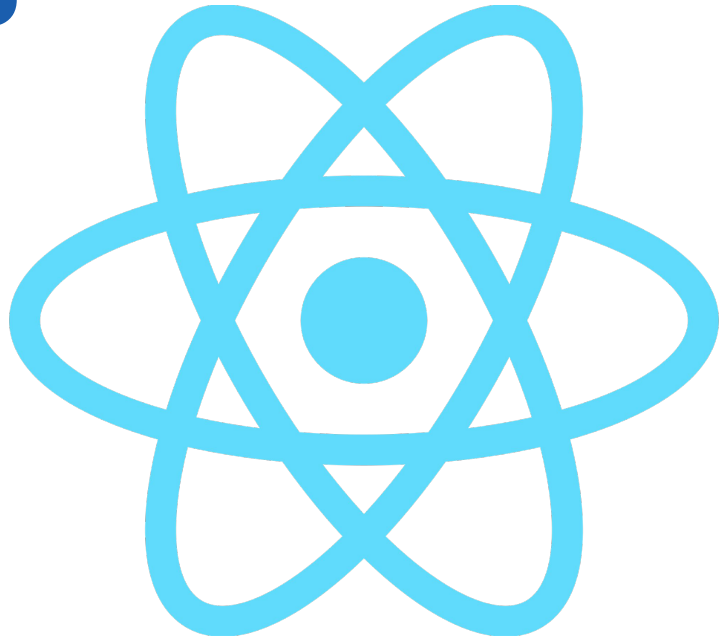
Clase 6

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

¿Qué es React Router?

Es una librería muy utilizada con React ya que facilita el proceso de definir las rutas de navegación dentro de nuestra aplicación.

Esta librería está disponible tanto para web como para móvil con React Native. Su cometido principal es ofrecer un enrutamiento dinámico gracias a los componentes, en otras palabras, ofrecer rutas que renderizan un componente. En este curso estudiaremos el router para navegador web **BrowserRouter** pero existen otros como **MemoryRouter, NativeRouter, StaticRouter.** (más info aquí)

Instalación:

```
npm install --save react-router-dom
```

Configuración:

- Creamos una aplicación **my-twelfth-app**
- En App.js importamos el **RouterProvider** crear nuestro router para web, **createBrowserRouter**
- Creamos el router la ruta principal que renderice un JSX
- Renderizamos el **RouterProvider** c como prop

```
import React from "react";
import {
  createBrowserRouter,
  RouterProvider } from "react-router-dom";
import "./App.css";

const router = createBrowserRouter([
  {
    path: "/",
    element: <div>Hello world!</div>,
  },
]);

const App = () => {
  return <RouterProvider router={router} />;
}

export default App;
```

Rutas: definiendo rutas

Hasta ahora hemos visto cómo renderizar un componente si accedemos a la ruta raíz "/", sin embargo, una aplicación real consta de varias rutas que muestran diferentes componentes según la interacción del usuario.

Definiendo rutas

Podemos crear nuestro router llamando la función **createBrowserRouter** con:

Opción 1: Un array de rutas como argumento, como en nuestro primer ejemplo, ó,

Opción 2: Componentes **Route** en una estructura JSX usando el método **createRoutesFromElements.**

```
// src/router.js
import {
    createBrowserRouter
} from "react-router-dom";

// OPTION 1
const router = createBrowserRouter([
  {
    path: "/",
    element: <div>Hello world!</div>,
  },
]);

export default router;
```

```
// src/router.js
import {
  createBrowserRouter,
  createRoutesFromElements,
  Route,
} from "react-router-dom";

// OPTION 2
const router = createBrowserRouter(
  createRoutesFromElements(
    <Route
      path="/"
      element={<div>Hello world!</div>}
      />
  )
);

export default router;
```

Rutas: Estructura

La definición de una ruta consta de los siguientes atributos:

- **path:** texto con la ruta en la que se renderizará este componente, por ejemplo: "/", "users", "students", etc.
- **element:** el componente a renderizar
- **errorElement:** componente que se renderizará en caso de error.
- **loader:** función a ejecutar para cargar los datos antes de renderizar el componente
- **action:** función a ejecutar en acciones de formulario
- **index:** booleano que indica que este componente es el componente inicial. Muy utilizado en rutas hijas para indicar el elemento hijo que se mostrará al acceder al padre. Se trata de una ruta especial (**index route**) que no requiere el atributo path.
- **children:** array de rutas que se denominan rutas hijas. El path de los hijos se concatena con el path del padre. Si un hijo no tiene configurado un **errorElement** se usará el elemento de error del padre.

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Root />,
    errorElement: <ErrorPage />,
    loader: rootLoader,
    action: rootAction,
    children: [
      {
        errorElement: <ErrorPage />,
        children: [
          { index: true, element: <Index /> },
          {
            path: "contacts/:contactId",
            element: <Contact />,
            loader: contactLoader,
            action: contactAction,
          },
          {
            path: "contacts/:contactId/edit",
            element: <EditContact />,
            loader: contactLoader,
            action: editAction,
          },
          {
            path: "contacts/:contactId/destroy",
            action: destroyAction,
            errorElement: <div>Oops! There was an error.</div>,
          },
        ],
      },
    ],
  },
]);
```

Rutas: Layout

Todas las páginas web tienen un esquema o composición de elementos (**Layout**), es decir, tienen un header, sidebar, footer, etc. ubicados en posiciones concretas según el propósito de la misma.

En nuestro caso vamos a crear una estructura inicial sobre la que pintar todos nuestros componentes.

Sobre nuestra aplicación my-twelfth-app:

- Reemplazamos el archivo **index.css** por el que se encuentra en los materiales de clase 5 dentro de Prácticas en moodle
- En **src** creamos una carpeta **routes** para almacenar nuestros componentes
- Dentro de src/routes creamos un componente **PublicLayout**
- En src creamos un archivo router.js para almacenar todas las rutas de nuestra aplicación
- En el archivo router.js importamos el componente **PublicLayout** y lo asignamos como elemento de la ruta "/"
- En App.js importamos **RouterProvider** y nuestro router y lo renderizamos.

```
// src/routes/publicLayout.js
export default function PublicLayout() {
  return (
    <>
      <div id="sidebar">
        <h1>React Router Veridas</h1>
        <nav>
          <ul>
            <li>
              <a href={`/clock`}>Clock</a>
            </li>
            <li>
              <a href={`/people`}>People</a>
            </li>
          </ul>
        </nav>
      </div>
      <div id="detail">[AQUI SE DEBEN RENDERIZAR LOS HIJOS]</div>
    </>
  );
}
```

```
import React from "react";
import { RouterProvider } from "react-router-dom";
import "./App.css";
import router from "./router";

const App = () => {
  return <RouterProvider router={router} />;
};

export default App;
```

Rutas: Layout

Si observamos el Layout, tenemos un listado con 2 rutas, una para mostrar el reloj y otra para mostrar un listado de personas. Por tanto, debemos crear las rutas para cada uno de ellos, como hijos de la ruta principal "/".

Sobre nuestra aplicación my-twelfth-app:

- Copiamos el componente **Clock** del ejercicio my-eighth-functional dentro de src/routes
- Copiamos los componentes **Table**, **Form** y **App** del ejercicio my-seventh-app dentro de src/routes. Al componente **App** lo renombramos como **TableUsers** (tanto el archivo como el nombre de la clase)
- Dentro de nuestro archivo router.js creamos las rutas hijas para clock y people importando los componentes correspondientes.

Si ahora clicamos sobre los links vemos que no se muestran los componentes. En un componente padre, se indica el lugar de renderizado de los hijos mediante el componente **<Outlet />**

- En nuestro **PublicLayout** cambiamos el texto [AQUI SE DEBEN RENDERIZAR LOS HIJOS] por **<Outlet />**

```
// src/router.js
import { createBrowserRouter } from "react-router-dom";
import PublicLayout from "./routes/publicLayout";
import Clock from "./routes/clock";
import TableUsers from "./routes/TableUsers";

const router = createBrowserRouter([
  {
    path: "/",
    element: <PublicLayout />,
    children: [
      {
        path: "clock",
        element: <Clock />,
      },
      {
        path: "people",
        element: <TableUsers />,
      },
    ],
  },
]);

export default router;
```

```
// src/routes/publicLayout.js
import { Outlet } from "react-router-dom";

export default function PublicLayout() {
  return (
    <>
      <div id="sidebar">…
      </div>
      <div id="detail"><Outlet /></div>
    </>
  );
}
```

Rutas: Client Side Routing

En este momento, si clicamos sobre cualquiera de las opciones el navegador se está recargando completamente, ya que está haciendo una solicitud del documento al servidor, es el comportamiento por defecto.

Client side routing permite que nuestra app actualice la URL sin solicitar otro documento al servidor, en su lugar, la aplicación renderizará inmediatamente la nueva UI. Esto lo podemos conseguir con el componente **Link** de react-router-dom en lugar del tag **a.**

Sobre nuestra aplicación my-twelfth-app:

- En nuestro componente **PublicLayout** cambiamos los links de html por el componente **Link**
- En nuestro componente **Clock,** añadimos como segundo argumento de **useEffect**, un array vacío, []

Ahora observamos que la URL cambia pero no se carga toda la página del navegador. Además, con el segundo argumento de useEffect, conseguimos que la función sólo se ejecute una vez.

```jsx
// src/routes/publicLayout.js
import { Outlet, Link } from "react-router-dom";

export default function PublicLayout() {
  return (
    <>
      <div id="sidebar">
        <h1>React Router Veridas</h1>
        <nav>
          <ul>
            <li>
              <Link to={`/clock`}>Clock</Link>
            </li>
            <li>
              <Link to={`/people`}>People</Link>
            </li>
          </ul>
        </nav>
      </div>
      <div id="detail">
        <Outlet />
      </div>
    </>
  );
}
```

```jsx
useEffect(() => {
  const timerID = setInterval(() => tick(), 1000);
  return function cleanup() {
      clearInterval(timerID);
  };
}, []);
```

Ejercicios

1. Replicar el código de clase:
    - App my-twelfth-app

ALBAÑILESDIGITALES

Developing your new career

servicio navarro de empleo

das·nano

VeriDas