El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

CICLO IV:

Desarrollo de Aplicaciones Web

MisionTIC2022

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

Sesión 08:

Desarrollo de Aplicaciones Web

Desarrollo de Front-End web con React - Patrones de Diseño

MisiónTIC2022

Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

1. Implementar patrones de diseño de React en proyectos de desarrollo de aplicaciones web.

React - Listas

- Es posible que necesitemos retornar en nuestros componentes uno o más elementos, inicialmente uno podría pensar que se hace de la siguiente forma:

```
import React from 'react'

const Multiple = () => {
    return (
            <p>Linea 1</p>
            <p>Linea 2</p>
    );
}

export default Multiple;
```

- Si realizamos esto obtendremos un error, ya que de la forma en la que React compila sus componentes esto no es posible.

React - Listas

- Por lo que, para mostrar varios elementos necesitamos usar lo que se conoce como el componente Fragment, el cual proviene de React, esto se puede hacer de las siguientes formas:

```
import React from 'react'

const Multiple = () => {
    return (
        <React.Fragment>
            <p>Linea 1</p>
            <p>Linea 2</p>
        </React.Fragment>
    );
}

export default Multiple;
```

```
import React, {Fragment} from 'react'

const Multiple = () => {
    return (
        <Fragment>
            <p>Linea 1</p>
            <p>Linea 2</p>
        </Fragment>
    );
}

export default Multiple;
```

```
import React, from 'react'

const Multiple = () => {
    return (
        <>
            <p>Linea 1</p>
            <p>Linea 2</p>
        </>
    );
}

export default Multiple;
```

- Cabe resaltar que la última forma, es la que se usa actualmente debido a que es su versión resumida.

React - Listas

- De igual forma podemos mostrar un vector de elementos de la siguiente manera:

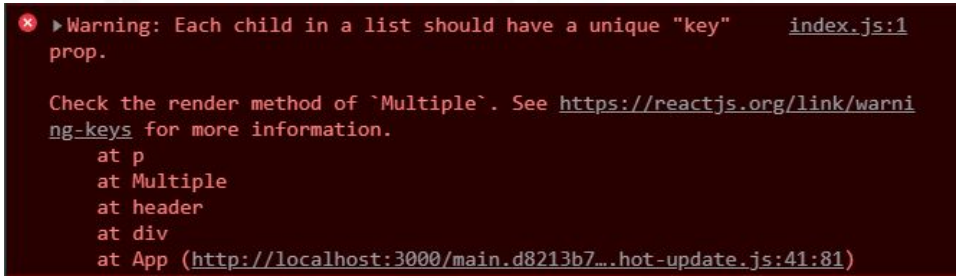
```
import React from 'react';

const Multiple = () => {
    const lista = ['linea 1', 'linea 2', 'linea 3'];
    return (
        <>
            {lista.map(elemento => (
                <p>{elemento}</p>
            ))}
        </>
    );
};

export default Multiple;
```

React - Listas

- Al realizar esto observaremos nuestro listado de elementos, pero si revisamos las herramientas de desarrollador observaremos la siguiente advertencia:



- Esto se debe a que para poder identificar diferencias entre cada elemento, React necesita una forma única de poder identificarlos.
- A este identificador se le conoce como key o llave.

React - Listas

- Por lo general se usa una propiedad del objeto, o el mismo objeto. En situaciones donde no hay un buen key para escoger podemos seleccionar el índice del elemento en la lista.
- Utilizar los índices de los elementos puede generar conflictos con el renderizado de nuestra app, ya que si en algún momento nuestros elementos son modificados React tendrá problemas renderizando el listado de forma correcta.
- Esto a se debe a que asocia un estado único a cada elemento y ya que hay más de uno, React lo reconoce como idéntico y no es capaz de persistir su estado.
- Para más información revisar la documentación de React y este artículo.
- Adicionalmente, puede revisar este repositorio de JSBin a modo de demostración.

React - Listas

- Finalmente nuestro componente `<Multiple />` quedaría de la siguiente forma:

```
import React from 'react';

const Multiple = () => {
    const lista = ['linea 1', 'linea 2', 'linea 3'];
    return (
        <>
            {lista.map(elemento => (
                <p key={elemento} >{elemento}</p>
            ))}
        </>
    );
};

export default Multiple;
```

- Tomando en cuenta el supuesto que cada elemento es diferente.

React - Listas

- Entonces podemos reescribir **<Multiple />** de la siguiente forma:

```
import React from 'react';

const Mensaje = ({ mensaje }) => <p>{mensaje}</p>;

const Multiple = () => {
    const lista = ['linea 1', 'linea 2', 'linea 3'];
    return (
        <>
            {lista.map(elemento => (
                <Mensaje key={elemento}
mensaje={elemento} />
            ))}
        </>
    );
};

export default Multiple;
```

- Tenemos el componente **<Mensaje />**.
- La prop key va directamente sobre el elemento que es retornado por nuestra lista.

React - Listas

- De igual forma, podemos reescribir nuestros componentes de la siguiente forma para mejor su legibilidad:

```
import React from 'react';

const Mensaje = ({ mensaje }) => <p>{mensaje}</p>;

const Multiple = () => {
    const lista = ['linea 1', 'linea 2', 'linea 3'];
    const mostrarLista = lista =>
        lista?.map(elemento => <Mensaje key={elemento} mensaje={elemento} />);

    return <>{mostrarLista(lista)}</>;
};

export default Multiple;
```

React - Listas

- Finalmente, podemos trabajar con un atributo `id` para diferenciar nuestros elementos:

```
import React from 'react';

const Mensaje = ({ mensaje }) => <p>{mensaje}</p>;

const Multiple = () => {
    const lista = [
        { id: 1, msg: 'a' },
        { id: 2, msg: 'b' },
        { id: 3, msg: 'c' },
    ];
    const mostrarLista = listado =>
        listado?.map(elemento => (
            <Mensaje key={elemento.id} mensaje={elemento.mensaje} />
        ));

    return <>{mostrarLista(lista)}</>;
};

export default Multiple;
```

React - Formularios

- Para mostrar formularios podemos crear un componente que contenga un formulario con la misma estructura que en HTML, lo único que cambiaría sería la nomenclatura de los eventos:

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

const NameForm = () => {
  const [value, setValue] = useState('');

  const changeHandler = (event) => setValue(event.target.value);
  const submitHandler = (event) => {
    event.preventDefault();
    alert('Nombre: ' + value);
  };

  return (
    <form onSubmit={submitHandler}>
      <label htmlFor="name"> Nombre: </label>
      <input type="text" name="name" value={value} onChange={changeHandler} />
      <input type="submit" value="Enviar" />
    </form>
  );
};

export default NameForm;
```

- Proyecto en StackBlitz.
- Demostración en vivo.
- Para este ejemplo usaremos el `input` tag, para ver ejemplos con otros tags, ver la documentación de React.

React - Formularios

- Adicionalmente podemos agregar validación a nuestro campo `name`.
- Para esto podemos añadir la validación de tres formas diferentes:
  - A nivel de formulario.
  - A nivel de campo.
  - Con cada cambio del usuario.
- Para mostrar un mensaje informativo al usuario usaremos otro estado que llamaremos `error`.

React - Formularios

- Inicialmente, definimos nuestra validación con una RegExp o expresión regular para solo admitir nombres de personas latinas:

```
const MIN = 2;
const MAX = 50;
const GROUP = `[a-zñáéíóúüA-ZÁÉÍÓÚÜÑ]{${MIN},${MAX}}`;
const VALIDATION = new RegExp(`^(${GROUP})( ${GROUP})*$`);
```

- Agregamos nuestro estado de error:

```
const [error, setError] = useState('');
```

- Y mostramos nuestro error en caso de que tengamos uno:

```
{Boolean(error) && <p>{error}</p>}
```

React - Formularios

- Luego, configuramos nuestra validación a nivel de formulario en nuestro `submitHandler` de la siguiente forma:

```
const submitHandler = (event) => {
  event.preventDefault();

  const isValid = VALIDATION.test(value);
  console.log({ VALIDATION, isValid });
  if (isValid) {
    alert('Nombre: ' + value);
    setError('');
  } else setError('Nombre no válido');
};
```

- Para poder ver esta implementación, revisar el componente `<NameFormWithFormValidation />` en nuestro proyecto de StackBlitz.

React - Formularios

- Por otro lado, para configurar nuestra validación a nivel de campo, debemos escuchar el evento `onBlur` para ejecutar nuestra validación al perder el foco del campo:

```
const blurHandler = () => {
  const isValid = VALIDATION.test(value);

  if (isValid) setError('');
  else setError('Nombre no válido');
};
```

React - Formularios

- Luego, en nuestro campo escuchamos el evento y finalmente modificamos nuestro `submitHandler`
  - `input: <input type="text" name="name" value={value} onChange={changeHandler} onBlur={blurHandler} />`
  - `submitHandler: if (!Boolean(error)) alert('Nombre: ' + value);`
- Ver cambios en `<NameFormWithFieldValidation />`, en nuestro proyecto de StackBlitz.

React - Formularios

- Por último, para configurar nuestra validación con cada cambio en el campo `name` nos deshacemos de nuestro evento `onBlur` y editamos `changeHandler` de la siguiente forma:

```
const changeHandler = (event) => {
  setValue(event.target.value);
  const isValid = VALIDATION.test(event.target.value);

  if (isValid) setError('');
  else setError('Nombre no válido');
};
```

- Para poder ver esta implementación, de cambios por “cambio” de campo, revisar el componente `<NameFormWithKeyStrokeValidation />` en nuestro proyecto de StackBlitz.

React - Formularios

- Ya que tenemos nuestra configuración por cambios de campo, podríamos construir un validador personalizado de la siguiente forma:

```
const CUSTOM_VALIDATION = (input) => {
  const matchesRegex = VALIDATION.test(input.trim());

  if (input.length < MIN) return `Se necesitan mínimo ${MIN} caracteres`;
  else if (input.length > MAX * 4)return `Se necesitan aceptan  máximo ${MAX * 4}
caracteres`;
  else if (!matchesRegex) return 'Formato inválido';
  else return '';
};
```

React - Formularios

- Editamos nuestro `changeHandler`:

```
const changeHandler = (event) => {
  setValue(event.target.value);

  const error = CUSTOM_VALIDATION(event.target.value);
  setError(error);
};
```

- Cambios en `<IdealNameForm />` via StackBlitz.

React - Levantar Estado

- Levantar Estado es necesario cuando necesitamos compartir estados entre diferentes componentes, consideremos el siguiente diagrama:

```
<App>
     <A />
     <B />
<App />
```

- En este caso, podríamos requerir que el componente `<A />` modifique información en el componente `<B />` y viceversa.
- Para esto normalmente centramos la lógica en nuestro componente padre `<App />`.
- Y en cada nodo hijo `<A />` y `<B />`, aceptamos eventos personalizados para levantar nuestro estado.

React - Levantar Estado

- Consideremos que queremos realizar una calculadora de temperatura en grados Celsius a Farenheit y viceversa.
- Adicionalmente, queremos saber si la temperatura introducida es mayor o menor al punto de ebullición del agua.
  - Se sabe que para °C es de 100 °C.
  - Se sabe que para °F es de 212 °F.
- Adicionalmente, sabemos las conversiones:
  - F => C sigue la fórmula `(farenheit - 32) * (9/5)`.
  - C => F sigue la fórmula `celsius * (5/9) + 32`.
- Ejemplo adaptado de la documentación de React.

React - Levantar Estado

- Primero consideremos estas utilidades:

```
const CONVERSIONS = {
  C: (f) => (f - 32) * (5 / 9), // from F to C: (fahrenheit  -32) * (5/9);
  F: (c) => c * (9 / 5) + 32, //   from C to F: (celsius    *9/5) + 32;
};
const VALID_UNITS = ['C', 'F'];
export const convertTo = (from, to, value) => {
  if (value.length === 0) return '';
  else if (from === to) return value;
  else if (VALID_UNITS.includes(to)) return CONVERSIONS[to](value);
  else return '';
};
const WATER_BOILING_POINT = {
  C: 100,
  F: 212,
};
export const isWaterBoiling = (unit, value) =>
  value >= WATER_BOILING_POINT[unit];
```

- Estas utilidades se encontraran en **src/utilities/temperature.js**.
- Ver StackBlitz.

React - Levantar Estado

- De igual forma consideremos el componente `<TemperatureField />`:

```
import React from 'react';
import { convertTo } from '../../utilities/temperature';

const TemperatureField = ({ value, from, to, onUpdate }) => {
  const temperature = convertTo(from, to, value);
  const changeHandler = (event) => {
    const value = event.target.value;
    const isNumeric = !isNaN(value);
    if (isNumeric) onUpdate(to, value);
  };
  return (
    <div>
      <label htmlFor={`T-${to}`}>°{to}  </label>
      <input type="number" name={`T-${to}`} value={temperature} onChange={changeHandler} />
    </div>
  );
};
export default TemperatureField;
```

- Este componente se encuentra en **src/components/TemperatureField.js**.
- Ver StackBlitz.

React - Levantar Estado

- Por último consideremos <App />:

```
import React, { useState } from 'react';
import TemperatureField from './components/TemperatureField';
import { isWaterBoiling } from './utilities/temperature';
import './style.css';

export default function App() {
  const [value, setValue] = useState('');
  const [unit, setUnit] = useState('');
  const show = Boolean(unit && value);
  const updateHandler = (newU, newT) => { setValue(newT); setUnit(newU); };
  const formatTemperatureLabel = () => show && ` (${value} °${unit}):`;
  const getAnswer = () => show && (isWaterBoiling(unit, value) ? ' Si' : ' No');
  return (<div>
      <h1>¿Está el agua hirviendo?</h1>
      <h2> Temperatura {formatTemperatureLabel()} {getAnswer()} </h2>
      <TemperatureField to="C" value={value} from={unit} onUpdate={updateHandler} />
      <TemperatureField to="F" value={value} from={unit} onUpdate={updateHandler} />
  </div>);
}
```

- Estas utilidades se encontraran en `src/App.js`.
- Ver StackBlitz.

React - Levantar Estado

- Para ver la demostración en vivo visitar este enlace.
- Como se puede observar, lo que hacemos es que desde `<App />` definimos dos estados `setUnit` y `setValue`, los cuales nuestros componentes `<TemperatureField />` consumirán mediante el evento `onUpdate`.
- Este evento captura los cambios de los campos y se lo comunica a `<App />`, levantando así el estado para ambas variables `unit` y `value`.
- De esta forma podemos actualizar un estado y propagarlo a diferentes componentes.

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

Ejercicios de práctica

Mision TIC 2022

Referencias

- https://reactjs.org/docs/lists-and-keys.html
- https://robinpokorny.medium.com/index-as-a-key-is-an-anti-pattern-e0349aece318
- https://reactjs.org/docs/forms.html
- https://reactjs.org/docs/lifting-state-up.html

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

¡GRACIAS

POR SER PARTE DE ESTA EXPERIENCIA DE APRENDIZAJE!

Mision TIC 2022