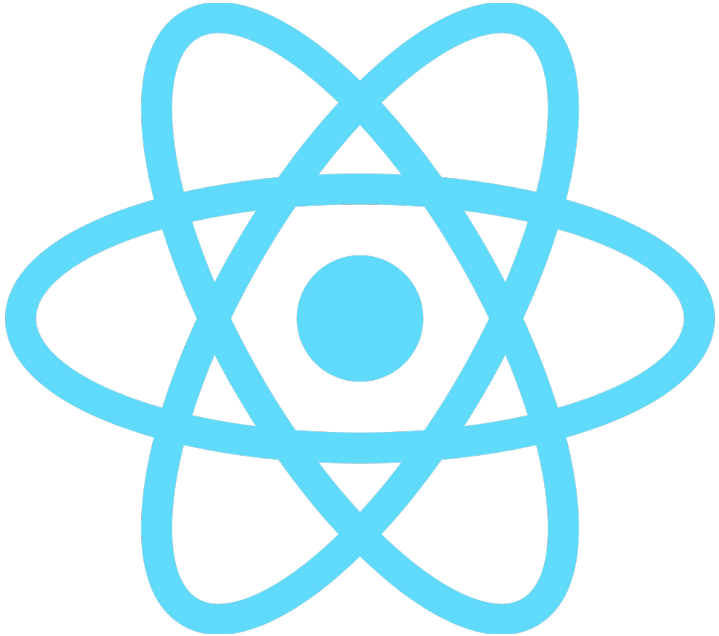
Clase 7

REACT



Temario

1. React
2. React Components and Props
3. Composing React Components and Stateful React Components
4. Events, conditional rendering
5. Lists and forms
6. React Router I
7. **React Router II**
8. Debugging and Testing

Rutas: Loaders

Con React Router, podemos definir una función en la propiedad **loader** de la ruta que es la que se encargará de obtener los datos y devolverlos. A continuación, dentro del componente renderizado (asignado a la propiedad **element**) accedemos a los datos con el hook **useLoaderData.** Nota: Este hook sólo funciona en componentes funcionales.

Sobre nuestra aplicación my-twelfth-app:

- En el archivo **TableUsers.js** vamos a exportar una función **loader** que se encargará de traer los datos de una api de test.

- En el componente **TableUsers** obtenemos los datos obtenidos por la función anterior usando el hook **useLoaderData** y los almacenamos en la variable **users.** En la inicialización del estado le pasamos los datos de **users.**
- Comprobamos que el funcionamiento sigue siendo el mismo

```
// src/TableUsers.js
export async function loader() {
  const url = "https://dummyjson.com/users";
  let usersApi = await fetch(url);
  usersApi = await usersApi.json();
  const users = usersApi.users.map((user) => {
    return {
      name: user.firstName,
      job: user.company.title };
  });
  return users;
}
```

```
// src/TableUsers.js
import { useState } from "react";
import { useLoaderData } from "react-router-dom";
import Table from "./Table";
import Form from "./Form";

const TableUsers = () => {
  const users = useLoaderData();
  const [people, setPeople] = useState(users);
```

Rutas: Actions

Los formularios HTML provocan navegación (cambios en la URL) igual que ocurría con los tags "**a**" que hemos sustituido por "**Link**" para hacer el renderizado en cliente. La única diferencia entre ellos es que además los formularios también pueden cambiar el método de la solicitud (GET, POST, PUT, etc).

Sin enrutamiento del lado del cliente, el navegador serializará los datos del formulario automáticamente y los enviará al servidor como el cuerpo de la solicitud para POST, y como URLSearchParams para GET. React Router hace lo mismo, excepto que en lugar de enviar la solicitud al servidor, utiliza el enrutamiento del lado del cliente y la envía a una acción de ruta.

Sobre nuestra aplicación my-twelfth-app:

- En el archivo src/routes/Form.js renombramos nuestra clase a **UserForm**
- En el componente **UserForm** eliminamos la línea this.props.handleSubmit(this.state) de la función **submitForm**
- Cambiamos el tag **form** por el componente **Form** de react router y asignamos el evento onSubmit a this.submitForm
- Eliminamos el evento **onClick** del botón submit y cambiamos a type="submit"

```
// src/routes/Form.js
import React, { Component } from "react";
import { Form } from "react-router-dom";

class UserForm extends Component {
  initialState = {…
  };
  state = this.initialState;
  handleChange = (event) => {…
  };
  submitForm = () => {
    // this.props.handleSubmit(this.state);/
    this.setState(this.initialState);
  };
  render() {
    const { name, job } = this.state;
    return (
      <Form method="post" id="user-form" onSubmit={this.submitForm}>
        <label htmlFor="name">Name: </label>
        <input
          type="text"…
          onChange={this.handleChange}
        />
        <label htmlFor="job">Job: </label>
        <input
          type="text"…
          onChange={this.handleChange}
        />
        {/* <input type="button" value="Submit"
                    onClick={this.submitForm} /> */}
        <input type="submit" value="Submit"/>
      </Form>
    );
  }
}

export default UserForm;
```

Rutas: Actions

En este punto podemos observar que nuestro navegador no se recarga al clicar en el botón Submit.

Sobre nuestra aplicación my-twelfth-app:

- En el archivo src/routes/TableUsers.js añadimos otra función que se va a encargar de gestionar los datos del formulario, la llamamos **action** pero podría llamarse de otra manera.
- En la función anterior añadimos como argumento destructurado "**request**" desde el que podremos obtener los datos del formulario.
- Añadimos una llamada al API fake para añadir un usuario.
- Finalmente devolvemos el usuario creado como respuesta de la función.
- En nuestro archivo src/router.js importamos la función **action** de **TableUsers** y la añadimos como atributo **action** de la ruta "people"

```
export async function action({ request }) {
  const formData = await request.formData();
  const fields = Object.fromEntries(formData);
  let user = await fetch("https://dummyjson.com/users/add", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      firstName: fields.name,
      lastName: "Test",
      age: 250,
      company: {
        title: fields.job,
      },
    }),
  });
  user = await user.json();
  return user;
}
```

```
import TableUsers, {
  loader as usersLoader,
  action as usersAction,
} from "./routes/TableUsers";
```

```
{
  path: "people",
  element: <TableUsers />,
  loader: usersLoader,
  action: usersAction,
},
```

Rutas: Actions

Ahora podemos dar de alta el usuario, no obstante, debido a que es una API fake, la creación es ficticia y no podemos ver que los datos se actualizan.

Para ello podemos hacer otra cosa, vamos a obtener los datos devueltos por la función **action** (usando el hook **useActionData**) y lo añadimos al principio de la tabla.

Sobre nuestra aplicación my-twelfth-app:

- En el archivo src/routes/TableUsers.js importamos el hook **useActionData** y **useEffect**
- En el componente TableUser declaramos una constante **userAdded** con el valor de useActionData
- El el hook **useEffect** hago una comprobación de la variable **userAdded** para añadirla a nuestra tabla.
- Para una mejor visualización en el renderizado, he movido el formulario arriba de la tabla.
- Modificamos la variable **title** para que muestre el número de personas.

```jsx
// src/routes/TableUsers.js
import React, { useState, useEffect } from "react";
import { useLoaderData, useActionData } from "react-router-dom";
import Table from "./Table";
import UserForm from "./UserForm";

const TableUsers = () => {
  const users = useLoaderData();
  const userAdded = useActionData();
  const [people, setPeople] = useState(users);

  useEffect(() => {
    if (userAdded) {
      const newUser = {
        id: userAdded.id,
        name: userAdded.firstName,
        job: userAdded.company.title,
      };
      setPeople((p) => [newUser, ...p]);
    }
  }, [userAdded]);
  const removePeople = (index) => {…
  };
  const handleSubmit = (character) => {…
  };
  const title = <h1>Nice People</h1>;
  return (
    <div className="container">
      <UserForm handleSubmit={handleSubmit} />
      <Table peopleData={people} removePeople={removePeople} title={title} />
    </div>
  );
};
```

Rutas: Params in loaders

Como bien sabéis, una ruta puede tener parámetros en la URL, como userId, contactId, etc etc.

Para acceder a estos parámetros de una función **loader,** podemos destructurar la variable **params** en la declaración de la función.

Por ejemplo, si tenemos la siguiente ruta ->

```
{
  path: "users/:userId",
  element: <User />,
  loader: userLoader,
  action: userAction,
},
```

Desde el **userLoader** podemos acceder al parámetro **userId** de la siguiente manera.

```
export async function userLoader({ params }) {
  console.log(params.userId);
  return null;
}
```

Rutas: Errores

Tal y como hemos visto en la definición de las rutas, se pueden declarar elementos para gestionar un error.

Los errores pueden renderizar un elemento genérico de error si se define en el padre, ó, puede renderizar un elemento por cada ruta.

Si se define a nivel de elemento padre, el renderizado no se hará en la misma posición que un hijo.

Sobre nuestra aplicación my-twelfth-app:

- Añadir un errorElement a la ruta "/"
- Si desde el loader de **TableUsers** provocamos una excepción, vemos que obtenemos un error en toda la página web.

- Para solucionarlo, podríamos crear un elemento genérico con tan sólo 2 atributos, errorElement y children con el resto de rutas.
- Probamos el resultado

```
path: "/",
element: <PublicLayout />,
errorElement: <div>Oops! There was an error.</div>,
children: [
```

```
// src/TableUsers.js
export async function loader() {
  throw new Error("oh an error!");
```

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <PublicLayout />,
    errorElement: <div>Oops! There was an error.</div>,
    children: [
      {
        errorElement: <div>Oops! There was an error in a child.</div>,
        children: [
          {
            path: "clock",
            element: <Clock />,
          },
          {
            path: "people",
            element: <TableUsers />,
            loader: usersLoader,
            action: usersAction,
          },
        ],
      },
    ],
  },
]);
```

Ejercicios

1. Replicar el código de clase:
    - App my-twelfth-app
2. Hacer la app de contactos siguiendo el tutorial oficial.

ALBAÑILESDIGITALES

Developing your new career

servicio navarro de empleo

das·nano

VeriDas