Configuración de Rutas, Páginas y Axios.

Parte – 7

Siguiendo desde el punto de las fotocopias anteriores, vamos a instalar una extensión en Visual Studio Code, para que nos facilite el autocomplimentado de las clases que utiliza Tailwindcss.

TailWind CSS IntelliSense.

Si en vez de trabajar con el Framework de tailwind trabajamos con el de Bootstrap, podríamos instalar la extensión de Bootstrap con el mismo objetivo.

Bootstrap 5 Quick Snippets.

Con esto, tenemos todas las herramientas que nos hacen falta para empezar a desarrollar nuestra parte del FrontEnd. Para comprobar que TailWind funciona correctamente, nos vamos a desplazar a nuestro fichero App.js y vamos a introducir las siguientes etiquetas entre

```
<div>
        <h1 classname=”text-4xl font-bold”>
                Segundo de DAM.
        </h1>
</div>
```

Las classes de estilos de text-4xl y de font-bold, son de TailWind y como veréis, tampoco hay mucha diferencia con las de Bootstrap.

Ya que estamos con las clases de estilos, vamos a desplazarnos a nuestro fichero index.css y vamos a crear un estilo de la siguiente forma (más que nada para ir dando forma a nuestro front).

```
body{
        background: #202020;
        color: white;
}
```

Como entendéis, seguro, podéis utilizar los estilos propios y/o ajenos de la forma que queráis, no son excluyentes.

IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891

Rutas:

Si os acordáis, a la hora de crear nuestro backend, hemos utilizado express para crear diferentes rutas a las que asociar diferentes funciones. De tal manera que nos encontramos la ruta de login, register, task, etc…que ejecutaban funciones para logearse, registrarse, añadir tareas al usuario registrado, etc., respectivamente.

Nuestra forma de trabajar, en el lado del Frontend, no va a ser diferente. Es decir, vamos a aplicar la lógica de hacer varias rutas a las que en este caso, vamos a asociar diferentes páginas.

Para hacer esto, vamos a utilizar un módulo de React que es el "react router dom" que nos va a proporcionar las herramientas necesarias para navegar, crear un conjunto de rutas y definir por separado cada una de las rutas.

Es interesante que veáis las diferencias que hay entre trabajar con React o con Angular. Angular es una herramienta más potente que React y que se va a utilizar en proyectos más complejos del que estamos creando. Angular además está creada y se trabaja sobre ella con TypeScript, que es un lenguaje que engloba a javascript, es decir, cualquier proyecto que esté creado en javascript, está validado por typescript y podrá ser fácilmente incorporado a él.

React es una librería creada por Facebook y que se dedica, únicamente a crear interfaces web, que es lo que nosotros buscamos aquí.

Después de este inciso, volvemos al "react router dom". Podéis entrar en su página web y ver como se trabaja y que es lo que nos permite hacer:

https://reactrouter.com/en/main

En la parte de "Tutorial", tenéis descrito el como funciona.

Lo que hace es crear un contenedor con la etiqueta <BrowserRoute> en el que vamos a insertar los diferentes grupos de rutas <Routes> especificando, una a una <Route> a través de su path y del elemento "página" que queremos que se carge.

Lo primero que hacemos es cargar, como se nos indica desde su página, la librería de "react-router-dom": **Es importante que recordéis que hay que instalar todas estas dependencias dentro de la carpeta del cliente, para que queden registradas en el package.json del cliente.**

npm i react-router-dom

Una vez cargada esta librería, vamos a volver a lanzar nuestro frontend:

npm run dev

IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891

Una vez hecho todo esto, nos vamos a desplazar a nuestro fichero App.jsx, que es desde el que vamos a importar y configurar todo lo necesario para nuestras rutas.

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

function App(){

    return (
        <BrowserRouter>
            <Routes>
                <Route path='/' element={<h1> Home Page </h1>} />
                <Route path='/login' element={<h1> Login </h1>} />
            <Route path='/register' element={<h1> Register</h1>} />
            </Routes>
        </BrowserRouter>

    )

}

export default App
```

Si todo es correcto y temenos el servidor del frontend arrancado, cada vez que entramos en cada una de esas rutas, se cargarán los elementos <h1> que hemos asociado a cada una de ellas.

Una vez aquí y comprobado que nos funciona todo lo anterior, empezamos a crear las diferentes interfaces que vamos a asociar a nuestras rutas. Lo primero que vamos a hacer es crear una carpeta que vamos a llamar **"pages"** en la que vamos a ir guardando todos los ficheros .jsx, que van a lanzar las diferentes interfaces de usuario.

Dentro de esta carpeta, creamos un fichero que vamos a llamar **RegisterPage.jsx** y que vamos a asociar a nuestra ruta '**/register**'.

Cuando edito el fichero voy a escribir rfce y al pulsar intro, nos debería de generar el DOM de ese documento. Fijaros que importa la librería de react, esto lo podéis eliminar. Para la interface de usuario que voy a asociar con la ruta 'login', hago exactamente lo mismo pero en un fichero denominado LoginPage.jsx.

Como podéis observar, al crear la estructura del documento, nos ha creado una función con el mismo nombre que el nombre del fichero .jsx. Si esto es así, tenemos dentro del fichero RegisterPage.jsx una función denominada RegisterPage() y dentro del fichero LoginPage.jsx tenemos otra función que es la de LoginPage(). Estas dos funciones, son exportadas al final de cada uno de los documentos.

IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891

Como lo que queremos es asociar esas funciones (en las que se generan las diferentes interfaces de usuario) a la ruta correspondiente, voy a ir al fichero App.jsx y las vamos a importar:

import RegisterPage from ‘./pages/RegisterPage’
import LoginPage from ‘./pages/LoginPage’

Y, en la línea en la que declaramos la ruta, la añadimos dentro del atributo “element={}”.

<Route path=’/register’ element={<RegisterPage />} />
<Route path=’/login’ element={<LoginPage />} />

Ahora ya lo tenemos asociado. Nos interesa también el poder controlar de alguna manera, los datos que estamos que vamos a enviar a nuestro Backend. Recordar que en el backend hemos creado con modelos de datos en los que decíamos si tenían que ser requeridos (required) el tamaño máximo o mínimo, etc. En la parte del frontend, vamos a hacer lo mismo. Para llevar a cabo este control de datos, vamos a utilizar el módulo de react de “react-hook-form”. Además de hacer este control de datos, nos permite o facilita el envío de datos al servidor.

https://react-hook-form.com

```
return (
  /* "handleSubmit" will validate your inputs before invoking "onSubmit" */
  <form onSubmit={handleSubmit(onSubmit)}>
    {/* register your input into the hook by invoking the "register" function */}
    <input defaultValue="test" {...register("example")} />

    {/* include validation with required or other standard HTML validation rules */}
    <input {...register("exampleRequired", { required: true })} />
    {/* errors will return when field validation fails  */}
    {errors.exampleRequired && <span>This field is required</span>}

    <input type="submit" />
  </form>
)
```

react-hook-form utiliza la propiedad register{} para controlar la información. Fijaros como cuando creo la etiqueta <form> en el atributo onSubmit={} le pasamos el handleSubmit que es parte del react-hook-form y es el que nos va a permitir conectar con nuestro backend.

El fichero RegisterPage.jsx nos va a quedar de la siguiente manera:

```
import { useForm } from 'react-hook-form';

function RegisterPage() {
```


IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891

```
    const { register, handleSubmit } = useForm();


    return (
        <div className="bg-zinc-800 max-w-md p-10 rounded-md">
            <form onSubmit={handleSubmit((values) => {
                    console.log(values);
             })}>
                <input type="text" {...register('username', { required: true
})}
                    className='w-full bg-zinc-700 text-white px-4 py-2
rounded-md my-2'
                    placeholder='User Name'
                />
                <input type="email" {...register('email', { required: true })}
                    className='w-full bg-zinc-700 text-white px-4 py-2
rounded-md my-2'
                    placeholder='E-mail'
                />
                <input type="password" {...register('password', { required:
true })}
                    className='w-full bg-zinc-700 text-white px-4 py-2
rounded-md my-2'
                    placeholder='Password'
                />
                <button type='submit' className="bg-slate-600 text-white
rounded-md py-2 px-4 my-2">
                    Registrar
                </button>
            </form>
        </div>
    )
}

export default RegisterPage
```

Para hacer la comunicación con nuestro servidor o backend, vamos a utilizar “axios”. Esta librería nos permite hacer llamadas API REST (backend capaz de gestionar información en ambas direcciones en formato .json) con retorno json.

Como siempre, instalamos “axios”
npm i axios

IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891