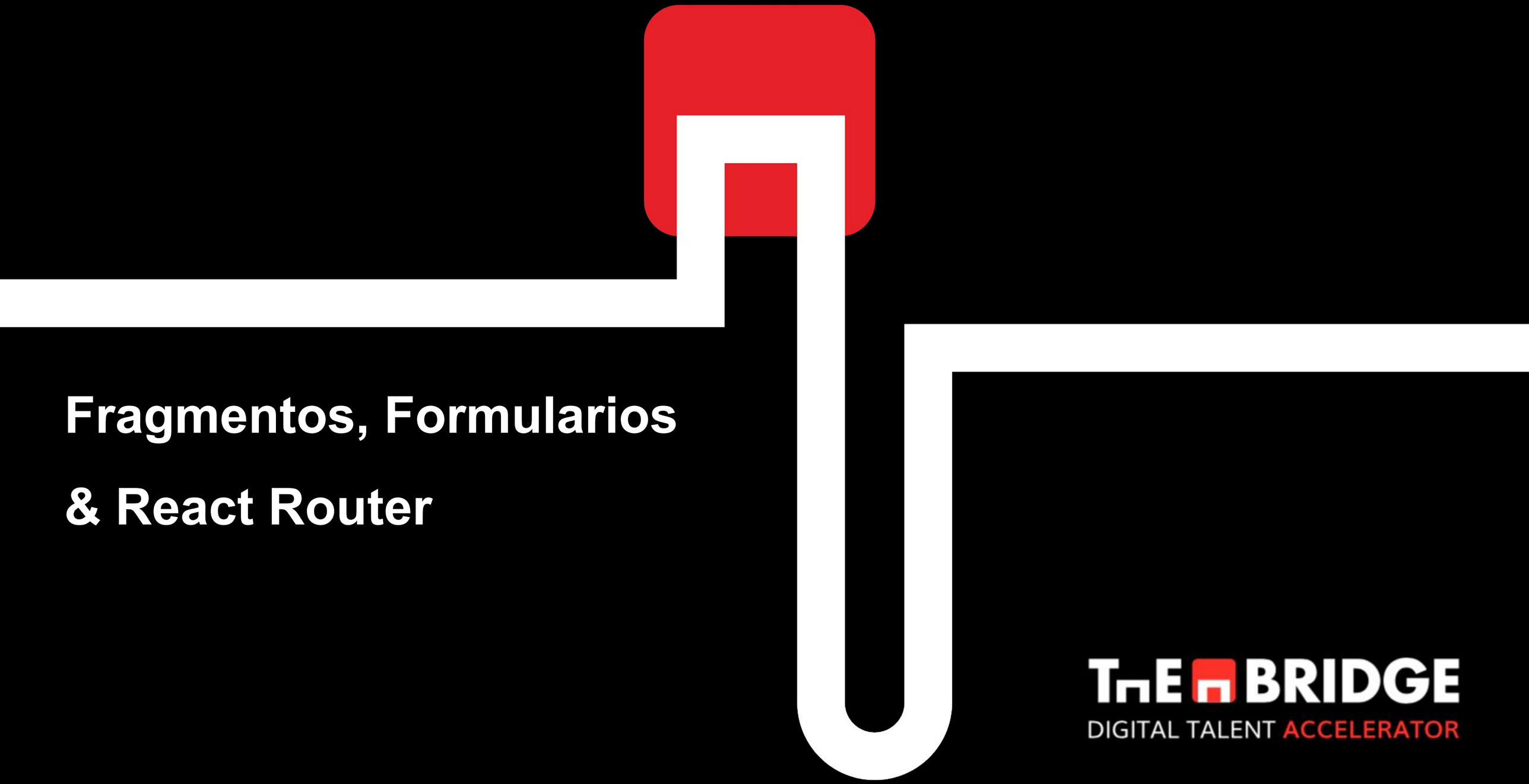
Fragmentos, Formularios & React Router

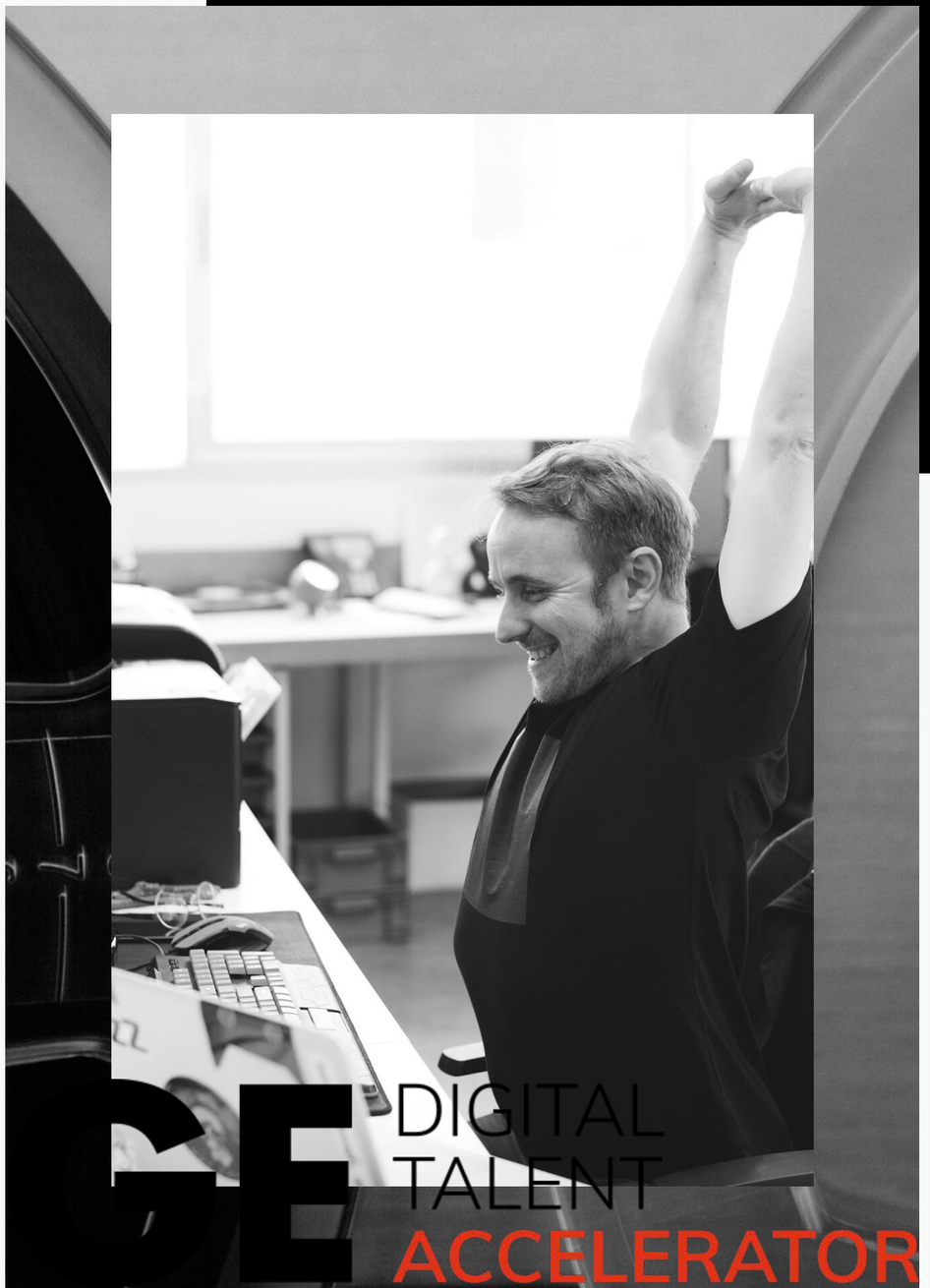


Índice

Fragmentos

Formularios

React Router



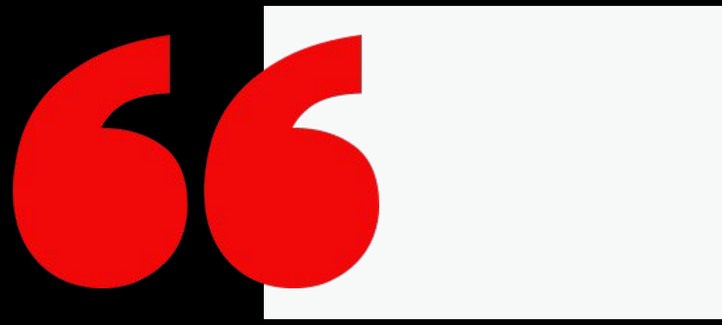
THE BRIDGE DIGITAL TALENT ACCELERATOR

Extensión

React Devtools

https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=es

Fragmentos



Fragmentos

Un patrón común en React es que un componente **devuelve múltiples elementos**.

Los **fragmentos** permiten agrupar una lista de elementos secundarios sin agregar nodos adicionales al DOM (ej. div).

```jsx
import React from "react";

const Home = () => {
  return (
    <React.Fragment>
      <span>Home</span>
    </React.Fragment>
  );
};

export default Home;
```

Sintaxis corta

Hay una sintaxis nueva y más corta que se puede usar para declarar fragmentos.

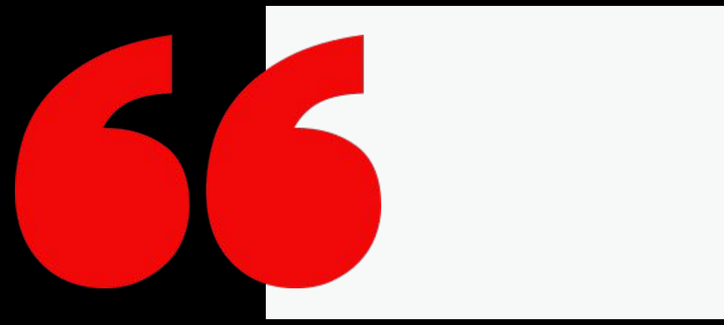
Se puede usar **<></>** de la misma manera que usaría cualquier otro elemento, excepto que **no admite claves ni atributos**.

```
import React from "react";

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

export default Home;
```

Formularios



Los **elementos de formulario HTML** funcionan un poco diferente de otros elementos DOM en React, porque los elementos de formulario naturalmente mantienen algún **estado interno**.

Pero en la mayoría de los casos, es conveniente tener una función de JavaScript que **maneje el envío del formulario** y tenga **acceso a los datos** que el usuario ingresó en el formulario.

State

Los **datos** que utilizaremos en nuestro formulario:

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

export const UserForm = () => {
  const [data, setData] = useState({
    name: "",
    email: "",
  });
  return <div>UserForm</div>;
};
```

Form

Creamos el siguiente formulario:

```
<form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="name"
        onChange={handleInputChange}
        name="name"
      />
      <input
        type="email"
        placeholder="email"
        onChange={handleInputChange}
        name="email"
      />
      <button type="submit">Enviar</button>
</form>
```

Evento onChange

Estará pendiente de los cambios que se registren en nuestro **input**, por lo tanto creamos una función para modificar nuestro **state**.

```
const handleInputChange = (event) => {
    console.log(event.target.name)
    console.log(event.target.value)

    setData({
      ...data,
      [event.target.name]: event.target.value,
    });
 };
```

Evento onSubmit

Ahora con los datos ya ingresados, podemos utilizar el evento **onSubmit** para procesar el formulario:

```
const handleSubmit = (event) => {
    event.preventDefault();
    console.log(`sending data… ${data.name} ${data.email}`);
 };
```

Form

Ahora en el formulario le añadimos el **atributo value** para definir el valor que tiene el **input**:

```
<input
    type="text"
    placeholder="name"
    value={data.name}
    onChange={handleInputChange}
    name="name"
  />
  <input
    type="email"
    placeholder="email"
    value={data.email}
    onChange={handleInputChange}
    name="email"
  />
```

Limpiando formulario

Para limpiar el formulario vamos a crear una constante **initialState** que tengan los campos vacíos. Después crearemos una función que **cambiará el estado de data** por el de **initialState**:

```
const initialState = {
    name: "",
    email: "",
  };

  const clearState = () => {
    setData({ ...initialState });
  };
```

Evento onSubmit

A la función **handleSubmit** que hemos creado antes le añadiremos la función **clearState** para que **limpie** el estado una vez se **envíe el formulario**:

```
const handleSubmit =  (event) => {
    event.preventDefault();
    console.log("sending data..." + data.name + " " + data.email);
    clearState();
 };
```

Nuestra primera validación

Definimos los siguientes estados:

```
const [btnDisabled,setBtnDisabled] = useState(true)
const [message,setMessage] = useState("")
```

handleInputChange

Modificamos nuestra función y le decimos que el campo **name** tiene que tener al menos 3 caracteres:

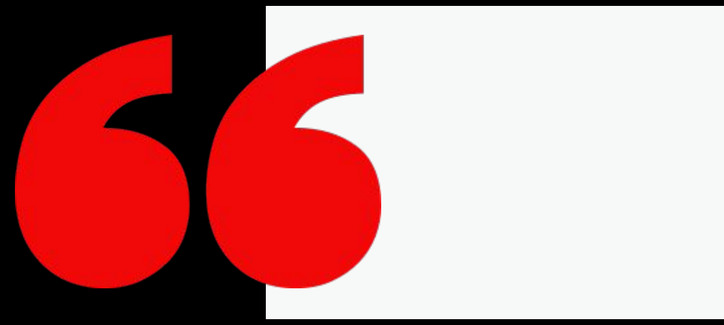
```
const handleInputChange = (event) => {
  if(data.name.length + 1 < 3){
      setMessage('Name must be at least 3 characters')
      setBtnDisabled(true)
    } else {
      setMessage(null)
      setBtnDisabled(false)
    }
    setData({ ...data, [event.target.name]:
event.target.value });
  };
```

Botón deshabilitado

Definimos el **botón** como **disabled** y debajo añadimo el **mensaje**:

```
. . .

  <button type="submit" disabled={btnDisabled}>Enviar</button>
  <p>{message}</p>
</form>
```



React Router

React Router es la **librería de enrutamiento estándar** para React.

Nos permite crear una **aplicación web de una sola página** con navegación sin que la página se actualice a medida que el usuario navega.

Instalación

React Router

```
$ npm install react-router-dom@6
```

Route

El componente de **Route** es quizás el componente más importante en **React Router** para comprender y aprender a usarlo bien. Su responsabilidad más básica es **representar el componente cuando su ruta coincida con la URL actual**.

```
<Route path="/" element={<Home />} />
```

Definiendo rutas

Por lo general, definiremos las rutas en el componente de la aplicación. Deberemos importar **BrowserRouter**, **Routes** y **Route** desde **react-router-dom**.

```
import { BrowserRouter, Route, Routes } from
"react-router-dom";

function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <Header />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/user" element={<UserForm />} />
        </Routes>
      </BrowserRouter>
    </div>
  );
}
```

Link

El **Link** proporciona una navegación **declarativa** y **accesible** alrededor de la aplicación. Es el equivalente a un elemento **<a>** pero en React es necesario usar **Link**, ya que un elemento **a** no redirigirá.

Por lo tanto, en React, si necesitamos cambiar una ruta, deberemos usar el componente **Link** y no un elemento **<a>**.

```
<Link to="/about">About</Link>
```

Creando el nav

Importamos el **Link** de **react-router-dom** y comenzamos a usarlo como ejemplo.

A medida que hace clic en los diferentes **<Link>**, el enrutador muestra la **<Ruta>** correspondiente.

```
import React from "react";
import "./Header.scss";
import { Link } from "react-router-dom";

const Header = () => {
  return (
    <nav className="header">
      <span>Header</span>
      <div>
        <span>
          <Link to="/">Home</Link>
        </span>
        <span>
          <Link to="/user">UserForm</Link>
        </span>
      </div>
    </nav>
  );
};

export default Header;
```

useNavigate

La mayoría de las veces, la **URL cambia en respuesta** al usuario haciendo **clic en un enlace.** Pero a veces tú, **el programador**, quieres cambiar la URL.

Un **caso de uso muy común al logearnos** será redirigidos.

En el ejemplo le decimos que una vez **clique** en el botón y **ejecute** la función **handleSubmit** al pasar 1 segundo **navegue a la ruta de home.**

```
. . .
import { useNavigate } from "react-router-dom";

const UserForm = () => {
  let navigate = useNavigate();

  const handleSubmit = (event) => {
    . . .
    setTimeout(() => {
      navigate("/");
    }, 1000);
  };
. . .
```

importamos useNavigate

inicializamos useNagivate

Le decimos que nos lleve a la ruta “/”