



Curso de React y React Native

Clase 03



Agenda de la clase



Agenda

- Repaso.
- Lists en React
- Props.children
- PropTypes.
- Default Prop Values.
- Forms en React.
- Ejercicios.



Repaso



Lists en React



Lists – map (1)

En la mayoría de los frameworks existe alguna sintaxis especial para crear listas de elementos en la UI, por ejemplo, en **Angular** se usaría `ng-repeat`. En **React** se utiliza simplemente JavaScript, por ende, para crear una lista de elementos se puede utilizar el método nativo de JavaScript `map`.

`map` es un método (que tienen todos los arrays) que retorna un nuevo array con los elementos del array original habiéndose aplicado una función sobre cada uno de ellos.

```
var numbers = [1,2,3];
var numbersPlusTen = numbers.map(
  function (num) {
    return num + 10;
  }
);
console.log(numbersPlusTen);
// Resultado: [11, 12, 13]
```

```
// ES6:
const numbers = [1, 2, 3];
const numbersPlusTen = numbers.map(num => num + 10);
console.log(numbersPlusTen); // [11, 12, 13]
```



Lists – map (2)

Si se quiere crear una lista de elementos en la UI en React, se puede simplemente utilizar `map` y retornar elementos en la función. Por ejemplo, el siguiente componente recibe por *props* una lista de nombres de amigos:

```
const ShowList = (props) => {  
  return (  
    <div>  
      <h3>Amigos:</h3>  
      <ul>  
        {props.names.map((friend) => {  
          return <li> {friend} </li>;  
        })}  
      </ul>  
    </div>  
  );  
}
```

Lists – map (3)



Utilizando el componente anterior de la siguiente forma:

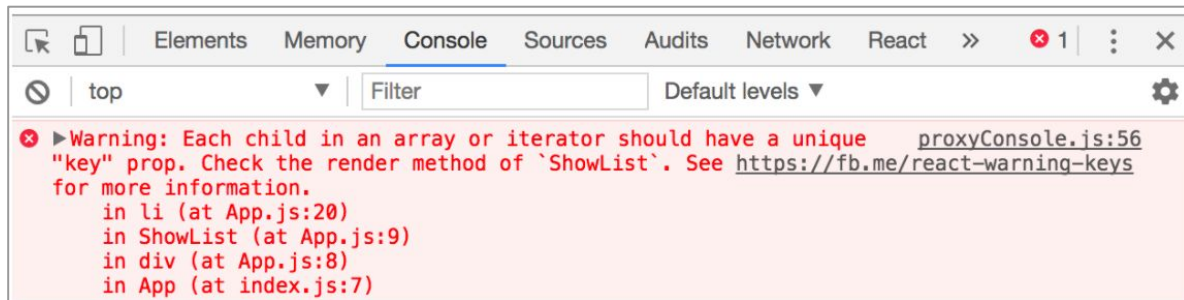
```
<ShowList names={["Diego", "Juan", "Pedro"]} />
```

En la página se verá el siguiente resultado:

Amigos:

- Diego
- Juan
- Pedro

Y en la consola se verá:





Lists – `key` (1)

Warning: Each child in an array or iterator should have a unique "key" prop

- Este es un *warning* que se verá siempre que un componente de React renderice una lista de elementos y no se le asigne una `key` propia a cada uno de ellos. Esto es un requerimiento de React para poder ser más eficiente a la hora de actualizar la UI.
- React utiliza esta `key` para identificar exactamente qué elemento fue modificado, eliminado o agregado en una lista de elementos, sin necesidad de volver a renderizar toda la lista.
- Esta `key` debe ser única. Si se cuenta con un `id` claramente es la mejor opción.



Lists – filter

Otro método de los arrays en JavaScript que suele ser útil en React es `filter`. Funciona de forma muy similar a `map` pero, en vez de aplicar una función sobre cada elemento, permite filtrar ciertos elementos que no cumplan la condición dada. En el siguiente ejemplo se obtienen sólo los nombres que empiecen con E:

```
var friends = ['Emiliano', 'Juan', 'Pablo', 'Ernesto', 'Tomás'];
var newFriends = friends.filter(function (name) {
  return name[0] === 'E'
});

console.log(newFriends) // Retorna: ['Emiliano', 'Ernesto']
```



Props.children



Props.children (1)

Cuando las expresiones JSX tienen tag de apertura y de cierre, el contenido entre ellas es enviado al componente como una propiedad especial, que se accede mediante `props.children`.

Estos "hijos" pueden ser de distintos tipos.

En el siguiente caso, `props.children` será simplemente un string:

```
<MyComponent>Hello world!</MyComponent>
```



Props.children (2)

En realidad, `props.children` es un array, lo cual es útil para poder brindar varios elementos como hijos.

Esto permite crear componentes que agreguen comportamiento a la aplicación y simplemente envuelvan al resto de los componentes. Pero para esto el "container" debe explícitamente renderizar a sus hijos, de la siguiente forma:

```
<MyContainer>
  <MyFirstComponent />
  <MySecondComponent />
</MyContainer>
```

```
class MyContainer extends Component {
  render() {
    return <div>{this.props.children}</div>
  }
}
```




PropTypes



PropTypes (1)

```
class Users extends Component {  
  render() {  
    return (  
      <ul>  
        {this.props.list.map((friend) => {  
          return <li>{friend}</li>  
        })}  
      </ul>  
    )  
  }  
}
```

Este componente espera una lista de nombres y realiza un `map` para mostrarlos como una lista no ordenada (``).



¿Qué pasaría si en vez de pasarle una lista en la **prop** `list`, se le pasa un string?

```
<Users list="Diego, Juan, Pedro" />
```



PropTypes (2)

Las **PropTypes** permiten declarar el **tipo** (string, número, función, etc) de cada ***prop*** que se pasa a un componente.

Si se pasa una ***prop*** que no es del tipo declarado, se verá una advertencia en la consola. Obviamente que la utilidad es durante el desarrollo de una aplicación, pero es muy útil para trabajar en equipos con más de un desarrollador o simplemente para escribir **componentes más "seguros"**.

Nota: Desde React v15.5 Prop Types se movió a un paquete separado, por lo que para poder utilizarlo es necesario instalarlo por `npm` o `yarn`:

- `npm install --save prop-types`
- `yarn add prop-types`



PropTypes (3)

El ejemplo visto anteriormente, declarando las **PropTypes** sería así:

```
import React, { Component } from "react";
import PropTypes from 'prop-types';
class Users extends Component {
  render() {
    return (
      <ul> {this.props.list.map((friend) => {
        return <li>{friend}</li>
      })} </ul>
    )
  }
}
Users.propTypes = {
  list: PropTypes.array.isRequired
}
```



PropTypes (4)

En el caso de definir los PropTypes, y cometer un error al utilizar un componente, se mostrará un *warning* en la consola como el siguiente:

```
►Warning: Failed prop type: Invalid prop `user` of type `string` supplied to `Avatar`, expected `object`.
  in Avatar (at UserInfo.js:8)
  in UserInfo (at Comment.js:8)
  in div (at Comment.js:7)
  in Comment (at App.js:44)
  in div (at App.js:42)
  in div (at App.js:37)
  in App (at index.js:7)
```



PropTypes (5)

La **API** completa se puede encontrar en <https://github.com/facebook/prop-types>.

Por defecto, siempre son opcionales. Estos son los tipos básicos, notar que las **funciones** y los **booleanos** son especiales.

```
MyComponent.propTypes = {  
  optionalArray: PropTypes.array,  
  optionalBool: PropTypes.bool,  
  optionalFunc: PropTypes.func,  
  optionalNumber: PropTypes.number,  
  optionalObject: PropTypes.object,  
  optionalString: PropTypes.string,  
  optionalSymbol: PropTypes.symbol,
```



PropTypes (6) – Más usos

```
MyComponent.propTypes = {  
  // Se puede agregar que la prop sea requerida  
  requiredFunc: PropTypes.func.isRequired,  
  // Un Elemento de React  
  optionalElement: PropTypes.element,  
  // Se puede declarar que una prop es una instancia de una clase  
  optionalMessage: PropTypes.instanceOf(Message),  
  // Se puede limitar que la prop esté limitada a valores especificados, como un enum  
  optionalEnum: PropTypes.oneOf(['News', 'Photos']),  
}
```



PropTypes (7) – Más usos

```
MyComponent.propTypes = {  
  // Un objeto de uno de varios tipos  
  optionalUnion: PropTypes.oneOfType([  
    PropTypes.string,  
    PropTypes.number,  
    PropTypes.instanceOf(Message)  
  ]),  
  // Un objeto con determinada forma  
  optionalObjectWithShape: PropTypes.shape({  
    color: PropTypes.string,  
    fontSize: PropTypes.number  
  }),  
  // Un array de un tipo de datos  
  optionalArrayOf: PropTypes.arrayOf(PropTypes.number),  
}
```



Default Prop Values



Default Prop Values

Se pueden definir valores por defecto para las **Props**, para el caso en que no sean especificadas, utilizando `defaultProps`.

El chequeo de los tipos definidos en las **PropTypes** sucede luego de interpretar las **Default Props**, por lo que también validará estos valores contra las reglas definidas.



Default Prop Values – Ejemplo

```
class Greeting extends React.Component {  
  render() {  
    return (  
      <h1>Hello, {this.props.name}</h1>  
    );  
  }  
}
```

```
// El valor por defecto:  
Greeting.defaultProps = {  
  name: 'Stranger'  
};  
Greeting.propTypes = {  
  name: PropTypes.string  
};
```

```
<Greeting />  
<Greeting name="dieg0" />
```




Forms en React



Forms – Text Area

En HTML un elemento `<textarea>` define su contenido mediante sus hijos.

```
<textarea>
  Texto en un textarea, como hijo el elemento html
</textarea>
```

En React, se utiliza el atributo `value` (al igual que con un `input`):

```
<form onSubmit={this.handleSubmit}>
  <label>
    Name:
    <textarea value={this.state.value} onChange={this.handleChange} />
  </label>
  <input type="submit" value="Submit" />
</form>
```



Forms – Select

En HTML para indicar la opción seleccionada en un `select` se puede utilizar el atributo `selected` dentro de un `option`.

```
<select>
  <option value="grapefruit">Grapefruit</option>
  <option value="lime">Lime</option>
  <option selected value="coconut">Coconut</option>
  <option value="mango">Mango</option>
</select>
```

En React, en vez de esto, se utiliza el `value` en el tag raíz `select`, de forma de que solo haya que actualizar el valor seleccionado en un lugar. En las próximas slides, se muestra un ejemplo.



Forms – Select (ej. Parte 1)

```
// PARTE 1
class FlavorForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'coconut'};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('Your favorite flavor is: ' + this.state.value);
    event.preventDefault();
  }
}
```



Forms – Select (ej. Parte 2)

```
// PARTE 2
render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Pick your favorite Icecream flavor:
        <select value={this.state.value} onChange={this.handleChange}>
          <option value="grapefruit">Grapefruit</option>
          <option value="lime">Lime</option>
          <option value="coconut">Coconut</option>
          <option value="mango">Mango</option>
        </select>
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```



Forms – Manejando múltiples inputs (1)

En el caso de tener que manejar varios inputs en un componente, no es necesario crear un *handler* para cada uno de ellos, sino que se puede aprovechar el atributo `name`, asignándole un `name` distinto a cada elemento, y luego en el *handler* decidir qué actualizar en función del `event.target.name`.

```
handleInputChange(event) {  
  const target = event.target;  
  const value = target.type === 'checkbox' ? target.checked : target.value;  
  const name = target.name;  
  
  this.setState({  
    [name]: value  
  });  
}
```

¡Computed Properties de ES6!



Forms – Manejando múltiples inputs (2)

```
class Reservation extends React.Component {
  state = { isGoing: true, numberOfGuests: 2 };
  render() {
    return (
      <form>
        <label>
          Is going: <input name="isGoing" type="checkbox" checked={this.state.isGoing}
            onChange={this.handleInputChange} />
        </label><br/>
        <label>
          Number of guests: <input name="numberOfGuests" type="number"
            value={this.state.numberOfGuests} onChange={this.handleInputChange} />
        </label>
      </form>
    );
  }
}
```

El valor de name coincide con la clave del state



Ejercicios



Ejercicios

1. Crear un proyecto de react
2. Crear un formulario de contacto que incluya los siguientes campos:
 - a. Nombre
 - b. Apellido
 - c. Edad
 - d. Email
 - e. Género (un select con 3 opciones: “femenino”, “masculino”, “no especifica”)
 - f. “Carta de presentación” (text area)
3. Agregar un botón de ENVIAR
4. Cada vez que se realice un cambio en un campo, imprimir en consola el nombre del campo y su último valor.
5. Cuando se ENVÍE el formulario, imprimir en consola el estado final completo del formulario.

Ejercicios



6. Conforme se envíen los formularios, ir guardando los mismos en el estado, acumulandolos en un array.
7. Crear un componente que mostraremos debajo del formulario y que renderizará en pantalla los mismos de forma amigable (hacer un `map` del array de sumisiones de formularios que tenemos en el estado). Dicho componente recibirá los valores del form por props y tendrá asignadas `propTypes` para dichas props.
8. Crear un componente Layout que recibirá como `children` tanto el form como el listado de sumisiones y que lo coloque en centro de la pantalla (`children` suele usarse mucho con fines de layout).