Actividad evaluable UT2A2 (AE1.3) – Login y enrutamiento

RA1, RA3, RA4

En esta actividad continuaremos con el proyecto de UT2. En esta ocasión elaborarán la página Login de la aplicación. Además, aprenderán enrutamiento, con lo cual podrán navegar por las diferentes páginas de la aplicación.

¿Qué tengo que entregar? (Requisito)

- Enlace al github en la rama login.
- Memoria en formato PDF con capturas de pantalla y breve explicación de lo especificado en verde en este documento. La memoria debe tener portada e índice, el texto debe tener alineación justificada, la expresión escrita debe ser correcta. Numera las páginas de la memoria.
- El proyecto será descargado por la profesora desde el github: es requisito indispensable que la aplicación se pueda ejecutar en cualquier equipo para poder ser corregida.

¿Cómo identifico mi trabajo? (Requisito)

- Debes poner tu nombre y apellidos en el fichero README.md
- Debes nombrar el PDF como sigue:

PrimerApellido_SegundoApellido_Nombre_UT2A2

Desarrollo de la actividad

PARTE I: Login

1. Crea una nueva rama en tu proyecto: te sitúas en la carpeta proyecto*Tusiniciales* y ahí escribes el siguiente comando:

git checkout -b login

Con esto estás creando una rama nueva llamada login y además, te estás cambiando a esa rama.

2. Crea la página de login de tu aplicación:

Debe quedar como la siguiente, pero con tu tema personalizado. Puedes cambiar el icono del candado por otro igualmente intuitivo. También puedes cambiar la palabra Acceder por cualquier otra que indique lo mismo (Conectar, Ingresar, etc.)

Te recuerdo aquí la estructura básica de un formulario con componentes MUI:

```
<Box
  component = 'form'
  onSubmit={handleSubmit}
>
  //Aquí irían los campos necesarios para el formulario
  //...
  //…
  //Botón del formulario es del tipo submit: cuando picas en el botón responde al evento onSubmit
  <Button variant='contained' fullWidth type='submit'>Acceder</Button>
  //…
  //…
</Box>
```

Recuerda que en la función handleSubmit tenemos que escribir dentro **e.preventDefault()** para evitar que el formulario envíe los datos de la forma que tiene predeterminada y recargue la página.

```
const handleSubmit = (e:any) => {
//Para que no mande el formulario, sino que haga lo que yo le diga.
e.preventDefault()
//Aquí iría el código que quiero que se ejecute
}
```

Ten en cuenta que el TextField de la contraseña debe ser de `type='password'` para que no se vea cuando escribamos la contraseña.

Los campos Usuario y Contraseña son **requeridos**. De esta forma, la aplicación nos avisa que, aunque piquemos en el botón Acceder, no tendrá efecto porque nos falta rellenar el/los campos.

3. Crea la lógica de funcionamiento del login (la que tendrá por ahora)

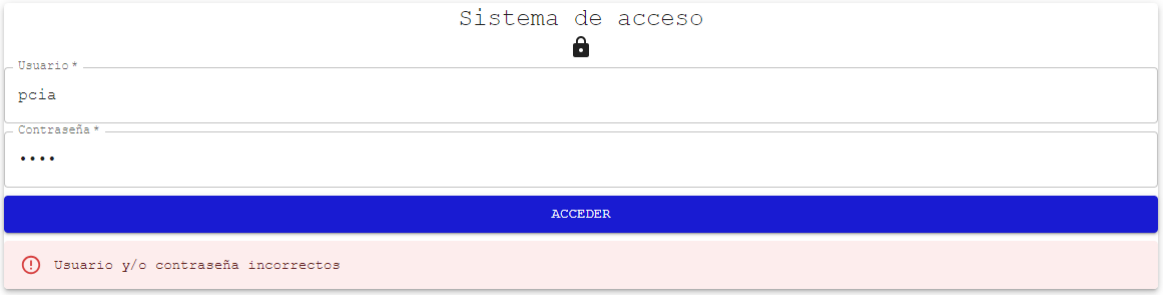
En este punto del proyecto la lógica de funcionamiento del login va a ser la siguiente:

En un login, tenemos que comprobar que el usuario y contraseña que nos pasan a través del formulario corresponden a los que están almacenados en la base de datos. En nuestro caso, vamos a compararlos con dos variables que definamos en el código. En mi caso son:

```
const bduser = 'patricia'
const bdpasswd = '1234'
```

Así que cuando el usuario de la aplicación escriba sus datos de acceso en el formulario tendremos que compararlos con los que hemos definido nosotros en el programa.
Si coinciden mostraremos un componente <Alert> de MUI con un mensaje similar al siguiente:

Si no coinciden mostraremos un componente <Alert> con un mensaje similar al siguiente:



Documentación del Alert de MUI: https://mui.com/material-ui/react-alert/

→ Captura de pantalla de la aplicación funcionando cuando escribes el usuario y contraseña correctos. Se debe ver a la vez la consola con el usuario y contraseña que has puesto.
→ Captura de pantalla de la aplicación funcionando cuando el usuario y/o contraseña son incorrectos. Se debe ver a la vez la consola con el usuario y contraseña que has puesto.
→ Explicación de la parte del código (con captura de pantalla del código correspondiente) en la que compruebas el usuario y contraseña y cómo renderizas el Alert en el caso de usuario y contraseña correctas y en el caso de que alguno de los dos no sea correcto.

PARTE II: Enrutamiento

El enrutamiento consiste en establecer los mecanismos necesarios para navegar entre las páginas de una aplicación.

En nuestra aplicación tendremos las siguiente rutas (por ahora):

- /: corresponde a la página raíz, en el navegador es la siguiente URL http://localhost:5173/. Aquí se renderiza el componente <Login />
- /home: corresponde en el navegador a la URL http://localhost:5173/home. Aquí se renderizará el componente <Home />
- /reports: corresponde en el navegador a a la URL http://localhost:5173/reports . Aquí se renderizará el componente <Reports />

El componente <Login /> ya lo tienes creado.

1. Creación de componente <Home /> y componente <Reports />

Crea el componente <Home /> en src/pages/Home.tsx y el componente <Reports /> en src/pages/Reports.tsx. Dentro de cada uno de ellos pon un <Typography> que diga: Página Home de *tu nombre* y Página Reports *de tu nombre* respectivamente.

2. Instalación de la librería react-router-dom y sus tipos

La librería react-router-dom nos permite realizar enrutamiento en React. Puesto que **el enrutamiento lo haremos en el frontend** debemos **instalar** dicha **librería dentro de la carpeta frontend** de nuestro proyecto:

npm install react-router-dom

Además, instalaremos los tipos de la librería:

npm install --save @types/react-router-dom

Vemos en el package.json que se han instalado:

```
"@types/react-router-dom": "^5.3.3",
"react": "^18.3.1",
"react-dom": "^18.3.1",
"react-router-dom": "^6.27.0"
```

La documentación oficial de la librería react-router-dom la pueden encontrar aquí: https://reactrouter.com/en/main . Nosotros nos basaremos en el siguiente ejemplo, aunque está ajustado para nuestra aplicación: https://reactrouter.com/en/main/start/tutorial#index-routes

3. Creación del enrutamiento en el fichero App.tsx

En el fichero App.tsx del frontend crearemos la estructura de navegación, teniendo en cuenta que en nuestro proyecto, la página inicial / montará el componente <Login/>, la página /home montará el componente <Home/> y la página /reports montará el componente <Reports/>.

La estructura de navegación se crea usando la función `createBrowserRouter`, que es una función propia de la librería react-router-dom. Además, debemos especificar en el return de App.tsx que usaremos el enrutamiento. Eso lo haremos con el componente **<RouterProvider />**, que es un componente de la librería react-router.dom.

El código que tienen hasta ahora en App.tsx es el siguiente:

```
import React from 'react'
import './App.css';
import Login from './pages/Login'

function App() {
  return (
    <Login />
  )
}
export default App;
```

Una vez realizado el enrutamiento en el fichero App.tsx se quedará como sigue (OJO, está incompleto, lo rellenaremos en clase):

```
import './App.css'
import Login from './pages/Login'
import Home from './pages/Home'
//AQUÍ IMPORTAMOS Reports → hacerlo
import {createBrowserRouter, RouterProvider} from 'react-router-dom'
```

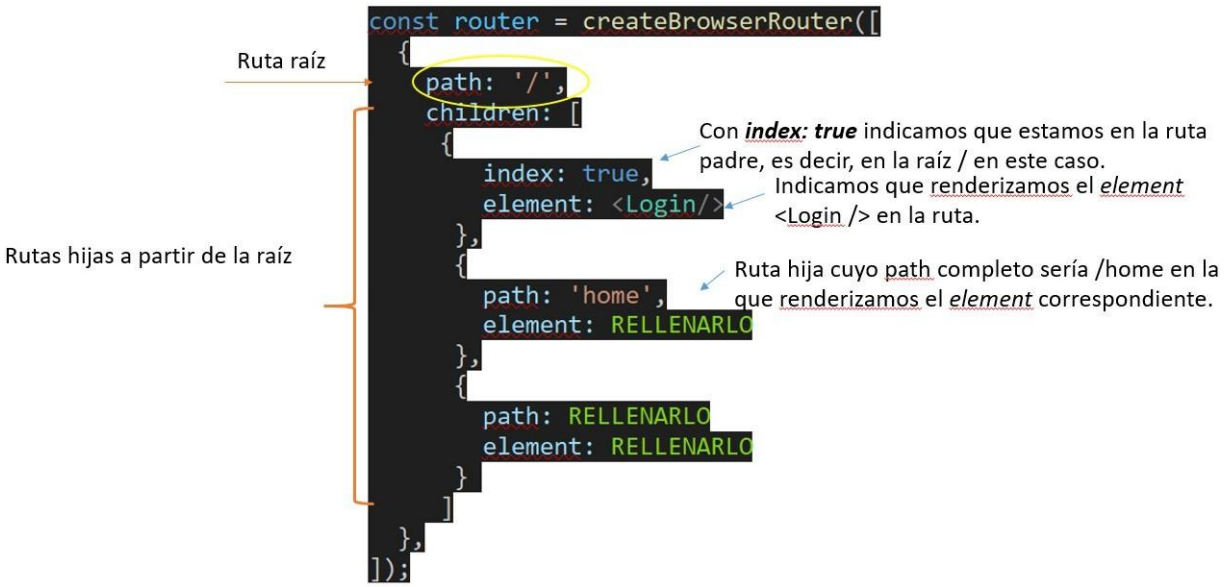
```
const router = createBrowserRouter([
  {
    path: '/',
    children: [
     {
        index: true,
        element: <Login/>
      },
      {
        path: 'home',
        element: RELLENARLO
      },
      {
        path: RELLENARLO
        element: RELLENARLO
      }
    ]
  },
]);

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

export default App
```

En resumen, creamos la estructura de navegación con createBrowserRoute() y luego pasamos el objeto router como una prop al componente <RouterProvider />.

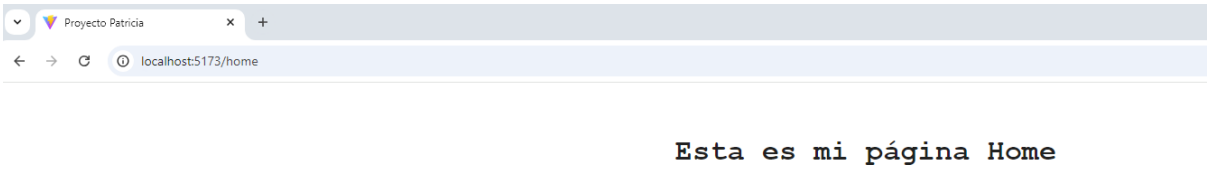
→ Captura de pantalla de la estructura de navegación explicándola.



4. Probar la navegación cambiando las URL

Prueba las diferentes URL y comprueba que realmente enruta a las páginas correspondientes.

Por ejemplo, si pongo en la URL http://localhost:5173/home va a mi página home:



→ Capturas de pantalla donde se vean las URL de las diferentes páginas.

5. ¿Cuándo y cómo navegamos a la página http://localhost:5173/home?

Cuando piquemos en ACCEDER y además confirmemos que los datos introducidos en el frontend (el usuario y la contraseña) pertenecen a un usuario válido. Es ahí donde tenemos que dar la orden de navegar a la página /home. Por lo tanto, lo haremos dentro del fichero Login.tsx.

Para navegar entre páginas hay varias formas, pero la que nos interesa aquí es mediante el uso del *hook* useNavigate.

Primero lo importamos: `import { useNavigate } from 'react-router-dom'`
Luego, definimos una variable que será el hook useNavigate. Como todos los hooks, será lo primero que se defina justo después del comienzo de la function Login():

```
const navigate = useNavigate()
```

En el punto donde queramos navegar a otra página debemos escribir: **navigate(*ruta*)**.

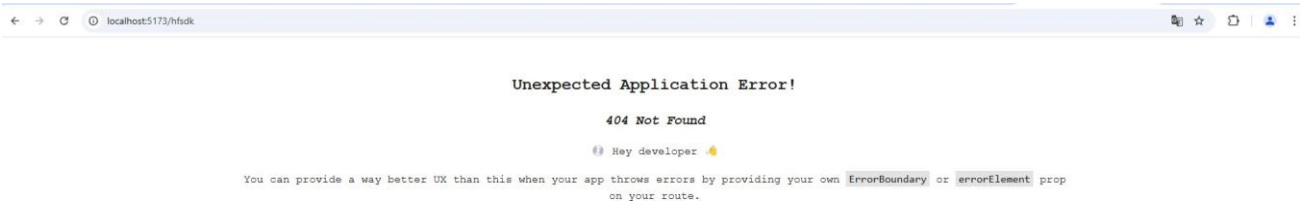
Ejemplos:
si quisiese navegar a la página /biblioteca pondría: navigate('/biblioteca')
si quisiese navegar a la página /results pondría: navigate('/results')

Implementa en el Login la navegación a la página Home cuando el usuario pica en ACCEDER y ha puesto el nombre de usuario y contraseña correctos. Fíjate que ahora no hace falta poner el Alert de acceso concedido.

→ Captura de pantalla de la parte del código donde navegas a /home explicando por qué lo pusiste ahí y cómo funciona.

6. Añadir una página de error al enrutamiento

¿Qué pasaría si te equivocas al poner la URL? Esto es lo que sale por defecto:

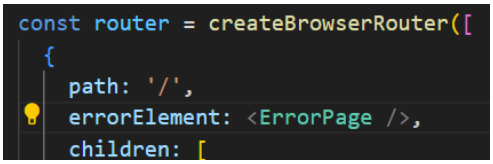


Para que no salga esa página tan fea, podríamos crear nosotros una propia y manejar el error.

Para ello tenemos que hacer dos cosas:

1) Crear la página de error: como siempre la creas en src/pages/
   Fíjate en el siguiente ejemplo para hacer tu página de error:
   https://reactrouter.com/en/main/start/tutorial#handling-not-found-errors
   Debe aparecer el motivo del error, pero todo lo demás lo puedes personalizar.

2) Añadirla al enrutamiento de la siguiente forma:



   Añades un errorElement y ahí renderizas la página de error.

Una vez lo tengas hecho:

→ Captura de pantalla del código de tu página de error explicando lo que hace, incluido el *hook* que usas.
→ Captura de pantalla de tu página de error (aplicación funcionando) cuando pones una ruta equivocada.

PUNTUACIÓN

Entrega	Nota MÁXIMA
1) El ejercicio se entrega en plazo y resuelto correctamente.	10
2) Igual que el 1) pero fuera de plazo. Máximo 24 horas	6
3) No se admiten entregas fuera de plazo cuando se han pasado más de 24 horas de la fecha límite de entrega.	0
4) Si no se cumplen las especificaciones de entrega la tarea irá directamente a recuperación.	0

En la tabla se indican las notas máximas en cada situación.