React – Conceptos avanzados

“Llevando React un paso más allá…”

React – Conceptos avanzados

- Fragments
- Refs y el DOM
- Higher Order Components (HOCs)
- Render props

Fragments

- Componente que permite agrupar una listado de *children* sin añadir nodos extras al DOM
  - Resuelven el problema de crear HTML no válido por añadir nodos intermedios
  - Fragments en un array deben llevar *key*

```
// sintaxis normal
class Columns extends React.Component {
  render() {
    return (
      <React.Fragment>
        <td>Hello</td>
        <td>World</td>
      </React.Fragment>
    );
  }
}
```

```
// sintaxis abreviada
class Columns extends React.Component {
  render() {
    return (
      <>
        <td>Hello</td>
        <td>World</td>
      </>
    );
  }
}
```

Refs y el DOM (I)

- Las *refs* nos permiten acceder a hijos y a toda su API desde un componente padre:
  - Elementos del DOM
  - Instancias de componentes
  - No podemos pasar refs a componentes function
- Sugerencias de uso:
  - Manejo de foco, selección de texto, reproducción media
  - Lanzar animaciones imperativamente
  - Integración de librerías de terceros de acceso a DOM
- Pero…**No abusar del uso de refs!!!**

Refs y el DOM (II) – Creando y accediendo a refs

- Creando una ref y pasándola a un elemento

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    // 1. creando y almacenando la ref
    this.myRef = React.createRef();
  }

  componentDidMount () {
    // 3. accediendo al elemento y a toda su API a través de la ref
    const node =  this.myRef.current;
    node.focus();
  }

  render() {
    // 2. pasando la ref al elemento
    return <input ref={this.myRef} />;
  }
}
```

Refs y el DOM (III) – Exponiendo refs

- Técnica para exponer la ref de un elemento al exterior de un componente, útil si estamos creando una librería de componentes que se comportan como los elementos nativos HTML.

```
// En lugar de definir el componente así…
const MyButton = props => (
  <button className="my-button">
    {props.children}
  </button>
);
// Lo definimos así
const MyButton = React.forwardRef((props, ref) => (
  <button ref={ref} className=“my-button">
    {props.children}
  </button>
));
// Y podemos obtener la ref al button
const ref = React.createRef();
<MyButton ref={ref}>Click me!</MyButton>;
```

Higher Order Components (I)

- Técnica avanzada para reusar lógica entre componentes
- Función que recibe como parámetro un componente y devuelve un nuevo componente

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

- Convenciones sobre HOCs:
  - Pasar props no relacionadas con el HOC
  - Potenciar la composición (*compose*)
  - *displayName* para facilitar el debug en React Developer Tools

Higher Order Components (II) - Advertencias

- No usar hocs dentro de un render:
  - Problemas de performance
  - Destrucción de estado
  - Debemos aplicar los HOCs estáticamente
- Los métodos estáticos de un componente se pierden
  - Copiar métodos estáticos
  - hoist-non-react-statics
- Problemas con colisión en los nombres de las props
- Las refs no son pasadas a través de HOCs (ref no son prop)
  - React.forwardRef

Render props

- Técnica para compartir código entre componentes usando una prop cuyo valor es una función
- **Render prop**: mediante una prop de tipo función delegamos la responsabilidad del renderizado en un componente padre
- El componente que recibe la render prop la ejecuta con sus datos para generar el elemento renderizado
- Podemos usar cualquier nombre para esta prop: *render, children…*
- Al contrario que los HOCs, la composición es dinámica
- Problemas de performance si definimos render prop inline, en lo posible, definir la render prop como método de instancia