que actualicemos la estructura del componente para que éste utilice su state, que acabamos de crear, al pasarle los filtros a los componentes hijos.

render() {

return (

<div>

<Hero filters={ this.state.filters } />

<Filters filters={ this.state.filters } />

</div>

)

}

Ya logramos que el estado de los filtros sea modificable por React durante la ejecución. Ahora debemos definir un comportamiento que permita realizar dichas modificaciones cuando el usuario interactúa con la interfaz.

## **Componentes**DateFilter**y**OptionsFilter

Para lograr esto debemos recurrir al evento onChange de los elementos input y select en los componentes DateFilter y OptionsFilter respectivamente.

Esto se logra convirtiendo nuevamente la declaración de estos componentes de function a class que extiende de [React.Component](https://reactjs.org/docs/state-and-lifecycle.html" \l "converting-a-function-to-a-class).

El comportamiento que debemos agregar no debe intentar modificar el state de forma directa ya que no es parte de este componente hijo, sino que debería simplemente informar a su componente padre (Filters) que hubo un cambio, para que luego este pueda informar a su componente padre (App) del cambio y este último se encargue de modificar su propio state.

Aquí hay un ejemplo de cómo debería quedarnos el componente DateFilter. Intenta interpolar esta sintaxis al componente OptionsFilter también.

class DateFilter extends React.Component {

constructor(props) {

super(props)

this.handleDateChange = this.handleDateChange.bind(this)

}

handleDateChange(event) {

this.props.onDateChange(event)

}

render() {

...

<input className="input" type="date" onChange={ this.handleDateChange } value={ date } name={ this.props.name } />

...

}

}

Analiza a fondo el código anterior. Podrás notar que hay una nueva propiedad name que nos servirá luego para identificar cuál instancia del componente está generando el evento.

## **Componente**Filters

Ahora que ambos componentes hijos tienen previsto contener una propiedad de nombre (name) y una propiedad de evento (onDateChange / onOptionChange), debemos preparar el componente Filters para que el mismo pueda interpretar dichos eventos y a su vez informar a su componente padre (App) acerca de los mismos. Empecemos convirtiéndolo de function a class que extiende de [React.Component](https://reactjs.org/docs/state-and-lifecycle.html" \l "converting-a-function-to-a-class).

Aquí tienes un ejemplo de cómo debería ser la lógica del método onOptionchange.

handleOptionChange(event) {

let payload = this.props.filters

payload[event.target.name] = event.target.value

this.props.onFilterChange(payload)

}

Si lo analizas con atención podrás notar que el método tiene una funcionalidad muy simple, tomar el valor del objeto filters y reemplazar el valor del atributo que corresponde al nombre del input o el select que está disparando el evento. Una vez reemplazado ese dato simplemente llama a la función onFilterChange que recibe como prop y de la cual definiremos el comportamiento en el componente App.

Intenta llevar este mismo funcionamiento a un nuevo método con nombre onDateChange. El objetivo de esta función será el mismo pero deberás agregar la lógica para evitar que la fecha del atributo dateTo quede previas a la del atributo dateFrom.

## **Estado del componente**App

Ahora que hemos definido todo el funcionamiento de los distintos eventos generados por el componente Filters y sus hijos, debemos agregar al componente App el método que efectivamente modifica su state frente a estos cambios.

Para ello lo único que debemos hacer es agregar una función dentro del componente que llame al método setState de React pasando la nueva información de filtros generada por el componente Filters.

handleFilterChange(payload) {

this.setState({

filters: payload

})

}

Y luego no debemos olvidar pasar este nuevo método a través de las props del componente en cuestión.

<Filters filters={ this.state.filters } onFilterChange={ this.handleFilterChange } />

### **Checkpoint**

Si todo salió bien, ya puedes poner a prueba el comportamiento de los eventos del componente Filters y sus hijos al ver cómo éstos modifican el state del componente App a medida que reciben cambios por parte del usuario.

También deberías ver estos cambios de datos reflejados en el componente Hero que creaste anteriormente.

Por último, si quieres asegurarte de que todo está funcionando correctamente, puedes monitorear el state de tu aplicación durante su ejecución usando [React Developer Tools](https://github.com/facebook/react-devtools).