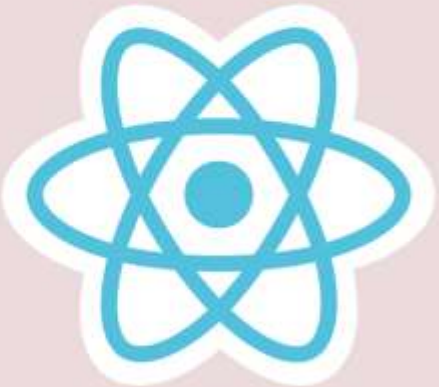
5_1.4

www.opentowork.es

ReactJS

Eventos, condicionales, listas, formularios, rutas y estilo

En React, los eventos permiten interacciones como clics y cambios de entrada, esenciales para la dinámica de la interfaz. Las expresiones condicionales controlan qué elementos se renderizan, adaptando la UI a diferentes estados. Las listas se manejan eficientemente a través de la renderización de arrays. Los formularios recogen y validan datos de usuario, mientras que las rutas permiten la navegación entre vistas. Finalmente, el estilo se aplica con CSS para personalizar la apariencia.



#01/24@ mihifidem

Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

1. Eventos en React
1.1. Añadir eventos
1.2. Pasar argumentos
1.3. Objeto React Evento

2. React condicionales
2.1. Declaración if
2.2. Operador lógico &&
2.3. Operador ternario

3. Listas en React
3.1. Keys

4. Formularios en React
4.1. Agregar formularios
4.2. Manejo de formularios
4.3. Envio de formularios
4.4. Multiples campos de entrada
4.5. Textarea
4.6. Select

5. Rutas en React
5.1. Agregar enrutador
5.2. Estructura de carpetas
5.3. Uso básico
5.4. Páginas / Componentes

6. Dar estilo a react usando CSS
6.1. Estilo en línea
6.2. camelCased Nombres de propiedad
6.3. Objeto Javascript
6.4. Hoja de estilos
6.5. Módulo CSS

Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

1. React Eventos

Al igual que los eventos HTML DOM, React puede realizar acciones en función de los eventos del usuario.React tiene los mismos eventos que HTML: hacer clic, cambiar, pasar el mouse, etc.

1.1. Añadir eventos

- Los eventos React están escritos en sintaxis

camelCase:onClick en lugar de onclick

- Los controladores de eventos React se escriben entre

llaves:onClick={shoot} en lugar de onClick="shoot()"

REACT

```
<button onClick={shoot}>Take the Shot!</button>
```

HTML

```
<button onclick="shoot()">Take the Shot!</button>
```

EJEMPLO

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = () =>
  {
    alert("Great Shot!");
  }

  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

1.2. Pasar argumentos

Para pasar un argumento a un controlador de eventos, use una función de flecha.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = (a)
  => {
    alert(a);
  }

  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

1.3. Objeto React Evento

Los controladores de eventos tienen acceso al evento React que activó la

función.En nuestro ejemplo, el evento es el evento "clic".

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = (a, b)
    => {alert(b.type);
               /*
               'b' represents the React event that triggered the function.
    In this case, the 'click' event
               */
  }

  return (
    <button onClick={(event) => shoot("Goal!", event)}>Take the shot!</button>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

2. React condicionales

2.1. Declaración if

Podemos usar el operador if de JavaScript para decidir qué componente renderizar.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function MissedGoal() {
       return <h1>MISSED!</h1>;
}

function MadeGoal() {
       return <h1>GOAL!</h1>;
}

function Goal(props) {
  const isGoal =
  props.isGoal;if (isGoal)
  {
    return <MadeGoal/>;
  }
  return <MissedGoal/>;
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Goal isGoal={false} />);
```

2.2. Operador lógico &&

Otra forma de renderizar condicionalmente un componente de React es usando el operador &&.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Garage(props)
  { const cars =
  props.cars;return (
    <>
      <h1>Garage</h1>
      {cars.length > 0 &&
        <h2>
          You have {cars.length} cars in your garage.
        </h2>
   }
  </>
  );
}

const cars = ['Ford', 'BMW', 'Audi'];
const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage cars={cars} />);
```

Si cars.length > 0 es igual a verdadero, la expresión posterior &&se representará.

2.3. Operador ternario

Otra forma de renderizar elementos condicionalmente es usando un operador ternario.

condition ? true : false

Devuelve el componente MadeGoal si isGoales true, de lo contrario devuelve el componenteMissedGoal

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function MissedGoal() {
        return <h1>MISSED!</h1>;
}

function MadeGoal() {
        return <h1>GOAL!</h1>;
}

function Goal(props) {
  const isGoal =
        props.isGoal;return
        (
                <>
                        { isGoal ? <MadeGoal/> : <MissedGoal/> }
                </>
        );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Goal isGoal={false} />);
```

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

3. Listas en React

En React, renderizarás listas con algún tipo de bucle.

El método map() de matriz de JavaScript es generalmente el método preferido.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Car(props) {
  return <li>I am a { props.brand }</li>;
}

function Garage() {
  const cars = ['Ford', 'BMW',
  'Audi'];return (
    <>
            <h1>Who lives in my garage?</h1>
            <ul>
        {cars.map((car) => <Car brand={car} />)}
      </ul>
    </>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);

/*
If you run this example in your create-react-app,
you will receive a warning that there is no "key" provided for the list items.
*/
```

Cuando ejecute este código en su create-react-app, funcionará, pero recibirá una advertencia de que no se proporciona una "clave" para los elementos de la lista.

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

3.1. Keys

Las claves permiten a React realizar un seguimiento de los elementos. De esta forma, si se actualiza o elimina unelemento, solo se volverá a representar ese elemento en lugar de la lista completa.

Las claves deben ser únicas para cada hermano. Pero se pueden duplicar globalmente.

Generalmente, la clave debe ser una identificación única asignada a cada elemento. Como último recurso, puede utilizar el índice de matriz como clave.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Car(props) {
  return <li>I am a { props.brand }</li>;
}

function
  Garage() {
  const cars = [
    {id: 1, brand: 'Ford'},
    {id: 2, brand: 'BMW'},
    {id: 3, brand: 'Audi'}
  ];
  return (
    <>
            <h1>Who lives in my garage?</h1>
            <ul>
        {cars.map((car) => <Car key={car.id} brand={car.brand} />)}
      </ul>
    </>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);
```

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

4. Formularios en React

Al igual que en HTML, React usa formularios para permitir que los usuarios interactúen con la página web.

4.1. Agregar formularios

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function
  MyForm() {
  return (
    <form>
    <label>Enter your name:
      <input type="text" />
     </label>
    </form>
  )
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

Esto funcionará normalmente, el formulario se enviará y la página se

actualizará.Pero esto generalmente no es lo que queremos que suceda en

React.

Queremos evitar este comportamiento predeterminado y dejar que React controle el formulario.

4.2. Manejo de formularios

El manejo de formularios se trata de cómo maneja los datos cuando cambia de valor o se

envía.En HTML, los datos del formulario generalmente son manejados por el DOM.

En React, los datos del formulario generalmente son manejados por los componentes.

Cuando los datos son manejados por los componentes, todos los datos se almacenan en el estado delcomponente.

Puede controlar los cambios agregando controladores de eventos en el onChangeatributo.

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

Podemos usar useStateHook para realizar un seguimiento de cada valor de entrada y proporcionar una "fuenteúnica de verdad" para toda la aplicación.

```
import { useState } from "react";
import ReactDOM from 'react-
dom/client';

function MyForm() {
  const [name, setName] = useState("");

return (
  <form>
      <label>Enter your name:
        <input
          type="tex
          t"
          value={na
          me}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
    </form>
  )
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

4.3. Envio de formularios

Puede controlar la acción de envío agregando un controlador de eventos en el onSubmitatributo para <form>:

```
import { useState } from "react";
import ReactDOM from 'react-
dom/client';

function MyForm() {
  const [name, setName] = useState("");

  const handleSubmit = (event)
    => {event.preventDefault();
    alert(`The name you entered was: ${name}`);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:
        <input
          type="tex
          t"
          value={na
          me}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

4.4. Multiples campos de entrada

- Puede controlar los valores de más de un campo de entrada agregando un nameatributo a cadaelemento.
- Inicializaremos nuestro estado con un objeto vacío.
- Para acceder a los campos en el controlador de eventos, use la sintaxis event.target.nameyevent.target.value.
- Para actualizar el estado, use corchetes [notación de corchetes] alrededor del nombre de la propiedad.

```
import { useState } from "react";
import ReactDOM from "react-
dom/client";

function MyForm() {
  const [inputs, setInputs] = useState({});

  const handleChange = (event) =>
    { const name =
    event.target.name; const value
    = event.target.value;
    setInputs(values => ({...values, [name]: value}))
  }

  const handleSubmit = (event)
    => {event.preventDefault();
    console.log(inputs);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:
      <input
        type="text"
        name="userna
        me"
        value={inputs.username ||
        ""}onChange={handleChange}
      />
      </label>
      <label>Enter your age:
        <input
          type="numb
          er"
          name="age"
          value={inputs.age ||
          ""}
          onChange={handleChang
          e}
        />
```

```
        </label>
        <input type="submit" />
```

```
    </form>
  )
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);

/*
Click F12 and navigate to the "Console
view"to see the result when you submit
the form.
*/
```

Nota: Usamos la misma función de controlador de eventos para ambos campos de entrada, podríamos escribir un controlador de eventos para cada uno, pero esto nos brinda un código mucho más limpio y es la forma preferida en React.

4.5. Textarea

El elemento textarea en React es ligeramente diferente del HTML ordinario.

En HTML, el valor de un área de texto era el texto entre la etiqueta inicial <textarea> y la etiqueta final </textarea>.

```
<textarea>

  Content of the textarea.

</textarea>
```

En React, el valor de un área de texto se coloca en un atributo de valor. Usaremos el useStategancho paraadministrar el valor del área de texto:

```
import { useState } from "react";
import ReactDOM from "react-
dom/client";

function MyForm() {
  const [textarea, setTextarea] = useState(
    "The content of a textarea goes in the value attribute"
  );

  const handleChange = (event)
    => {
    setTextarea(event.target.val
    ue)
  }

return (
  <form>
      <textarea value={textarea} onChange={handleChange} />
    </form>
  )
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

4.6. Select

Una lista desplegable, o un cuadro de selección, en React también es un poco diferente de HTML.en HTML, el valor seleccionado en la lista desplegable se definió con el selectedatributo:

```
<select>
  <option value="Ford">Ford</option>
  <option value="Volvo" selected>Volvo</option>
  <option value="Fiat">Fiat</option>
</select>
```

En React, el valor seleccionado se define con un value atributo en la selectetiqueta:

```
import { useState } from "react";
import ReactDOM from "react-
dom/client";

function MyForm() {
  const [myCar, setMyCar] = useState("Volvo");

  const handleChange = (event)
    => {
    setMyCar(event.target.value)
  }

return (
  <form>
      <select value={myCar} onChange={handleChange}>
        <option value="Ford">Ford</option>
        <option value="Volvo">Volvo</option>
        <option value="Fiat">Fiat</option>
      </select>
    </form>
  )
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

Al hacer estos ligeros cambios en <textarea>y <select>, React puede manejar todos los elementos de entrada dela misma manera.

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

5. Rutas en React

Create React App no incluye el enrutamiento de páginas.

5.1. Agregar enrutador

Para agregar React Router en su aplicación, ejecútelo en la terminal desde el directorio raíz de la aplicación:

```
npm i -D react-router-dom
```

5.2. Estructura de carpetas

Para crear una aplicación con varias rutas de página, primero comencemos con la estructura del

archivo.Dentro de la srccarpeta, crearemos una carpeta nombrada pagescon varios archivos:

src\pages\:

Layou

t.js

Home

.js

Blogs.

js

Conta

ct.js

NoPa

ge.js

Cada archivo contendrá un componente React muy básico

5.3. Uso básico

Ahora usaremos nuestro Router en nuestro index.jsarchivo.

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-
dom";import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from
"./pages/Contact";import NoPage
from "./pages/NoPage";

export default function
  App() {return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Envolvemos nuestro contenido primero con <BrowserRouter>.

Luego definimos nuestro <Routes>. Una aplicación puede tener múltiples <Routes>. Nuestro ejemplo básico solousa uno.

<Route>s se pueden anidar. El primero <Route>tiene una ruta / y representa el Layoutcomponente.

Los <Route>s anidados heredan y se agregan a la ruta principal. Entonces la blogsruta se combina con el padre yse convierte en /blogs.

La Homeruta del componente no tiene una ruta pero tiene un indexatributo. Eso especifica esta ruta como laruta predeterminada para la ruta principal, que es /.

Establecer el pathto *actuará como un cajón de sastre para cualquier URL indefinida. Esto es genial para unapágina de error 404.

5.4. Páginas / Componentes

El Layoutcomponente tiene <Outlet>y

<Link>elementos.El <Outlet>renderiza la ruta actual

seleccionada.

<Link>se utiliza para establecer la URL y realizar un seguimiento del historial de

navegación.Cada vez que nos vinculemos a una ruta interna, usaremos <Link> en lugar

de <a href="">.

La "ruta de diseño" es un componente compartido que inserta contenido común en todas las páginas, como unmenú de navegación.

Layout.js:

```
import { Outlet, Link } from "react-router-dom";

const Layout = ()
  => {return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />
    </>
  )
};

export default Layout;
```

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

Home.js:

```
const Home = () => {
  return
  <h1>Home</h1>;
};

export default Home;
```

Blogs.js:

```
const Blogs = () => {
  return <h1>Blog Articles</h1>;
};

export default Blogs;
```

Contact.js:

```
const Contact = () => {
  return <h1>Contact
  Me</h1>;
};

export default Contact;
```

NoPage.js:

```
const NoPage = ()
  => {return
  <h1>404</h1>;
};

export default NoPage;
```

6. Dar estilo a react usando CSS

Hay muchas formas de diseñar React con CSS, este tutorial analizará más de cerca tres formas comunes:

- estilo en línea
- hojas de estilo CSS
- Módulos CSS
-

6.1. Estilo en línea

Para diseñar un elemento con el atributo de estilo en línea, el valor debe ser un objeto de JavaScript:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const Header = ()
  => {return (
    <>
      <h1 style={{color: "red"}}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

Nota: en JSX, las expresiones de JavaScript se escriben entre llaves y, dado que los objetos de JavaScript también usan llaves, el estilo del ejemplo anterior se escribe entre dos conjuntos de llaves {{}}.

6.2. camelCased Nombres de propiedad

Dado que el CSS en línea está escrito en un objeto JavaScript, las propiedades con separadores de guiones, comobackground-color, deben escribirse con la sintaxis de mayúsculas y minúsculas camel:

Usar `backgroundColor`en lugar de `background-color`:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const Header = ()
  => {return (
    <>
      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

6.3. Objeto Javascript

También puede crear un objeto con información de estilo y hacer referencia a él en el atributo de

estilo:Cree un objeto de estilo llamado myStyle:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

 const Header = ()
=> {const myStyle =
 { color: "white",
    backgroundColor:
    "DodgerBlue",padding:
    "10px",
    fontFamily: "Sans-Serif"
  };
  return (
    <>
      <h1 style={myStyle}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

6.4. Hoja de estilos

Puede escribir su estilo CSS en un archivo separado, simplemente guarde el archivo con la .cssextensión dearchivo e impórtelo en su aplicación.

Aplicación.css:

Cree un nuevo archivo llamado "App.css" e inserte un código CSS en él:

```css
body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}
```

Nota: puede llamar al archivo como desee, solo recuerde la extensión de archivo correcta.

Importe la hoja de estilo en su aplicación:

```
import React from 'react';
import ReactDOM from 'react-
dom/client';import './App.css';

const Header = ()
  => {return (
    <>
      <h1>Hello Style!</h1>
      <p>Add a little style!.</p>
    </>
  );
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

6.5. Módulo CSS

Otra forma de agregar estilos a su aplicación es usar Módulos CSS.

Los módulos CSS son convenientes para los componentes que se colocan en archivos separados.

El CSS dentro de un módulo está disponible solo para el componente que lo importó y no tiene que preocuparsepor los conflictos de nombres.

Cree el módulo CSS con la .module.css extensión, ejemplo: my-style.module.css.

Cree un nuevo archivo llamado "my-style.module.css" e inserte un código CSS

en él:mi-estilo.módulo.css:

```
.bigblue {
  color: DodgerBlue;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}
```

Importe la hoja de estilo en su

componente:Coche.js:

```
import styles from './my-style.module.css';

const Car = () => {
  return <h1 className={styles.bigblue}>Hello Car!</h1>;
}

export default Car;
```

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo

Importe el componente en su aplicación:

```
import React from 'react';
import ReactDOM from 'react-
dom/client';import Car from
'./Car.js';

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

Tema 3: Tema 3: Eventos, condicionales, listas, formularios, rutas y estilo