Desarrollo de Software: Frontend

Objetivo: crear una Aplicación frontend con vistas en Html, Boostrap y código javascript con React, que consumirá las webapi de nuestro backend: dds-backend-2025. Nuestra aplicación estará compuesta por un menú que nos permitirá navegar entre una página de inicio, una página de consulta sobre la tabla Categorias y una página que nos permitirá realizar un ABMC sobre la tabla Articulos.

- Version final del proyecto backend: https://labsys.frc.utn.edu.ar/dds-backend-2025/
- Version final del proyecto con frontend: https://labsys.frc.utn.edu.ar/dds-frontend-2025/

  Nota: mientras labsys no esté disponible puede verse back y front juntos en:
  https://pymes2025.azurewebsites.net/

- Requisitos tener instalado:
  - Visual studio Code
  - Node.js
  - Todos los comandos se ejecutan desde una consola de `git bash`

Etapa 1
Proyecto básico

Creación del proyecto: Ubicándonos en la carpeta que contendrá nuestro proyecto, por ej c:/users/miusuario, desde la consola ejecutamos:

npm create vite@latest dds-frontend (si no le funciona pruebe con npx en lugar de npm)

y luego elegimos nuestro framework: "react" y nuestro lenguaje: "javascript"

- **Observe:**
  - que si no está instalada la librería vite, le solicitará instalarla
  - que este comando genera una carpeta y varias subcarpetas con una serie de archivos que constituyen la interface de una nueva aplicación basada en react
  - antes de ejecutar la primera vez hay que instalar las dependencias (npm i)
  - para ejecutar la aplicación en desarrollo debemos usar el script dev: el comando seria npm run dev

Para verificar la funcionalidad la plantilla inicial del proyecto recién creado, nos ubicamos dentro de la carpeta creada y ejecutamos el siguiente comando de consola:
npm i
`npm run dev`

Observe:

- el comando anterior ejecutará el proyecto en localhost:5173, mostrando en la terminal un link para poder abrir desde la aplicación en el explorador
- podemos detener nuestro servidor de aplicación node/react, estando ubicados en la terminal desde donde iniciamos el proyecto y pulsando Ctrl+C o cerrando la misma.
- si cerramos el explorador y no detuvimos la aplicación (punto anterior), nuestra aplicación seguirá ejecutándose y podemos volver a verla en el explorador con la url por defecto

Mediante Visual Studio Code, vamos a cambiar la pantalla inicial de nuestro proyecto, dentro de los archivos generados, buscamos src/App.jsx que es el que proporciona la interface html inicial, y reemplazamos todo su código por el siguiente:

```
function App() {
  return (
```

```
        <h1>Bienvenidos a Desarrollo de Software!</h1>
        );
    }
    export default App;
```

Grabe los cambios y si la aplicación estaba corriendo, verá como se ha actualizado la salida en el explorador, caso contrario en la ventana de consola vuelva a ejecutarla con el comando:

```
npm run dev
```

Observe:

- la pantalla tiene un letras blancas sobre fondo negro debido a los archivo Index.css y App.css, que tienen codigo css que vienen en la plantilla de ejemplo, si borra el contenido de ambos archivos y graba el proyecto ,podra ver como cambia el aspecto del diseño. Para continuar con el ejercicio borre sus contenidos.

Primer componente: Inicio

- Nuevamente eliminamos todo el contenido del archivo App.jsx y lo reemplazamos por el código de nuestro primer componente:

```
    function App() {
      return (
                              <div    className="mt-4    p-5    rounded"
style={{backgroundColor:"lightgray"}} >
            <h1>Pymes 2025</h1>
                  <p>Este ejemplo está desarrollado con las siguientes
tecnologías:</p>
            <p>
               Backend: NodeJs, Express , WebApiRest, Swagger, Sequelize,
Sqlite y Javascript.
            </p>
```

```
              <p>
                   Frontend: Single Page Aplication, HTML, CSS, Bootstrap,
NodeJs,
                Javascript y React.
              </p>
              <button className="btn btn-lg btn-primary">
                <i className="fa fa-search"> </i>
                Ver Categorias
              </button>
            </div>
        );
      }
      export default App;
```

Observe:

- que hemos usado clases de bootstrap e iconos en nuestro html, por lo cual necesitaremos dichas librerías..
- que hemos usado propiedades de css y en react style tiene una sintaxis especial mediante un objeto de javascript.

Agregamos al proyecto las librerías de Bootstrap y sus dependencias

```
npm install bootstrap
```

Agregamos al proyecto la librería de íconos Font-Awesome (version 6.20)

```
npm install @fortawesome/fontawesome-free
```

Nota: todos los paquetes/librerías deben instalarse en la terminal, estando ubicados en la carpeta raíz del proyecto.

Para que las librerías recién instaladas, se carguen en nuestro proyecto y podamos hacer uso de sus funcionalidades, debemos importarlas al mismo, lo que haremos modificando el archivo src/main.jsx (o index.jsx segun corresponda) agregando las siguientes líneas de código al inicio del mismo:

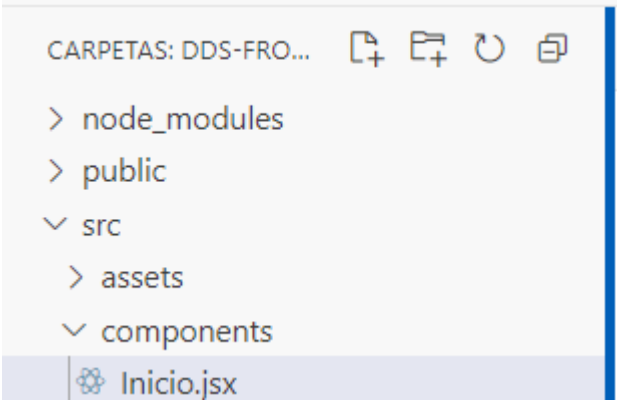
```
import 'bootstrap/dist/css/bootstrap.min.css'
import 'bootstrap/dist/js/bootstrap.min.js'
import '@fortawesome/fontawesome-free/css/all.min.css'
```

Ejecutemos la aplicación, y verificamos si nuestra salida html tiene aplicadas las clases de bootstrap y puede verse el icono utilizado. El resultado seria al similar a este:

Hasta aquí hemos usado el componente raíz (o principal) de la aplicación: App.jsx para nuestra página de inicio, pero en realidad lo que tenemos que hacer es crear inicialmente al menos un componente específico para cada página de nuestra aplicación, para luego ir reconociendo interface/código con sus respectivas responsabilidad y que también podremos reutilizar a partir del cual generamos nuevos componentes.

Vamos a crear un nuevo componente para nuestra página de inicio y para tener un mejor orden, crearemos una nueva carpeta denominada “components” (será hija de /src)

- en dicha carpeta, creamos el archivo /src/components/Inicio.jsx (respete la mayúscula inicial del nombre de este archivo)



- copiamos el código desde App.jsx a Inicio.js, reemplazando el nombre de la función "App" por "Inicio", con lo que el código nos quedaría así:

```
      function Inicio() {
        return (
             <div  className="mt-4  p-5  rounded"  style={{  backgroundColor:
"lightgray" }}>
            <h1>Pymes 2025</h1>
                 <p>Este  ejemplo  está  desarrollado  con  las  siguientes
tecnologías:</p>
            <p>
                Backend: NodeJs, Express , WebApiRest, Swagger, Sequelize,
Sqlite y Javascript.
            </p>
            <p>
                 Frontend: Single Page Application, HTML, CSS, Bootstrap,
Javascript, NodeJs y React.
            </p>
            <button className="btn btn-lg btn-primary">
              <i className="fa fa-search"> </i>
              Ver Categorias
            </button>
          </div>
```

```
      );
    }
    export { Inicio };
```

Observe:

- que siempre los archivos de componentes de react deben iniciar con mayúsculas

Ahora modificamos el App.jsx para que muestre el componente Inicio, para lo cual reemplazamos su código con el siguiente:

```
    import {Inicio} from "./components/Inicio";
    function App() {
    return (
        <>
            <Inicio/>
        </>
    );
    }
    export default App;
```

Ejecutemos las aplicaciones y verifiquemos los resultados obtenidos, como hicimos anteriormente con el comando:

```
npm run dev
```

Para concluir está etapa vamos a hacer un par de ajustes visuales de la página: vamos a cambiar el html del index.html para configurar el nombre y el icono de la aplicación en la pestaña del navegador, para lo cual buscamos el archivo index.html y dentro de la etiqueta <head> actualizamos las siguientes etiquetas hijas: link y title

```
 <link
   rel="icon"

href="https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@5.14/svgs/solid/industry.svg"
   />


   <title>Pymes 2025</title>
```

Observe:

- la etiqueta <link rel='icon' …./> define el icono de la pestaña del explorador
- la etiqueta <title> define el título de la pestaña del explorador

Etapa 2

Componente Categorias

Ahora vamos a crear el segundo componente de nuestra aplicación que se llamará **Categorias** y servirá para listar los datos de la tabla Categorias, simplemente será una tabla html que nos mostrará los dos campos de la tabla Categorias .

- Vamos a crear el componente **Categorias**
  - en la carpeta components agregamos el archivo Categorias.jsx

Copiamos en Categorias.jsx el siguiente código en donde definimos una tabla html en donde tenemos hardcodeados 2 registros.

```
    function Categorias() {
return (

        <>
          <div className="tituloPagina">Categorias</div>
          <div className="table-responsive">
             <table className="table table-bordered table-striped">
              <thead>
                <tr>
                  <th style={{ width: "40%" }}>IdCategoria</th>
                  <th style={{ width: "60%" }}>Nombre</th>
                </tr>
              </thead>
              <tbody>
                <tr>
                  <td>1</td>
                  <td>Accesorios</td>
                </tr>
                <tr>
                  <td>2</td>
                  <td>Audio</td>
                </tr>
              </tbody>
```

```
            </table>
          </div>
        </>
      );
    }
    export { Categorias };
```

En el código html anterior se usa la clase de css llamada "tituloPagina" para destacar el título del componente, como la misma va a ser reutilizada por varios componentes, la vamos a definir dentro del archivo App.css (elimine todo el contenido que venía en la plantilla inicial, si aun no lo hizo), con el código que vemos a continuación:

```
    .tituloPagina {
      font-size: 1.75rem;
      font-weight: 500;
      color: white;
      text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
      border-bottom-style: solid;
      border-color: gray;
      border-bottom-width: thin;
      padding-bottom: 0.1em;
      margin-bottom: 0.5em;
    }
  body {
      background-color: rgb(241, 243, 247);
    }
    .divBody {
      background-color: white;
      min-height: 75vh;
```

```
    padding: 1rem;
  }
```

Observe:

- Que el archivo App.css ya existía, ya que fue creado al crear el proyecto, por lo que seguramente tenía código de ejemplo habíamos eliminado, porque solo queremos dejar nuestro código.
- que junto a la clase ".tituloPagina" existen otras reglas de css que usaremos más adelante.

Ahora modificamos el componente App (archivo App.jsx) para que muestre el componente Categorias, usando la definición de estilos que hicimos en App.css, para lo cual necesitamos:

- Importar el archivo de estilo App.css
- Importar el código del componente Categorias
- Modificar el retorno de nuestra función para que devuelva la etiqueta que representa al componente Categorias

quedando como vemos a continuación:

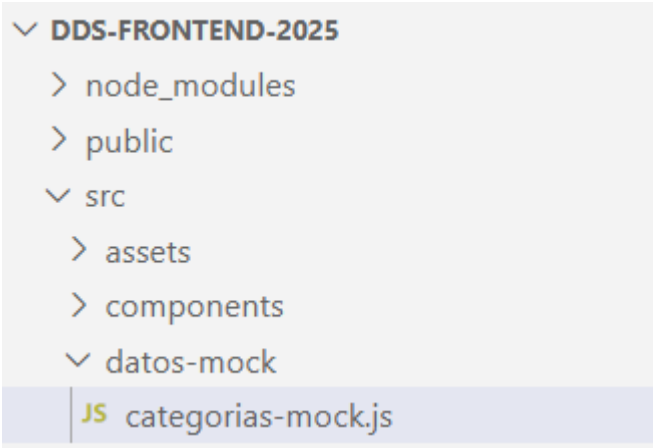
```
import "./App.css";
import {Categorias} from './components/Categorias';
function App() {
  return (
    <>
      <div className="divBody">
          <Categorias/>
      </div>
    </>
  );
}
export default App;
```

Observe:

- En el código anterior se usa la clase css "divBody", para agregar algunos estilos al contenedor de los componentes que ya hemos definido en el archivo App.css

Grabamos todos los cambios y desde el explorador comprobamos que se carga la página definida en el componente Categorias

En el componente Categorias, las filas de la tabla contienen datos estáticos definidos en el html, pero en una aplicación real estos datos son dinámicos y serán recuperados de un backend; yendo hacia ese modelo, vamos a pasar por una etapa intermedia, usando un array de datos datos hardcodeados que representarán los datos de Categorias.
Para lo cual crearemos en /src una carpeta llamada datos-mock y dentro de la misma un archivo llamado categorias-mock.js



con el siguiente contenido:

```
const arrayCategoria = [
    { IdCategoria: 1, Nombre: "ACCESORIOS" },
    { IdCategoria: 2, Nombre: "AUDIO" },
    { IdCategoria: 3, Nombre: "CELULARES" },
    { IdCategoria: 4, Nombre: "CUIDADO PERSONAL" },
    { IdCategoria: 5, Nombre: "DVD" },
    { IdCategoria: 6, Nombre: "FOTOGRAFIA" },
```

```
    { IdCategoria: 7, Nombre: "FRIO-CALOR" },
    { IdCategoria: 8, Nombre: "GPS" },
    { IdCategoria: 9, Nombre: "INFORMATICA" },
]
export default arrayCategoria;
```

A continuación vamos a modificar el componente Categorias para que desde su código se pueda acceder al array recién creado. Al inicio del archivo, Importamos el array arrayCategoria desde su archivo que lo contiene.

```
import arrayCategoria from '../datos-mock/categorias-mock';
```

Dentro de la función del componente Categorias agregamos:

- una constante llamada "categorias" que contenga el array de categorias recién importado que luego va ser recorrido/transformado (mediante la función map) en el html para generar la tabla
- y otra constante "tituloPagina" para mostrar como título de la página

```
function Categorias() {
  const categorias = arrayCategoria;
  const tituloPagina = 'Categorias';
  return (
    //...
  )
}
```

Luego modificamos la respuesta html de nuestro componente Categorias para que: 1) muestre la propiedad Titulo y 2) con ayuda de la función map recorra el array Items y dibuje (pinte o renderice) la tabla. El código completo de nuestro componente quedaria asi:

```
import arrayCategoria from '../datos-mock/categorias-mock';
```

```
function Categorias() {
  const categorias = arrayCategoria;
  const tituloPagina = 'Categorias';
  return (
    <div>
      <div className="tituloPagina">{tituloPagina}</div>
      <table className="table table-bordered table-striped">
        <thead>
          <tr>
            <th style={{ width: "40%" }}>IdCategoria</th>
            <th style={{ width: "60%" }}>Nombre</th>
          </tr>
        </thead>
        <tbody>
          {categorias &&
            categorias.map((categoria) => (
              <tr key={categoria.IdCategoria}>
                <td>{categoria.IdCategoria}</td>
                <td>{categoria.Nombre}</td>
              </tr>
            ))}
        </tbody>
      </table>
    </div>
  );

}

export { Categorias };
```

Ahora probamos los cambios realizado ejecutando la aplicación mediante el comando:

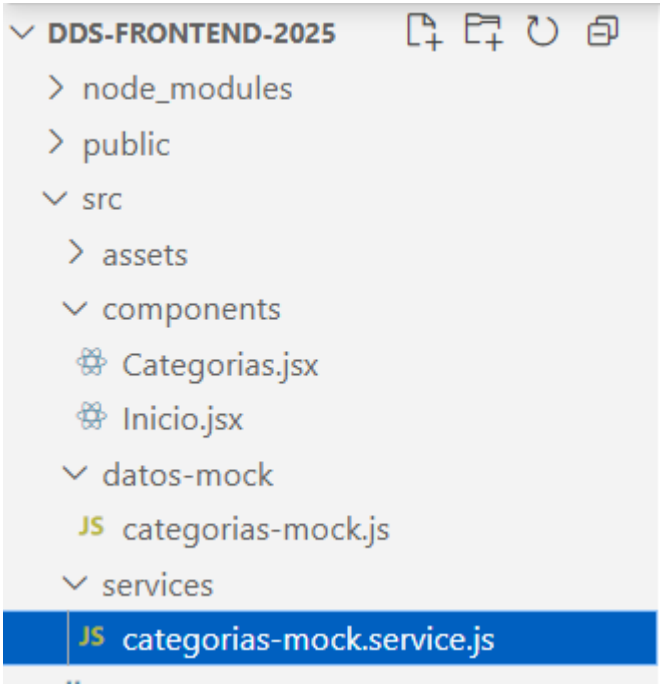
```
npm run dev
```

Observe:

- la técnica usada para renderizar condicionalmente el tbody de la tabla solo si existe la variable categorias.
- La técnica usada para transformar el array Items mediante la función map y por cada ítem generar el tag tr correspondiente a la fila de la tabla.

Servicios:

Para mantener simple nuestro componente, es deseable que solo maneje la renderización de nuestra html y mediante servicios recibir o enviar datos desde y hacia el servidor (o como en está etapa previa con datos moqueados). Para ir hacia ese concepto, seguidamente creamos un servicio que denominaremos categorias-mock.service.js, análogamente como hicimos anteriormente con los componentes, lo haremos dentro de una carpeta “services” donde agrupamos los servicios de nuestra aplicación.



En dicho servicio, crearemos:

- un método “Buscar” que devuelva todos los registros del array Categorias
- un método "BuscarPorId" que devuelve el categoria solicitado.
- un método "Agregar" para dar de alta un registro
- un método "Modificar" para modificar un registro
- un método "Eliminar" para eliminar un registro

Finalmente exportamos la funcionalidad desarrollada

A continuación el código del servicio /src/services/categorias-mock.service.js

```
import arrayCategoria from '../datos-mock/categorias-mock';

async function Buscar() {

     return arrayCategoria;

}

async function BuscarPorId(IdCategoria) {

        return arrayCategoria.find((categoria) => categoria.IdCategoria === IdCategoria);

}

async function Agregar(categoria) {

        categoria.IdCategoria = arrayCategoria.length + 1;    // simula autoincremental

    arrayCategoria.push(categoria);

}

async function Modificar(categoria) {

        let categoriaEncontrado = arrayCategoria.find((categoriafind) => categoriafind.IdCategoria === categoria.IdCategoria);

    if (categoriaEncontrado) {

        categoriaEncontrado.Nombre = categoria.Nombre;

    }

}

async function Eliminar(IdCategoria){

        let categoriaEncontrado = arrayCategoria.find((categoriafind) => categoriafind.IdCategoria === IdCategoria);

    if (categoriaEncontrado) {

          arrayCategoria.splice(arrayCategoria.indexOf(categoriaEncontrado), 1);

    }

}

export const categoriasMockService = {
```

```
    Buscar, BuscarPorId, Agregar, Modificar, Eliminar
};
```

Observe:

- ya pensando en que nuestro servicio real interactuará contra un servidor remoto mediante llamadas asíncronas, hemos definido este mock lo más parecido al servicio que está imitando por lo que la funciones son asíncronas.

Ahora modificamos el componente **Categorias** para que consuma el nuevo servicio y recupere desde allí el array de Categorias. Los cambios serán los siguientes:

- dejaremos de usar directamente el arrayCategoria, en cambio ahora el mismo será provisto por el servicio categoriasMockService
- Haremos uso del hook UseEffect para invocar este servicio al montarse por primera vez el componente.
- Haremos uso del hook UseState para mantener dentro del estado del componente los datos que nos devuelve el servicio.

El nuevo código completo de Categorias.jsx quedaría así:

```
import React, {useState, useEffect} from 'react';
import { categoriasMockService } from '../services/categorias-mock.service';
function Categorias() {
  const tituloPagina = 'Categorias';
  const [categorias, setCategorias] = useState(null);
  // cargar al montar el componente (solo una vez)
  useEffect(() => {
    BuscarCategorias();
  }, []);
  async function BuscarCategorias() {
    let data = await categoriasMockService.Buscar();
    setCategorias(data);
  };
  return (
```

```
    <div>
      <div className="tituloPagina">{tituloPagina}</div>
      <table className="table table-bordered table-striped">
        <thead>
          <tr>
            <th style={{ width: "40%" }}>IdCategoria</th>
            <th style={{ width: "60%" }}>Nombre</th>
          </tr>
        </thead>
        <tbody>
          {categorias &&
            categorias.map((categoria) => (
              <tr key={categoria.IdCategoria}>
                <td>{categoria.IdCategoria}</td>
                <td>{categoria.Nombre}</td>
              </tr>
            ))}
        </tbody>
      </table>
    </div>
  );
}
export {Categorias};
```

Observe:

- Dentro del hook useEffect no llamamos directamente al servicio porque al ser asíncrono recibimos una advertencia del compilador.
- El hook useEffect se ejecuta solo una vez al montar el componente debido a su dependencia vacía: [].

Los efectos en React son código que se ejecuta después del *rende*r**. Es decir, se aplican después de que el componente se monte y se ejecute el primer *render.*** El código de los efectos en React también puede ser ejecutado a continuación de la actualización de un componente, ya sea porque se ha cambiado el estado o las *props*. https://keepcoding.io/blog/efectos-en-react/

Etapa 3
Componentes Menu y Footer

Para poder navegar entre las diferentes páginas de nuestra aplicación, hasta ahora representadas por los componentes **Inicio** y **Categorias** vamos a crear un nuevo componente llamado "**Menu**" que nos permitirá implementar dicha funcionalidad.

Pero antes necesitamos preparar a nuestra aplicación para permitir la navegación según el modelo de SPA, para lo cual importamos un módulo de ruteo que nos ofrece react: react-router-dom. Para ello ejecutaremos el siguiente comando:

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

A continuación modificaremos el html de nuestro componente App en donde, gracias a la funcionalidad del router, indicaremos qué componente se mostrará según la url que se indique en el explorador

El código de nuestro componente App.jsx quedará así:

```
import "./App.css";
import { BrowserRouter, Route, Routes, Navigate } from "react-router-dom";
import {Inicio} from "./components/Inicio";
import {Categorias} from "./components/Categorias";
function App() {
  return (
    <>
      <BrowserRouter>
        <div className="divBody">
            <Routes>
              <Route path="/inicio" element={<Inicio />} />
              <Route path="/categorias" element={<Categorias />} />
              <Route path="*" element={<Navigate to="/Inicio" replace />} />
            </Routes>
        </div>
      </BrowserRouter>
```

```
    </>
  );
}
export default App;
```

Observe:

- que se importa todos los componentes que deseamos navegar (Inicio y Categorias)
- en la etiqueta Route la relación entre la propiedad path (la url) y element (el componente)
- en la última etiqueta Route que luego de evaluarse secuencialmente todas las anteriores y de no encontrar coincidencia en el path, la redirige al path "/Inicio"

En este momento podremos probar nuestra aplicación, la cual nos permitirá según la url solicitada mostrar el componente correspondiente, podríamos probarlo invocando las siguientes urls:

- http://localhost:5173/inicio
- http://localhost:5173/categorias
- http://localhost:5173

Ahora ya configurada nuestra aplicación para interpretar la url del explorador, crearemos el componente Menu que ofrece la interface html para elegir las distintas pantallas (rutas/componentes) que ofrece nuestra aplicación, creamos en la carpeta components el archivo Menu.jsx con el siguiente código:

```
      import React from "react";
      import { NavLink } from "react-router-dom";
      function Menu() {
        return (
              <nav  className="navbar  navbar-dark  bg-dark  navbar-expand-md
no-print px-2">
            <a className="navbar-brand">
              <i className="fa fa-industry"></i>
               <i>Pymes</i>
```

```
            </a>
            <button
              className="navbar-toggler"
              type="button"
              data-toggle="collapse"
              data-target="#navbarSupportedContent"
              aria-controls="navbarSupportedContent"
              aria-expanded="false"
              aria-label="Toggle navigation"
            >
              <span className="navbar-toggler-icon"></span>
            </button>
                              <div    className="collapse    navbar-collapse"
id="navbarSupportedContent">
              <ul className="navbar-nav mr-auto">
                <li className="nav-item">
                  <NavLink className="nav-link" to="/inicio">
                    Inicio
                  </NavLink>
                </li>
                <li className="nav-item">
                  <NavLink className="nav-link" to="/categorias">
                    Categorias
                  </NavLink>
                </li>
              </ul>
            </div>
          </nav>
        );
      }
      export {Menu};
```

Observe:

- el código base para el menú está sacado de la página de bootstrap
- en react los links de navegación se crean con el componente NavLink (de la librería react-router-dom), la cual permite que se le aplique un estilo css cuando el link esta activo, en este caso se le aplica el estilo nav-link-active
- Observe como ha sido exportado este componente y cualquier otra funcionalidad, ya que luego incide como debe ser importado!

Ahora modificamos nuevamente el html del App.jsx para renderizar el menú recién creado, este debe ir dentro de las etiquetas BrowserRouter y previamente debe ser importado, nuestro código quedará asi:

```
import "./App.css";
import { BrowserRouter, Route, Routes, Navigate } from "react-router-dom";
import { Inicio } from "./components/Inicio";
import { Categorias } from "./components/Categorias";
import {Menu} from "./components/Menu";
function App() {
  return (
    <>
      <BrowserRouter>
        <Menu />
        <div className="divBody">
          <Routes>
            <Route path="/inicio" element={<Inicio />} />
            <Route path="/categorias" element={<Categorias />} />
            <Route path="*" element={<Navigate to="/Inicio" replace />} />
          </Routes>
        </div>
      </BrowserRouter>
    </>
```

```
    );
  }
  export default App;
```

En el componente Inicio.jsx configuraremos el link del boton para poder navegar a la pantalla de Categorias, para lo cual haremos el siguiente reemplazo en el código del componente

buscamos el código del botón:

```
<button className="btn btn-lg btn-primary">
        <i className="fa fa-search"> </i>
        Ver Categorias
</button>
```

y lo reemplazamos por el siguiente:

```
<Link to="/categorias" className="btn btn-lg btn-primary">
          <i className="fa fa-search"> </i>  Ver Categorias
</Link>
```

Para que funcione este componente <Link> recien usado, el cual es provisto por la librería react-router-dom, necesitamos importarlo lo haremos con la siguiente línea al inicio archivo Inicio.jsx;

```
import { Link } from "react-router-dom";
```

Ahora en el navegador podremos ver el menú y podremos navegar entre las distintas pantallas de nuestra aplicación mediante los links del menú como así también mediante las urls del explorador.

Análogamente al componente menú que estará visible durante todo el ciclo de vida de nuestra aplicación, crearemos un componente **Footer** que será el pie de página de nuestra aplicación, el contenido del mismo es muy simple ya que solo tiene datos estáticos informativos con algunos links de interés.

Creamos el archivo Footer.jsx en la carpeta components, el código del mismo es el siguiente:

```
import React from 'react';

function Footer() {

  return (

    <footer className="text-center no-print">

    <small>

      <span>© Pymes 2025</span>

      <span className="m-4">-</span>

      <a href="tel:113"> <span className="fa fa-phone"></span> 0810-888-1234
</a>

      <span className="m-4">-</span>

      Seguinos en

      <a

        className="redes"

        href="https://www.facebook.com"

        style={{"backgroundColor": "#2962ff"}}

        target="_blank"

      >

        <i title="Facebook" className="fab fa-facebook-f"></i>

      </a>

      <a

        className="redes"

        href="https://twitter.com"

        style={{"backgroundColor": "#5ba4d6"}}

        target="_blank"

      >

        <i title="Twitter" className="fab fa-twitter"></i>

      </a>

      <a

        className="redes"

        style={{"backgroundColor": "#ec4c51"}}

        href="https://www.instagram.com"
```

```
        target="_blank"
      >
        <i title="Instagram" className="fab fa-instagram"></i>
      </a>
      <a
        className="redes"
        style={{"backgroundColor": "#00e676"}}
        href="https://www.whatsapp.com"
        target="_blank"
      >
        <i title="Whatsapp" className="fab fa-whatsapp"></i>
      </a>
    </small>
  </footer>

  );
}
export { Footer };
```

Finalmente modificamos nuevamente el html del componente App para renderizar el Footer recién creado, y cuya versión final completa quedaría así:

```
import "./App.css";
import React from "react";
import { BrowserRouter, Route, Routes, Navigate } from "react-router-dom";
import {Menu} from "./components/Menu";
import { Footer } from "./components/Footer";
import { Inicio } from "./components/Inicio";
import { Categorias } from "./components/Categorias";
```

```
function App() {
  return (
    <>
        <BrowserRouter>
          <Menu />
          <div className="divBody">
            <Routes>
              <Route path="/inicio" element={<Inicio />} />
              <Route
                path="/categorias"
                element={<Categorias />}
              />
              <Route path="*" element={<Navigate to="/inicio" replace />} />
            </Routes>
          </div>
          <Footer />
        </BrowserRouter>
    </>
  );
}
export default App;
```

y agregamos también algunas clases css para darle un poco de estilo al componente y a otras partes de la aplicación que desarrollaremos más adelante, el código de estas clases lo pondremos en el archivo App.css, aquí va su contenido completo (incluye desarroolado anteriormente):

```
body {
  background-color: rgb(241, 243, 247);
}
```

```
.divBody {
  background-color: white;
  min-height: 75vh;
  padding: 1rem;
}

/* pie de la pagina */
footer {
  padding: 1rem;
}
.navbar-brand {
  color: aquamarine;
  font-size: 1.5rem;
}

.tituloPagina {
  font-size: 1.75rem;
  font-weight: 500;
  color: white;
  text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
  border-bottom-style: solid;
  border-color: gray;
  border-bottom-width: thin;
  padding-bottom: 0.1em;
  margin-bottom: 0.5em;
}




/* todos los formularios */
form {
```

```
  background-color: rgb(241, 243, 247);
  border-radius: 0.5rem;
  border-style: solid;
  border-color: lightgrey;
  border-width: thin;
  margin: 0em 0.5rem 1rem 0.5rem;
  padding: 0.5rem;
  box-shadow: 0.62rem 0.62rem 0.31rem rgb(175, 173, 173);
}

/* espacio en los botones contenidos en un div con esta clase */
.botones button {
  margin: 0em 0.5em 0.5em 0em;
}

/* todos los labels menos los de los radio y check */
label:not([class^='custom-control']) {
  background-color: #d7dfe7;
  border-radius: 0.625em 0.2em 0rem 0.625em;
  border-bottom: solid;
  margin-right: -0.9em; /* para que columna label este cerca del input*/
  padding-left: 0.5em;
  display: block;
  height: 90%;
}

/*<=576*/
@media only screen and (max-width: 575px) {
  label:not([class^='custom-control']) {
    margin-right: 0em;
    margin-top: 0.3rem; /* para separar grupo label/input */
```

```
  }
}

/*>=576*/
@media only screen and (min-width: 576px) {
  .inputMedio {
    width: 12em;
  }
  .inputChico {
    width: 7em;
  }
}

.pyBadge {
  background-color: #6c757d;
  color: white;
  /* font-size: 1.1em; */
  display: inline-block;
  padding: 0.35em 0.4em;
  font-weight: 700;
  line-height: 1;
  text-align: center;
  white-space: nowrap;
  vertical-align: baseline;
  border-radius: 0.25rem;
}

/* mensajes de validacion */
.validacion {
  color: red;
  font-size: 0.8em;
```

```
  font-style: italic;
  padding-left: 1em;
  padding-bottom: 0.5em;
}

/* mensajes de alerta al buscar y grabar */
.mensajesAlert {
  /* color: #495057; */
  margin: 0rem 0.5rem 0rem 0.5rem;
}

.paginador {
  background-color: #e9ecef;
  padding: 0.5rem 1rem 0rem 1rem;
  margin-top: -1rem; /* para pegarlo a la tabla */
}

/* todos los input a mayusculas, y no sus placeholders*/
input {
  text-transform: uppercase;
  /* margin-bottom: 0.2em;  */
  /*separa renglones*/
}
::placeholder {
  text-transform: none;
}

/* encabezados de tablas */
thead {
  /* bootstrap thead-light*/
  color: #495057;
```

```
  background-color: #aeb7d7; /* #e9ecef; */
  border-color: #dee2e6;
}

/* redes al pie de pagina */
.redes {
  background-color: blue;
  color: white;
  padding: 0.4em;
  margin: 0.3em;
  border-radius: 1em;
}

/* no imprimir las clases no-print */
@media print {
  .no-print {
    display: none;
  }
}
```

Observe:

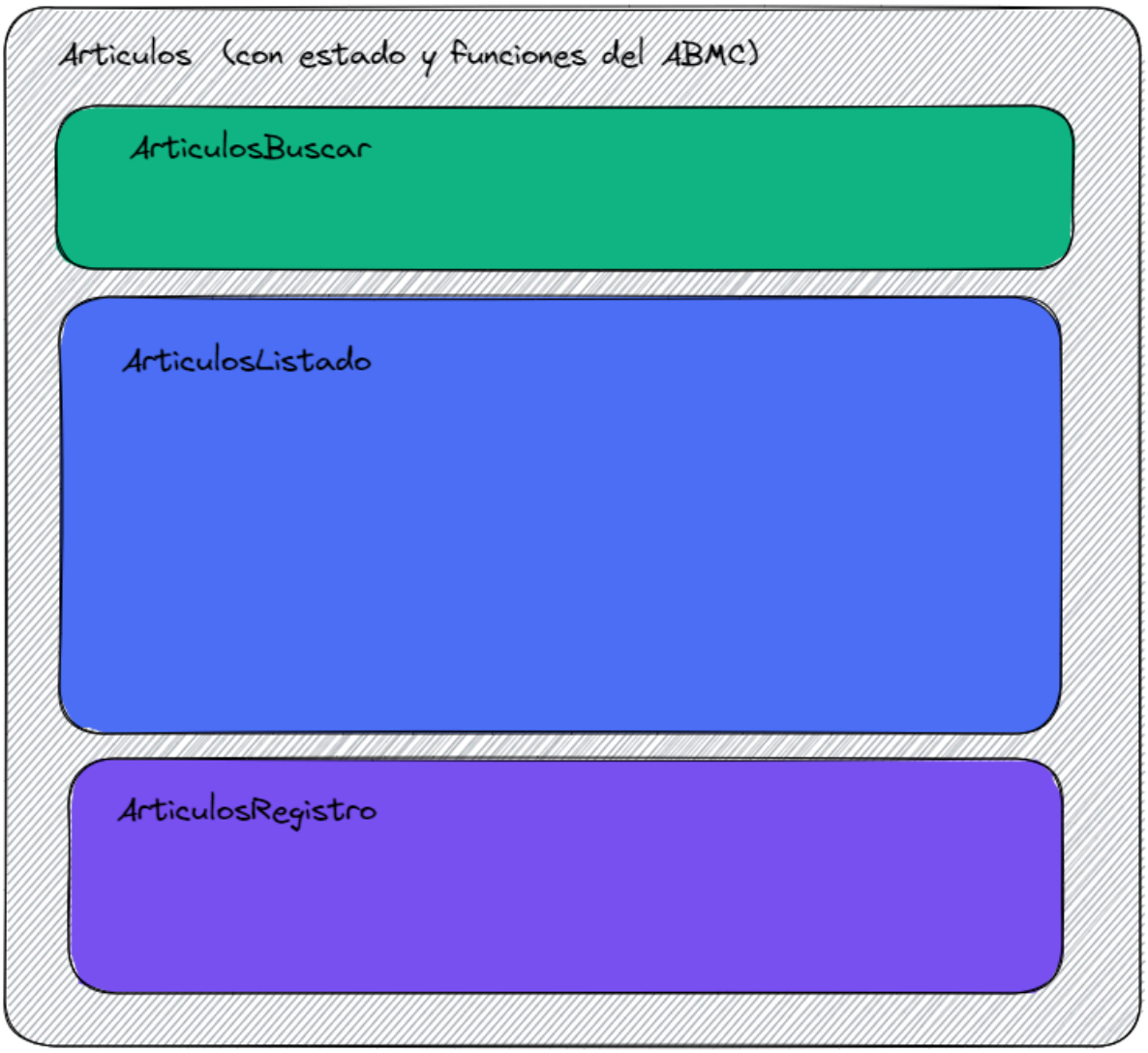
- que a los inputs se le aplica un estilo que hace que visualmente los datos ingresados se “vean” en mayúsculas, luego combinado con el backend el contenido de los mismos se guardará en mayúsculas en la base de datos.

  Ahora podemos ejecutar nuestra aplicación comprobar el código que hemos escrito!

Etapa 4

Componente Articulos

A continuación crearemos el siguiente componente (página) de nuestra aplicación: **Articulos** que como dijimos inicialmente nos brinda la funcionalidad de un ABMC (Alta, Baja, Modificaciones y Consultas) sobre los datos de la tabla Artículos. Pensando en la estructura de nuestro componente debido a que tendrá mayor complejidad que el componente anterior Categorias, comenzaremos con un diseño conceptual del mismo que estará representado en el siguiente gráfico:



Como vemos la estructura propuesta incluye:

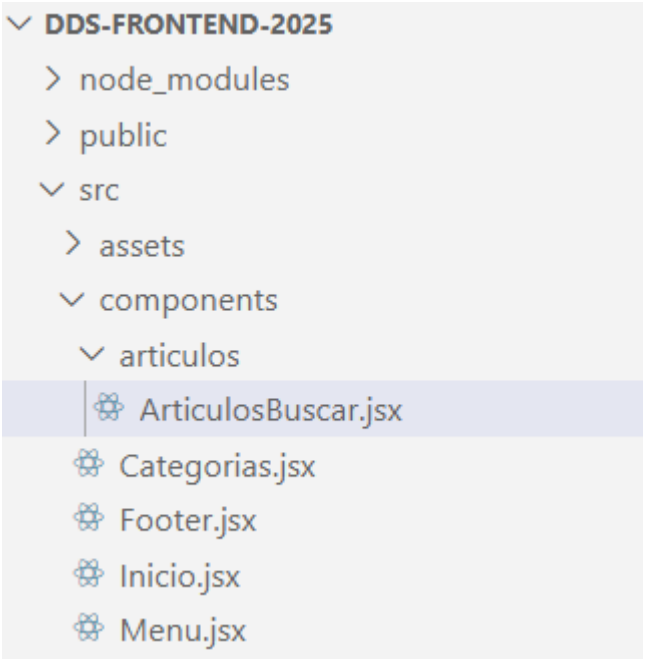
- Un componente contenedor llamado "Articulos.jsx" el cual estará encargado de gestionar los estados, la funcionalidad necesaria para el ABMC y una interface visual mínima que incluye el título de la página.
- y tres componentes hijos
    1. "ArticulosBuscar.jsx" que permitirá filtrar la búsqueda de los Artículos según un par de parámetros: Nombre (descripcion del articulo) y Activo (booleano que indica si el artículo está activo o no, se usa para reprensar una baja lógica)

2. "ArticulosListado.jsx" que permitirá mostrar en una tabla un resumen (algunos campos representativos) del resultado de la búsqueda según los parámetros establecidos en el componente anterior (componente hermano). Aquí se incluye también: un contador de registros que cumplen la condición de filtrado, un paginador, un botón imprimir y un mensaje para avisar cuando no se encuentren resultados según el criterio establecido.
3. "ArticulosRegistro" que permitirá ver todos los campos de un determinado registro seleccionado, los cuales podrán ser "Consultados" o "Modificados" y también esta misma interface se usará para dar de "Alta" un nuevo registro.

Comenzaremos creando una versión estática de los componentes y luego le iremos agregando funcionalidad, todo dentro de una carpeta dentro de components llamada igual que el componente "articulos"

Empecemos creando los componentes hijos:

- Creamos el componente "ArticulosBuscar" mediante el archivo src/components/articulos/ArticulosBuscar.jsx



- Le agregamos el siguiente código:

```
      import React from "react";

      export default function ArticulosBuscar ({Nombre, setNombre, Activo,
setActivo, Buscar, Agregar}) {


          return (

          <form name="FormBusqueda">
```

```
            <div className="container-fluid">
              <div className="row">
                <div className="col-sm-4 col-md-2">
                  <label className="col-form-label">Nombre:</label>
                </div>
                <div className="col-sm-8 col-md-4">
                  <input
                    type="text"
                    className="form-control"
                    onChange={(e) => setNombre(e.target.value)}
                    value={Nombre}
                    maxLength="55"
                    autoFocus
                  />
                </div>
                <div className="col-sm-4 col-md-2">
                  <label className="col-form-label">Activo:</label>
                </div>
                <div className="col-sm-8 col-md-4">
                  <select
                    className="form-control"
                    onChange={(e) => setActivo(e.target.value)}
                    value={Activo}
                  >
                    <option value={""}></option>
                    <option value={false}>NO</option>
                    <option value={true}>SI</option>
                  </select>
                </div>
              </div>
```

```
            <hr />


            {/* Botones */}
            <div className="row">
              <div className="col text-center botones">
              <button
                type="button"
                className="btn btn-primary"
                onClick={() => Buscar(1) }
              >
                <i className="fa fa-search"> </i> Buscar
              </button>
              <button
                type="button"
                className="btn btn-primary"
                onClick={() => Agregar() }
              >
                <i className="fa fa-plus"> </i> Agregar
              </button>
              </div>
            </div>
          </div>
        </form>
        )
      };
```

Observe:

- este componente recibe como parámetros dos estados: "Nombre" y "Activo" y dos funciones "Buscar" y "Agregar" que serán provistas por su componente padre "Articulos"
- la técnica ("formularios controlados") usada para enlazar los estados con los campos del formulario, que es mediante la propiedad "value" de los campos y la función "onChange" que se ejecuta cada vez que se modifica el valor del campo.

Creamos el componente "ArticulosListado" mediante el archivo src/components/articulos/ArticulosListado.jsx con el siguiente código:

```
      import React from "react";
      import moment from "moment";


      export default function ArticulosListado({
        Items,
        Consultar,
        Modificar,
        ActivarDesactivar,
        Imprimir,
        Pagina,
        RegistrosTotal,
        Paginas,
        Buscar,
      }) {
        return (
          <div className="table-responsive">
               <table  className="table  table-hover  table-sm  table-bordered
table-striped">
              <thead>
                <tr>
                  <th className="text-center">Nombre</th>
                  <th className="text-center">Precio</th>
                  <th className="text-center">Stock</th>
                  <th className="text-center">Fecha de Alta</th>
                  <th className="text-center">Activo</th>
                  <th className="text-center text-nowrap">Acciones</th>
                </tr>
```

```
            </thead>
            <tbody>
              {Items &&
                Items.map((Item) => (
                  <tr key={Item.IdArticulo}>
                    <td>{Item.Nombre}</td>
                    <td className="text-end">{Item.Precio}</td>
                    <td className="text-end">{Item.Stock}</td>
                    <td className="text-end">
                      {moment(Item.FechaAlta).format("DD/MM/YYYY")}
                    </td>
                    <td>{Item.Activo ? "SI" : "NO"}</td>
                    <td className="text-center text-nowrap">
                      <button
                        className="btn btn-sm btn-outline-primary"
                        title="Consultar"
                        onClick={() => Consultar(Item)}
                      >
                        <i className="fa fa-eye"></i>
                      </button>
                      <button
                        className="btn btn-sm btn-outline-primary"
                        title="Modificar"
                        onClick={() => Modificar(Item)}
                      >
                        <i className="fa fa-pencil"></i>
                      </button>
                      <button
                        className={
                          "btn btn-sm " +
                          (Item.Activo
```

```
                                ? "btn-outline-danger"
                                : "btn-outline-success")
                            }
                            title={Item.Activo ? "Desactivar" : "Activar"}
                            onClick={() => ActivarDesactivar(Item)}
                          >
                            <i
                                className={"fa fa-" + (Item.Activo ? "times" :
"check")}
                            ></i>
                          </button>
                        </td>
                      </tr>
                    ))}
                </tbody>
              </table>

              {/* Paginador*/}
              <div className="paginador">
                <div className="row">
                  <div className="col">
                                        <span   className="pyBadge">Registros:
{RegistrosTotal}</span>
                  </div>
                  <div className="col text-center">
                    Pagina:  
                    <select
                      value={Pagina}
                      onChange={(e) => {
                        Buscar(e.target.value);
                      }}
                    >
```

```
                    {Paginas?.map((x) => (
                      <option value={x} key={x}>
                        {x}
                      </option>
                    ))}
                  </select>
                    de {Paginas?.length}
                </div>

                <div className="col">
                   <button className="btn btn-primary float-end" onClick={()
=> Imprimir()}>
                    <i className="fa fa-print"></i>Imprimir
                  </button>
                </div>
              </div>
            </div>
          </div>
        );
      }
```

Observe:

- este componente recibe como parámetros los estados: Items, RegistrosTotal, Pagina y Páginas y las funciones Buscar, Consultar, Modificar, ActivarDesactivar, Imprimir, que serán provistas por su componente padre "Articulos"
- la tabla html para mostrar los registros y los botones que permiten ejecutar las acciones sobre los mismos: Consultar, Modificar, ActivarDesactivar
- el uso de la librería moment para formatear la fecha de alta. (requiere que la instalemos con npm install moment)
- la técnica usada en la tabla de artículos en la columna Activo para que aparezca según el valor de la propiedad item.Activo (true/false) el texto SI/NO

  ```
  <td>{Item.Activo ? "SI" : "NO"}</td>
  ```

- la técnica usada en la tabla de artículos en la columna Acciones para que aparezca según el valor de la propiedad item.Activo (true/false) el texto Desactivar/Activar y el icono times/check y otras similares.

- el modo de pasar el item actual como parámetro a las funciones Consultar, Modificar y ActivarDesactivar.

```
onClick={() => Consultar(Item)}
...
onClick={() => Modificar(Item)}
...
onClick={() => ActivarDesactivar(Item)}
```

Como vimos en las Observaciones instalamos la librería moment con el siguiente comando de consola:

```
npm install moment
```

Creamos el componente "**ArticulosRegistro**" mediante el archivo src/components/articulos/ArticulosRegistro.jsx con el siguiente código:

```
import React from "react";
export default function ArticulosRegistro({
    AccionABMC,
    Categorias,
    Item,
    Grabar,
    Volver,
}) {
    if (!Item) return null;
    return (
        <form>
            <div className="container-fluid">

                <fieldset disabled={AccionABMC === "C"}>

                    {/* campo nombre */}
                    <div className="row">
                        <div className="col-sm-4 col-md-3 offset-md-1">
```

```
                        <label className="col-form-label" htmlFor="Nombre">
                            Nombre<span className="text-danger">*</span>:
                        </label>
                    </div>
                    <div className="col-sm-8 col-md-6">
                        <input
                            type="text"
                            name="Nombre"
                            value={Item?.Nombre}
                            autoFocus
                            className="form-control "
                        />
                    </div>
                </div>

                {/* campo Precio */}
                <div className="row">
                    <div className="col-sm-4 col-md-3 offset-md-1">
                        <label className="col-form-label" htmlFor="Precio">
                            Precio<span className="text-danger">*</span>:
                        </label>
                    </div>
                    <div className="col-sm-8 col-md-6">
                        <input
                            type="number"
                            step=".01"
                            name="Precio"
                            value={Item.Precio}
                            className="form-control"
                        />
                    </div>
                </div>
```

```
                    {/* campo Stock */}
                    <div className="row">
                        <div className="col-sm-4 col-md-3 offset-md-1">
                            <label className="col-form-label" htmlFor="Stock">
                                Stock<span className="text-danger">*</span>:
                            </label>
                        </div>
                        <div className="col-sm-8 col-md-6">
                            <input
                                type="number"
                                name="Stock"
                                value={Item.Stock}
                                className="form-control"
                            />
                        </div>
                    </div>

                    {/* campo CodigoDeBarra */}
                    <div className="row">
                        <div className="col-sm-4 col-md-3 offset-md-1">
                                                <label  className="col-form-label"
htmlFor="CodigoDeBarra">
                                                                Codigo  De  Barra<span
className="text-danger">*</span>:
                            </label>
                        </div>
                        <div className="col-sm-8 col-md-6">
                            <input
                                type="text"
                                name="CodigoDeBarra"
                                value={Item.CodigoDeBarra}
                                className="form-control"
                            />
                        </div>
```

```
                    </div>


                    {/* campo IdCategoria*/}
                    <div className="row">
                        <div className="col-sm-4 col-md-3 offset-md-1">
                                                <label  className="col-form-label"
htmlFor="IdCategoria">
                                Categoria<span className="text-danger">*</span>:
                            </label>
                        </div>
                        <div className="col-sm-8 col-md-6">
                            <select
                                name="IdCategoria"
                                className="form-control"
                                value={Item?.IdCategoria}
                            >
                                <option value="" key={1}></option>
                                {Categorias?.map((x) => (
                                                        <option  value={x.IdCategoria}
key={x.IdCategoria}>
                                        {x.Nombre}
                                    </option>
                                ))}
                            </select>
                        </div>
                    </div>


                    {/* campo FechaAlta */}
                    <div className="row">
                        <div className="col-sm-4 col-md-3 offset-md-1">
                            <label className="col-form-label" htmlFor="FechaAlta">
                                Fecha Alta<span className="text-danger">*</span>:
                            </label>
                        </div>
```

```
                    <div className="col-sm-8 col-md-6">
                        <input
                            type="date"
                            name="FechaAlta"
                            className="form-control"
                            value={Item?.FechaAlta}
                        />
                    </div>
                </div>

                {/* campo Activo */}
                <div className="row">
                    <div className="col-sm-4 col-md-3 offset-md-1">
                        <label className="col-form-label" htmlFor="Activo">
                            Activo<span className="text-danger">*</span>:
                        </label>
                    </div>
                    <div className="col-sm-8 col-md-6">
                        <select
                            name="Activo"
                            className="form-control"
                            value={Item?.Activo}
                            disabled
                        >
                            <option value={null}></option>
                            <option value={false}>NO</option>
                            <option value={true}>SI</option>
                        </select>
                    </div>
                </div>

            </fieldset>
```

```
                {/* Botones Grabar, Cancelar/Volver' */}
                <hr />
                <div className="row justify-content-center">
                    <div className="col text-center botones">
                        {AccionABMC !== "C" && (
                            <button type="submit"
                                className="btn btn-primary"
                                onClick={() => Grabar()}>
                                <i className="fa fa-check"></i> Grabar
                            </button>
                        )}
                        <button
                            type="button"
                            className="btn btn-warning"
                            onClick={() => Volver()}
                        >
                            <i className="fa fa-undo"></i>
                            {AccionABMC === "C" ? " Volver" : " Cancelar"}
                        </button>
                    </div>
                </div>

                {/* texto: Revisar los datos ingresados... */}
                <div className="row alert alert-danger mensajesAlert">
                    <i className="fa fa-exclamation-sign"></i>
                    Revisar los datos ingresados...
                </div>

            </div>
        </form>
    );
}
```

Observe:

- este componente recibe como parámetros los estados: AccionABMC, Categorias, Item y las funciones Grabar y Volver que serán provistas por su componente padre "Articulos"
- que los inputs Nombre,Precio,Stock y CodigoDeBarra tienen "temporalmente", el atributo value que se vincula al estado Item; y aun cuando arrojen un warning por consola, solo forma parte del boceto inicial del componente y más adelante será reemplazado por un método adecuado de enlace de datos. Por lo tanto los campos Categoria, FechaAlta y Activo por ahora no están vinculados a sus datos correspondientes.
- que los inputs y selects están deshabilitados cuando AccionABMC es "C" (consulta), gracias al contenedor fieldset
- el div con el mensaje de error: "Revisar los datos ingresados...", el mismo será usado más adelante cuando se implementen validaciones de datos.
- la técnica usada para generar las etiquetas options del select Categorias con los datos traídos desde el servidor.

```
{Categorias?.map((x) => (
  <option value={x.IdCategoria} key={x.IdCategoria}>
    {x.Nombre}
  </option>
))}
```

Ahora que tenemos los 3 componentes hijos creados, pasamos a desarrollar el componente padre que los contendrá. Creamos el componente "**Articulos**" mediante el archivo src/components/articulos/Articulos.jsx con el siguiente código:

```
import React, { useState, useEffect } from "react";
import moment from "moment";
import ArticulosBuscar from "./ArticulosBuscar";
import ArticulosListado from "./ArticulosListado";
import ArticulosRegistro from "./ArticulosRegistro";
import { categoriasMockService as categoriasService } from "../../services/categorias-mock.service";

function Articulos() {
  const TituloAccionABMC = {
    A: "(Agregar)",
```

```
          B: "(Eliminar)",
          M: "(Modificar)",
          C: "(Consultar)",
          L: "(Listado)",
        };
        const [AccionABMC, setAccionABMC] = useState("L");

        const [Nombre, setNombre] = useState("");
        const [Activo, setActivo] = useState("");

        const [Items, setItems] = useState(null);
         const [Item, setItem] = useState(null); // usado en BuscarporId (Modificar, Consultar)
        const [RegistrosTotal, setRegistrosTotal] = useState(0);
        const [Pagina, setPagina] = useState(1);
        const [Paginas, setPaginas] = useState([]);

        const [Categorias, setCategorias] = useState(null);

         // cargar al "montar" el componente, solo la primera vez (por la dependencia [])
        useEffect(() => {
          async function BuscarCategorias() {
            let data = await categoriasService.Buscar();
            setCategorias(data);
          }
          BuscarCategorias();
        }, []);

      async function Buscar() {
          setAccionABMC("L");
          // harcodeamos 2 articulos para probar
```

```
          setItems([
            {
              IdArticulo: 108,
              Nombre: "ADAPTADOR USB WIFI TL-WN722N",
              Precio: 219.0,
              CodigoDeBarra: "0693536405046",
              IdCategoria: 9,
              Stock: 898,
              FechaAlta: "2017-01-23T00:00:00",
              Activo: false,
            },
            {
              IdArticulo: 139,
              Nombre: "AIRE ACONDICIONADO DAEWOO 3200FC DWT23200FC",
              Precio: 5899.0,
              CodigoDeBarra: "0779816944014",
              IdCategoria: 7,
              Stock: 668,
              FechaAlta: "2017-01-04T00:00:00",
              Activo: true,
            },
          ]);
          setRegistrosTotal(2);
          setPaginas([1]);
          alert("Buscando...");
        }

        async function BuscarPorId(item, accionABMC) {
          setAccionABMC(accionABMC);
          setItem(item);
          if (accionABMC === "C") {
```

```
            alert("Consultando...");
          }
          if (accionABMC === "M") {
            alert("Preparando la Modificacion...");
          }
        }

        function Consultar(item) {
           BuscarPorId(item, "C"); // paso la accionABMC pq es asincrono la busqueda y luego de ejecutarse quiero cambiar el estado accionABMC
        }
        function Modificar(item) {
          if (!item.Activo) {
            alert("No puede modificarse un registro Inactivo.");
            return;
          }
           BuscarPorId(item, "M"); // paso la accionABMC pq es asincrono la busqueda y luego de ejecutarse quiero cambiar el estado accionABMC
        }

        async function Agregar() {
          setAccionABMC("A");
          setItem({
              IdArticulo: 0,
              Nombre: '',
              Precio: '',
              Stock: '',
              CodigoDeBarra: '',
              IdCategoria: '',
              FechaAlta: moment(new Date()).format("YYYY-MM-DD"),
              Activo: true,
            });
```

```
    alert("preparando el Alta...");
  }



  function Imprimir() {
    window.print()
    //En desarrollo...
  }

  async function ActivarDesactivar(item) {
    const resp = window.confirm(
      "Está seguro que quiere " +
        (item.Activo ? "desactivar" : "activar") +
        " el registro?"
    );
    if (resp) {
      alert("Activando/Desactivando...");
    }
  }

  async function Grabar(item) {
    alert(
      "Registro " +
        (AccionABMC === "A" ? "agregado" : "modificado") +
        " correctamente."
    );

    Volver();
  }

  // Volver/Cancelar desde Agregar/Modificar/Consultar
```

```
        function Volver() {
          setAccionABMC("L");
        }

        return (
          <div>
            <div className="tituloPagina">
              Articulos <small>{TituloAccionABMC[AccionABMC]}</small>
            </div>

            <ArticulosBuscar
              Nombre={Nombre}
              setNombre={setNombre}
              Activo={Activo}
              setActivo={setActivo}
              Buscar={Buscar}
              Agregar={Agregar}
            />

            {/* Tabla de resutados de busqueda y Paginador */}
            <ArticulosListado
              {...{
                Items,
                Consultar,
                Modificar,
                ActivarDesactivar,
                Imprimir,
                Pagina,
                RegistrosTotal,
                Paginas,
                Buscar,
```

```
            }}
          />

          <div className="alert alert-info mensajesAlert">
            <i className="fa fa-exclamation-sign"></i>
            No se encontraron registros...
          </div>

          {/* Formulario de alta/modificacion/consulta */}
          <ArticulosRegistro
            {...{ AccionABMC, Categorias, Item, Grabar, Volver }}
          />
        </div>
      );
    }
    export { Articulos };
```

Observe:

- En la función Buscar, hemos hardcodeado el estado Ítems para simular datos y probar el componente ArticulosListado.jsx, luego generamos el servicio para buscar los artículos y sus datos reales del servidor.
- Observe en html la sintaxis usada para pasar los parametros al Componente `ArticulosBuscar` vs la usada para pasar los parámetros a los componentes `ArticulosListado` y `ArticulosRegistro`, la cual es mas simple gracias al uso del operador de propagación de javascript

Para poder utilizar el componente Articulos.jsx en la aplicación:

- debemos importarlo en el componente App.jsx
  import { Articulos } from "./components/articulos/Articulos";

- para luego poder agregarlo al ruteo correspondiente en el return de la función App(), también en el archivo App.jsx
  `<Route path="/articulos" element={<Articulos/>} />`

- y finalmente agregamos un enlace en el componente Menu.js para que el usuario pueda acceder a la página de Artículos.

```
<li className="nav-item">
   <NavLink className="nav-link" to="/articulos">
       Articulos
   </NavLink>
</li>
```

Ahora pruebe la aplicación y verifique que el componente Articulos.js renderice correctamente a sus componentes hijos y responda adecuadamente a los eventos del ABMC.

Inicialmente todos los componentes hijos de Articulos están visibles, pero a continuación modificaremos el html que cada componente se muestre cuando corresponda, para lo cual nos basaremos en la acción actual del ABMC indicado por el estado AccionABMC.

Hacemos los siguientes cambios en el componente Articulos.jsx:

reemplazamos:

```
<ArticulosBuscar
        Nombre={Nombre}
        setNombre={setNombre}
        Activo={Activo}
        setActivo={setActivo}
        Buscar={Buscar}
        Agregar={Agregar}
      />
```

por:

```
{AccionABMC === "L" && <ArticulosBuscar
      Nombre={Nombre}
      setNombre={setNombre}
      Activo={Activo}
      setActivo={setActivo}
      Buscar={Buscar}
      Agregar={Agregar}
```

```
    />
}
```

reemplazamos:

```
 <ArticulosListado
        {...{
          Items,
          Consultar,
          Modificar,
          ActivarDesactivar,
          Imprimir,
          Pagina,
          RegistrosTotal,
          Paginas,
          Buscar,
        }}
       />
```

por:

```
{AccionABMC === "L" && Items?.length > 0 &&
 <ArticulosListado
      {...{
        Items,
        Consultar,
        Modificar,
        ActivarDesactivar,
        Imprimir,
        Pagina,
        RegistrosTotal,
        Paginas,
        Buscar,
```

```
    }}
   />
}
```

reemplazamos:

```
<div className="alert alert-info mensajesAlert">
        <i className="fa fa-exclamation-sign"></i>
        No se encontraron registros...
</div>
```

por:

```
{AccionABMC === "L" && Items?.length === 0 &&
      <div className="alert alert-info mensajesAlert">
        <i className="fa fa-exclamation-sign"></i>
        No se encontraron registros...
      </div>
}
```

reemplazamos:

```
<ArticulosRegistro
        {...{ AccionABMC, Categorias, Item, Grabar, Volver }}
/>
```

por:

```
{AccionABMC !== "L" &&
      <ArticulosRegistro
        {...{ AccionABMC, Categorias, Item, Grabar, Volver }}
      />
}
```

Ejecute la aplicación y verifique que el componente Articulos.jsx renderice solo los componentes según la acción que se esté ejecutando y responde adecuadamente a cambios en el estado del ABMC.

Etapa 5

FUNCIONALIDADES DEL ABMC

En esta etapa comenzamos a completar las funcionalidades del ABMC de artículos, para lo cual vamos a crear el servicio de artículos que nos permitirá consumir los datos de la webapi expuesta por el backend. Inicialmente nos hará falta instalar la librería axios para poder realizar las peticiones http al servidor, para lo cual ejecutamos el siguiente comando en la consola:

```
npm install axios
```

Seguidamente creamos el archivo articulos.service.js en la carpeta services y agregamos el siguiente código:

```
import axios from "axios";
const urlResource = "https://labsys.frc.utn.edu.ar/dds-backend-2025/api/articulos";
async function Buscar(Nombre, Activo, Pagina) {
  const resp = await axios.get(urlResource, {
    params: { Nombre, Activo, Pagina },
  });
  return resp.data;
}
async function BuscarPorId(item) {
  const resp = await axios.get(urlResource + "/" + item.IdArticulo);
  return resp.data;
}
async function ActivarDesactivar(item) {
  await axios.delete(urlResource + "/" + item.IdArticulo);
}
async function Grabar(item) {
  if (item.IdArticulo === 0) {
    await axios.post(urlResource, item);
  } else {
    await axios.put(urlResource + "/" + item.IdArticulo, item);
  }
}
```

```
export const articulosService = {

  Buscar,BuscarPorId,ActivarDesactivar,Grabar

};
```

Observe:

- la correcta configuración de la url del recurso.
- que el servicio ofrece todas las funciones necesarias para el ABMC.
- la técnica usada en el función Grabar(), para identificar si es un alta o modificación y consecuentemente llamar al método post o put.

Análogamente al servicio de artículos, creamos el servicio de categorias en el archivo categorias.service.js (que reemplazara a la anterior version categorias-mock.service.js) y agregamos el siguiente código:

```
import axios from "axios";

const urlResource = "https://labsys.frc.utn.edu.ar/dds-backend-2025/api/categorias";

async function Buscar() {

  const resp = await axios.get(urlResource);

  return resp.data;

}

export const categoriasService = {

  Buscar

};
```

Observe:

- la correcta configuración de la url del recurso.
- que el servicio es una versión simplificada del mismo ya que solo ofrece la función Buscar.
- el mismo será usado por el componente ArticulosRegistro.jsx para cargar el "Select" de categorías. Y será recibido mediante propiedades desde el componente padre Articulos.jsx

Ahora vamos a completar las funcionalidades del ABMC de artículos, completando los siguiente pasos:

Hacemos uso de los nuevos servicios, al inicio en el componente Articulos.jsx importamos los servicios de articulos.service y categorias.service y comentamos el import de categoriasMockService que ahora es reemplazado:

```
import { articulosService } from "../../services/articulos.service";

//import { categoriasMockService as categoriasService } from "../../services/categorias-mock.service";
import { categoriasService } from "../../services/categorias.service";
```

Observe:

- la misma variable “categoriasService” (en el código anterior) que antes obtenía datos mockeados y referenciada desde categoriasMockService, ahora referencia al nuevo servicio que hace la llamada al backend para solicitar los datos del servidor.
- el código para traer datos del servidor para cargar el combo en el usado en el componente ArticulosRegistro.jsx, sigue siendo el que se ejecuta en la función `useEffect()` de Articulos.jsx que ahora usa el nuevo servicio. Este código solo se ejecuta una sola vez cada vez que se carga el componente Articulos.jsx al DOM

Funcionalidad Buscar:

En el componente Articulos.jsx completamos la función Buscar, su nuevo código será:

```
async function Buscar(_pagina) {
    if (_pagina && _pagina !== Pagina) {
      setPagina(_pagina);
    }
     // OJO Pagina (y cualquier estado...) se actualiza para el proximo render, para buscar usamos el parametro _pagina
    else {
      _pagina = Pagina;
    }

    const data = await articulosService.Buscar(Nombre, Activo, _pagina);
    setItems(data.Items);
    setRegistrosTotal(data.RegistrosTotal);

    //generar array de las páginas para mostrar en select del paginador
    const arrPaginas = [];
    for (let i = 1; i <= Math.ceil(data.RegistrosTotal / 10); i++) {
      arrPaginas.push(i);
```

```
    }
    setPaginas(arrPaginas);
  }
```

Observe:

- *el parámetro _pagina nos permitirá implementar la paginación de los resultados de la búsqueda en el servidor.*
- *todas las funciones del ABMC que realizan llamadas a apis externas las marcaremos con asíncronas*
- *cómo se genera el array "arrPaginas" que representa las páginas a mostrar en el paginador.*

Funcionalidad BuscarPorId:

El nuevo código de la función será:

```
async function BuscarPorId(item, accionABMC) {
  const data = await articulosService.BuscarPorId(item);
  setItem(data);
  setAccionABMC(accionABMC);
}
```

Observe:

- Cuando se recibe el json del servidor el campo fecha llega desde la webapi convertido en string con el formato ISO 8601 y es adecuado para el input type date. Para poder verificar esto debemos hacer uso de Chrome devtools y analizar la pestaña "Network", buscando los datos que vienen del servidor.

Funcionalidad ActivarDesactivar:

En el componente Articulos.jsx completamos la función ActivarDesactivar, la cual es una implementación particular de una baja lógica. El nuevo código de la función será:

```
async function ActivarDesactivar(item) {
  const resp = window.confirm(
    "Está seguro que quiere " +
      (item.Activo ? "desactivar" : "activar") +
      " el registro?"
  );
  if (resp) {
    await articulosService.ActivarDesactivar(item);
    await Buscar();
  }
}
```

Observe:

- que el método ActivarDesactivar() del servicio, es muy sencillo y solo cambia el estado Activo del registro seleccionado invirtiendo su valor. Sirviendo para pasar un registro de estado Activo a Inactivo y viceversa.

Funcionalidad Grabar:

En el componente Articulos.jsx completamos la función Grabar, la cual es usada tanto para grabar el alta como la modificación de un registro. El nuevo código de la función será:

```
async function Grabar(item) {
  // agregar o modificar
  try
  {
    await articulosService.Grabar(item);
  }
  catch (error)
  {
    alert(error?.response?.data?.message ?? error.toString())
```

```
        return;
      }
      await Buscar();
      Volver();

      setTimeout(() => {
        alert(
          "Registro " +
            (AccionABMC === "A" ? "agregado" : "modificado") +
            " correctamente."
        );
      }, 0);
    }
```

Observe:

- esta función no podrá ser ejecutada hasta que en la siguiente etapa se implemente el "formulario controlado" dentro del componente ArticulosRegistro, que es quien la invocara con el parámetro adecuado; por lo que hasta este momento no funcionara la edición de los campos (inputs y selects), dejamos para más adelante completar la funcionalidad de "Agregar" y "Modificar" del ABMC.
- que se llama la función alert() con un setTimeout() de 0 milisegundos para que se ejecute luego de que "React" actualice el estado de la UI.

Funcionalidad Agregar:

Ya completando nuestro ABMC, pasamos a la función Agregar, la cual se encarga de inicializar el estado del componente para que se muestre el componente **ArticulosRegistro**, con los campos vacíos para que el usuario pueda ingresar los datos del nuevo registro.

No haremos cambio a está función pero si, observamos algunos aspectos de la misma.

```
      async function Agregar() {
        setAccionABMC("A");
```

```
          setItem({
              IdArticulo: 0,
              Nombre: '',
              Precio: '',
              Stock: '',
              CodigoDeBarra: '',
              IdCategoria: '',
              FechaAlta: moment(new Date()).format("YYYY-MM-DD"),
              Activo: true,
            });

            // ya podemos comentar este alert que era solo para el boceto
inicial
          // alert("preparando el Alta...");
        }
```

Observe:

- que se inicializa la propiedad Fecha de Item con la fecha actual.
- que se inicializa la propiedad Activo de Item en true y como veremos más adelante este campo es solo de lectura, ya que la única forma que permitimos modificarlo es a través de la función ActivarDesactivar().
- el uso de la librería "moment" para el manejo de fechas, aqui es importante tener en cuenta que los input type date esperan una fecha en formato YYYY-MM-DD sin hora.
- que la función Alert del browser bloquea la ejecución del código (es síncrona) y hasta que no cerremos está ventana, React no podrá actualizar la interface visual. Este mensaje solo es para entender cómo funciona el código, por eso aquí ya lo podemos comentar !

Etapa 6
Componente ArticulosRegistro

A continuación trabajaremos sobre el componente **ArticulosRegistro**, el cual es el encargado de mostrar el formulario para Consultar, Agregar y/o Modificar un registro. La librería que usaremos para facilitar el enlace de los campos del formulario con el estado del componente es react-hook-form, la cual nos permite manejar los estados de los campos del formulario de una manera más sencilla y sin tener que escribir tanto código para cada campo. Para instalar la librería ejecutamos el comando:

```
npm install react-hook-form
```

** Para más informacion sobre esta libreria, ver: https://react-hook-form.com/

Lo primero que necesitamos es importar la librería en el componente ArticulosRegistro.jsx:

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

Luego mediante un hook personalizado que nos ofrece la librería, obtenemos un objeto con las propiedades register, handleSubmit y formState, que nos permitirán manejar los estados de los campos del formulario e implementar una versión sencilla de formularios controlados. El código del hook es el siguiente:

```
const {
    register,
    handleSubmit,
    formState: { errors, touchedFields, isValid, isSubmitted },
  } = useForm({ values: Item });
```

Observe:

- que se le pasa como parámetro al hook el objeto Item, el cual es un estado del componente Articulos que se recibe por parámetro y que contiene un objeto correspondiente al registro a mostrar en el formulario.
- Las propiedades del objeto formState que se obtienen del hook, nos permitirán implementar la validación de los datos ingresados por el usuario, tema que se verá más adelante.

El paso siguiente es implementar la función handleSubmit, la cual es una función que recibe como parámetro una función que se ejecutará cuando el usuario haga click en el botón Grabar, y que recibe como parámetro el estado del formulario, el cual es un objeto con los campos del formulario y sus valores. El código de la función será:

```
const onSubmit = (data) => {
```

```
    Grabar(data);
  };
```

y la misma será invocada en el evento onSubmit del formulario html: reemplazar:

```
<form>
```

por:

```
<form onSubmit={handleSubmit(onSubmit)}>
```

y este evento del formulario será invocado cuando el usuario haga click en el botón Grabar, ya que el mismo es de tipo submit.

Luego en el formulario, en cada campo se le pasa como parámetro el atributo register, el cual es una función que recibe como parámetro el nombre del campo y modifica al mismo, enlazandolo con un estado interno del componente.
A continuación se muestra como ejemplo el campo Nombre: reemplazar:

```
<input
  type="text"
  name="Nombre"
  value={Item?.Nombre}
  autoFocus
  className="form-control"
/>
```

por:

```
<input
  type="text"
  {...register("Nombre")}
  autoFocus
  className="form-control"
/>
```

Observe:

- que hemos eliminado la propiedad name del input, ya que esta propiedad ya es manejada internamente por la función register.

- que hemos eliminado la propiedad value que en el boceto inicial habíamos completado mediante interpolación y que ahora será manejada por la función register. Y considerar también que la función register también controla el evento onChange del control de ingreso de datos.

Habiendo hecho todos los cambios sugeridos el código completo del componente debería ser el siguiente:

```
import React from "react";
import { useForm } from "react-hook-form";


export default function ArticulosRegistro({
  AccionABMC,
  Categorias,
  Item,
  Grabar,
  Volver,
}) {
  const {
    register,
    handleSubmit,
    formState: { errors, touchedFields, isValid, isSubmitted },
  } = useForm({ values: Item });
  const onSubmit = (data) => {
    Grabar(data);
  };
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div className="container-fluid">


        <fieldset disabled={AccionABMC === "C"}>


          {/* campo nombre */}
```

```
          <div className="row">
            <div className="col-sm-4 col-md-3 offset-md-1">
              <label className="col-form-label" htmlFor="Nombre">
                Nombre<span className="text-danger">*</span>:
              </label>
            </div>
            <div className="col-sm-8 col-md-6">
              <input
                type="text"
                {...register("Nombre")}
                autoFocus
                className="form-control "
              />
            </div>
          </div>

          {/* campo Precio */}
          <div className="row">
            <div className="col-sm-4 col-md-3 offset-md-1">
              <label className="col-form-label" htmlFor="Precio">
                Precio<span className="text-danger">*</span>:
              </label>
            </div>
            <div className="col-sm-8 col-md-6">
              <input
                type="number"
                step=".01"
                {...register("Precio")}
                className= "form-control"
              />
            </div>
```

```
        </div>

        {/* campo Stock */}
        <div className="row">
          <div className="col-sm-4 col-md-3 offset-md-1">
            <label className="col-form-label" htmlFor="Stock">
              Stock<span className="text-danger">*</span>:
            </label>
          </div>
          <div className="col-sm-8 col-md-6">
            <input
              type="number"
              {...register("Stock")}
              className="form-control"
            />
          </div>
        </div>

        {/* campo CodigoDeBarra */}
        <div className="row">
          <div className="col-sm-4 col-md-3 offset-md-1">
            <label className="col-form-label" htmlFor="CodigoDeBarra">
              Codigo De Barra<span className="text-danger">*</span>:
            </label>
          </div>
          <div className="col-sm-8 col-md-6">
            <input
              type="text"
              {...register("CodigoDeBarra")}
              className="form-control"
            />
```

```
          </div>
        </div>

        {/* campo IdCategoria*/}
        <div className="row">
          <div className="col-sm-4 col-md-3 offset-md-1">
            <label className="col-form-label" htmlFor="IdCategoria">
              Categoria<span className="text-danger">*</span>:
            </label>
          </div>
          <div className="col-sm-8 col-md-6">
            <select
              {...register("IdCategoria")}
              className="form-control"
            >
              <option value="" key={1}></option>
              {Categorias?.map((x) => (
                <option value={x.IdCategoria} key={x.IdCategoria}>
                  {x.Nombre}
                </option>
              ))}
            </select>
          </div>
        </div>

        {/* campo FechaAlta */}
        <div className="row">
          <div className="col-sm-4 col-md-3 offset-md-1">
            <label className="col-form-label" htmlFor="FechaAlta">
              Fecha Alta<span className="text-danger">*</span>:
            </label>
```

```
            </div>
            <div className="col-sm-8 col-md-6">
              <input
                type="date"
                {...register("FechaAlta")}
                className="form-control"
              />
            </div>
          </div>

          {/* campo Activo */}
          <div className="row">
            <div className="col-sm-4 col-md-3 offset-md-1">
              <label className="col-form-label" htmlFor="Activo">
                Activo<span className="text-danger">*</span>:
              </label>
            </div>
            <div className="col-sm-8 col-md-6">
              <select
                {...register("Activo")}
                className="form-control"
                disabled
              >
                <option value={null}></option>
                <option value={false}>NO</option>
                <option value={true}>SI</option>
              </select>
            </div>
          </div>

        </fieldset>
```

```
        {/* Botones Grabar, Cancelar/Volver' */}
        <hr />
        <div className="row justify-content-center">
          <div className="col text-center botones">
            {AccionABMC !== "C" && (
              <button type="submit" className="btn btn-primary">
                <i className="fa fa-check"></i> Grabar
              </button>
            )}
            <button
              type="button"
              className="btn btn-warning"
              onClick={() => Volver()}
            >
              <i className="fa fa-undo"></i>
              {AccionABMC === "C" ? " Volver" : " Cancelar"}
            </button>
          </div>
        </div>

        {/* texto: Revisar los datos ingresados... */}
        <div className="row alert alert-danger mensajesAlert">
          <i className="fa fa-exclamation-sign"></i>
          Revisar los datos ingresados...
        </div>

      </div>
    </form>
  );
}
```

Si probamos la aplicación ya estaría funcionando correctamente el "Alta" y "Modificacion" de los articulos que nos había quedado inconcluso en la etapa anterior

---

Validaciones:

Si intentamos dar de alta de un registro, con algún error, por ej: sin completar los datos obligatorios, recibiremos un error de validación en el backend, que en el método "Grabar" del componente "Articulos" se encuentra dentro de un bloque "try/catch". En el catch se captura el error y se muestra un alert con el mensaje de error.

```
async function Grabar(item) {
  // agregar o modificar
  try
  {
    await articulosService.Grabar(item);
  }
  catch (error)
  {
    alert(error?.response?.data?.message ?? error.toString())
    return;
  }
  await Buscar();
  Volver();

  setTimeout(() => {
    alert(
      "Registro " +
        (AccionABMC === "A" ? "agregado" : "modificado") +
        " correctamente."
    );
  }, 0);
}
```

Las validaciones como estas, del lado del backend son fundamentales y no se deben omitir, pero también es importante realizar validaciones en el frontend, para una mejor la experiencia del usuario. Para esto seguiremos usando "react-hook-form" que nos permite realizar este tipo de validaciones.

Para nuestro ejemplo necesitamos 2 cambios en cada etiqueta (input, textarea o select) de ingreso de datos:

1. Agregar a la propiedad register que ya venimos usando para indicar el nombre del estado a enlazar un segundo parámetro en donde indicaremos qué validaciones se aplicarán a ese campo. Usaremos los validadores más simples: required, minLength, maxLength, pattern, min, max.
2. Agregar junto a la etiqueta de ingreso de datos, un div con la clase "invalid-feedback" para mostrar el mensaje de error de validación. El mismo se mostrará sólo cuando el campo no pase la validación.

Para el input "Nombre" reemplazar:

```
<input
  type="text"
  {...register("Nombre")}
  autoFocus
  className="form-control "
/>
```

por:

```
<input
  type="text"
  {...register("Nombre", {
    required: { value: true, message: "Nombre es requerido" },
    minLength: {
      value: 5,
      message: "Nombre debe tener al menos 5 caracteres",
    },
    maxLength: {
      value: 60,
      message: "Nombre debe tener como máximo 60 caracteres",
    },
  })}
  autoFocus
  className={
    "form-control " + (errors?.Nombre ? "is-invalid" : "")
  }
/>
{errors?.Nombre && touchedFields.Nombre && (
  <div className="invalid-feedback">
```

```
    {errors?.Nombre?.message}
  </div>
)}
```

Para el input "Precio" reemplazar:

```
<input
  type="number"
  step=".01"
  {...register("Precio")}
  className= "form-control"
/>
```

por:

```
<input
  type="number"
  step=".01"
  {...register("Precio", {
    required: { value: true, message: "Precio es requerido" },
    min: {
      value: 0.01,
      message: "Precio debe ser mayor a 0",
    },
    max: {
      value: 99999.99,
      message: "Precio debe ser menor o igual a 99999.99",
    },
  })}
  className={
    "form-control " + (errors?.Precio ? "is-invalid" : "")
  }
/>
<div className="invalid-feedback">{errors?.Precio?.message}</div>
```

Para el input "Stock" reemplazar:

```
<input
  type="number"
  {...register("Stock")}
  className= "form-control"
/>
```

por:

```
<input
  type="number"
  {...register("Stock", {
    required: { value: true, message: "Stock es requerido" },
    min: {
      value: 0,
      message: "Stock debe ser mayor a 0",
    },
    max: {
      value: 99999,
      message: "Stock debe ser menor o igual a 999999",
    },
  })}
  className={
    "form-control " + (errors?.Stock ? "is-invalid" : "")
  }
/>
<div className="invalid-feedback">{errors?.Stock?.message}</div>
```

Para el input "CodigoDeBarra" reemplazar:

```
<input
  type="text"
  {...register("CodigoDeBarra")}
  className="form-control"
```

```
/>
```

por:

```
<input
  type="text"
  {...register("CodigoDeBarra", {
    required: {
      value: true,
      message: "Codigo De Barra es requerido",
    },
    pattern: {
      value: /^[0-9]{13}$/,
      message:
        "Codigo De Barra debe ser un número, de 13 dígitos",
    },
  })}
  className={
    "form-control" + (errors?.CodigoDeBarra ? " is-invalid" : "")
  }
/>
<div className="invalid-feedback">
  {errors?.CodigoDeBarra?.message}
</div>
```

Para el input "IdCategoria" reemplazar:

```
<select
  {...register("IdCategoria")}
  className="form-control"
>
  <option value="" key={1}></option>
  {Categorias?.map((x) => (
    <option value={x.IdCategoria} key={x.IdCategoria}>
```

```
      {x.Nombre}
    </option>
  ))}
</select>
```

por:

```
<select
  {...register("IdCategoria", {
    required: { value: true, message: "Categoria es requerido" },
  })}
  className={
    "form-control " +
    (errors?.IdCategoria ? "is-invalid" : "")
  }
>
  <option value="" key={1}></option>
  {Categorias?.map((x) => (
    <option value={x.IdCategoria} key={x.IdCategoria}>
      {x.Nombre}
    </option>
  ))}
</select>
<div className="invalid-feedback">
  {errors?.IdCategoria?.message}
</div>
```

para el input "FechaAlta" reemplazar:

```
<input
  type="date"
  {...register("FechaAlta")}
  className="form-control"
/>
```

por:

```
<input
  type="date"
  {...register("FechaAlta", {
    required: { value: true, message: "Fecha Alta es requerido" }
  })}
  className={
    "form-control " + (errors?.FechaAlta ? "is-invalid" : "")
  }
/>
<div className="invalid-feedback">
  {errors?.FechaAlta?.message}
</div>
```

Observe:

- que no ponemos validaciónes al campo Activo, debido a que es de solo lectura. (disabled)

Finalmente al último del formulario, teníamos un div con el mensaje "Revisar los datos ingresados..." que se mostraba siempre, pero ahora solo queremos que se muestre cuando el usuario intente grabar y no haya pasado las validaciones. Para esto, la librería nos ofrece 2 estados: "isValid" y "isSubmitted". El primero se inicializa en true y se actualiza en el evento "onSubmit" del formulario, con el resultado de la validación del formulario. El segundo se inicializa en false y se actualiza en el evento "onSubmit" del formulario, con el valor true. Finalmente, en el div con el mensaje "Revisar los datos ingresados..." agregaremos una condición para que se muestre solo cuando "isValid" sea false y "isSubmitted" sea true.

reemplazar:

```
<div className="row alert alert-danger mensajesAlert">
  <i className="fa fa-exclamation-sign"></i>
  Revisar los datos ingresados...
</div>
```

por:

```
      {!isValid && isSubmitted && (
        <div className="row alert alert-danger mensajesAlert">
          <i className="fa fa-exclamation-sign"></i>
          Revisar los datos ingresados...
        </div>
      )}
```

Si probamos la aplicación ya estaría funcionando correctamente las validaciones de los campos del formulario, ahora también del lado del cliente!

El código completo del componente **ArticulosRegitros** quedaria asi:

```
import React from "react";
import { useForm } from "react-hook-form";


export default ArticulosRegitro({
  AccionABMC,
  Categorias,
  Item,
  Grabar,
  Volver,
}) {
  const {
    register,
    handleSubmit,
    formState: { errors, touchedFields, isValid, isSubmitted },
  } = useForm({ values: Item });


  const onSubmit = (data) => {
    Grabar(data);
  };


  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div className="container-fluid">
```

```
        <fieldset disabled={AccionABMC === "C"}>


          {/* campo nombre */}
          <div className="row">
            <div className="col-sm-4 col-md-3 offset-md-1">
              <label className="col-form-label" htmlFor="Nombre">
                Nombre<span className="text-danger">*</span>:
              </label>
            </div>
            <div className="col-sm-8 col-md-6">
              <input
                type="text"
                {...register("Nombre", {
                  required: { value: true, message: "Nombre es requerido" },
                  minLength: {
                    value: 5,
                    message: "Nombre debe tener al menos 5 caracteres",
                  },
                  maxLength: {
                    value: 60,
                    message: "Nombre debe tener como máximo 60 caracteres",
                  },
                })}
                autoFocus
                className={
                  "form-control " + (errors?.Nombre ? "is-invalid" : "")
                }
              />
              {errors?.Nombre && touchedFields.Nombre && (
                <div className="invalid-feedback">
                  {errors?.Nombre?.message}
                </div>
              )}
            </div>
          </div>


          {/* campo Precio */}
          <div className="row">
```

```
            <div className="col-sm-4 col-md-3 offset-md-1">
              <label className="col-form-label" htmlFor="Precio">
                Precio<span className="text-danger">*</span>:
              </label>
            </div>
            <div className="col-sm-8 col-md-6">
              <input
                type="number" step=".01"
                {...register("Precio", {
                  required: { value: true, message: "Precio es requerido" },
                  min: {
                    value: 0.01,
                    message: "Precio debe ser mayor a 0",
                  },
                  max: {
                    value: 99999.99,
                    message: "Precio debe ser menor o igual a 99999.99",
                  },
                })}
                className={
                  "form-control " + (errors?.Precio ? "is-invalid" : "")
                }
              />
              <div className="invalid-feedback">{errors?.Precio?.message}</div>
            </div>
          </div>

          {/* campo Stock */}
          <div className="row">
            <div className="col-sm-4 col-md-3 offset-md-1">
              <label className="col-form-label" htmlFor="Stock">
                Stock<span className="text-danger">*</span>:
              </label>
            </div>
            <div className="col-sm-8 col-md-6">
              <input
                type="number"
                {...register("Stock", {
```

```
                required: { value: true, message: "Stock es requerido" },
                min: {
                  value: 0,
                  message: "Stock debe ser mayor a 0",
                },
                max: {
                  value: 99999,
                  message: "Stock debe ser menor o igual a 999999",
                },
              })}
              className={
                "form-control " + (errors?.Stock ? "is-invalid" : "")
              }
            />
            <div className="invalid-feedback">{errors?.Stock?.message}</div>
          </div>
        </div>

        {/* campo CodigoDeBarra */}
        <div className="row">
          <div className="col-sm-4 col-md-3 offset-md-1">
            <label className="col-form-label" htmlFor="CodigoDeBarra">
              Codigo De Barra<span className="text-danger">*</span>:
            </label>
          </div>
          <div className="col-sm-8 col-md-6">
            <input
              type="text"
              {...register("CodigoDeBarra", {
                required: {
                  value: true,
                  message: "Codigo De Barra es requerido",
                },
                pattern: {
                  value: /^[0-9]{13}$/,
                  message:
                    "Codigo De Barra debe ser un número, de 13 dígitos",
                },
```

```
            })}
            className={
              "form-control" + (errors?.CodigoDeBarra ? " is-invalid" : "")
            }
          />
          <div className="invalid-feedback">
            {errors?.CodigoDeBarra?.message}
          </div>
        </div>
      </div>

      {/* campo IdCategoria */}
      <div className="row">
        <div className="col-sm-4 col-md-3 offset-md-1">
          <label className="col-form-label" htmlFor="IdCategoria">
            Categoria<span className="text-danger">*</span>:
          </label>
        </div>
        <div className="col-sm-8 col-md-6">
          <select
            {...register("IdCategoria", {
              required: { value: true, message: "Categoria es requerido" },
            })}
            className={
              "form-control " +
              (errors?.IdCategoria ? "is-invalid" : "")
            }
          >
            <option value="" key={1}></option>
            {Categorias?.map((x) => (
              <option value={x.IdCategoria} key={x.IdCategoria}>
                {x.Nombre}
              </option>
            ))}
          </select>
          <div className="invalid-feedback">
            {errors?.IdCategoria?.message}
          </div>
```

```
          </div>
        </div>

        {/* campo FechaAlta */}
        <div className="row">
          <div className="col-sm-4 col-md-3 offset-md-1">
            <label className="col-form-label" htmlFor="FechaAlta">
              Fecha Alta<span className="text-danger">*</span>:
            </label>
          </div>
          <div className="col-sm-8 col-md-6">
            <input
              type="date"
              {...register("FechaAlta", {
                required: { value: true, message: "Fecha Alta es requerido" }
              })}
              className={
                "form-control " + (errors?.FechaAlta ? "is-invalid" : "")
              }
            />
            <div className="invalid-feedback">
              {errors?.FechaAlta?.message}
            </div>
          </div>
        </div>

        {/* campo Activo */}
        <div className="row">
          <div className="col-sm-4 col-md-3 offset-md-1">
            <label className="col-form-label" htmlFor="Activo">
              Activo<span className="text-danger">*</span>:
            </label>
          </div>
          <div className="col-sm-8 col-md-6">
            <select
              name="Activo"
              {...register("Activo", {
                required: { value: true, message: "Activo es requerido" },
```

```
            })}
            className={
              "form-control" + (errors?.Activo ? " is-invalid" : "")
            }
            disabled
          >
            <option value={null}></option>
            <option value={false}>NO</option>
            <option value={true}>SI</option>
          </select>
          <div className="invalid-feedback">{errors?.Activo?.message}</div>
        </div>
      </div>

    </fieldset>

    {/* Botones Grabar, Cancelar/Volver' */}
    <hr />
    <div className="row justify-content-center">
      <div className="col text-center botones">
        {AccionABMC !== "C" && (
          <button type="submit" className="btn btn-primary">
            <i className="fa fa-check"></i> Grabar
          </button>
        )}
        <button
          type="button"
          className="btn btn-warning"
          onClick={() => Volver()}
        >
          <i className="fa fa-undo"></i>
          {AccionABMC === "C" ? " Volver" : " Cancelar"}
        </button>
      </div>
    </div>

    {/* texto: Revisar los datos ingresados... */}
    {!isValid && isSubmitted && (
```

```
          <div className="row alert alert-danger mensajesAlert">
            <i className="fa fa-exclamation-sign"></i>
            Revisar los datos ingresados...
          </div>
        )}


      </div>
    </form>
  );
}
```

Etapa 7

Servicio/Componente ModalDialog

Vamos a implementar un servicio y un componente asociado, que nos permitirá ejecutar funcionalidad desde los diferentes componentes y/o servicios de nuestra aplicación.

El mismo ofrecerá 3 funcionalidades:

a) Alert(): similar al alert() de javascript pero con bootstrap y en modo asíncrono

b) Confirm(): similar al confirm() de javascript pero con bootstrap y en modo asíncrono

c) Bloquear/DesbloquearPantalla(): formulario modal que evitará que un usuario del sistema llame 2 veces a la misma acción, pensando que no funcionó. Para lo cual se bloqueará la interface html hasta que se complete dicha acción. Un caso típico, que evitará, será que se clickee 2 veces el botón grabar porque el servidor está lento y el usuario no tiene una retroalimentación de que la acción ya está en curso y tiene que esperar a que termine su ejecución.

En está etapa haremos uso de una de las tantas librerías de componentes de interfaz de usuario que ofrece el mercado, en particular usaremos react-bootstrap que combina código react con una interface con boostrap, podemos ver todo lo que ofrece está libreria en https://react-bootstrap.github.io/

Para instalarla usaremos el siguiente comando:

```
npm install react-bootstrap
```

Comenzamos entonces a desarrollar nuestro componente y servicio:

1. Crearemos la interface visual que estará asociada a nuestro servicio, para lo cual crearemos el componente: components/ModalDialog.jsx. Le asignaremos el siguiente código:

```
import React, { useState, useEffect } from "react";
import Modal from "react-bootstrap/Modal";
import modalDialogService from "../services/modalDialog.service";


function ModalDialog() {
  const [mensaje, setMensaje] = useState("");
  const [titulo, setTitulo] = useState("");
  const [boton1, setBoton1] = useState("");
  const [boton2, setBoton2] = useState("");
  const [accionBoton1, setAccionBoton1] = useState(null);
  const [accionBoton2, setAccionBoton2] = useState(null);
  const [tipo, setTipo] = useState("");
```

```
const handleAccionBoton1 = () => {
  if (accionBoton1) {
    accionBoton1();
  }
  setMensaje((x) => (x = ""));
};
const handleAccionBoton2 = () => {
  if (accionBoton2) {
    accionBoton2();
  }
  setMensaje((x) => (x = ""));
};


const handleClose = () => {
  setMensaje((x) => (x = ""));
};


function Show(
  // cuidado en esta funcion cuando se invoca desde el servicio modalDialogService
  //   NO tiene las variables de state del componente, ej mensaje, titulo, boton1....
  //   pero SI a las funciones setMensaje, setTitulo, setBoton1....
  _mensaje,
  _titulo,
  _boton1,
  _boton2,
  _accionBoton1,
  _accionBoton2,
  _tipo
) {
  setMensaje((x) => (x = _mensaje));
  setTitulo((x) => (x = _titulo));
  setBoton1((x) => (x = _boton1));
  setBoton2((x) => (x = _boton2));
  setAccionBoton1(() => _accionBoton1);
  setAccionBoton2(() => _accionBoton2);
  setTipo((x) => (x = _tipo));
}


useEffect(() => {
  //suscribirse al servicio modalDialogService al iniciar el componente
  modalDialogService.subscribeShow(Show);
  return () => {
    //desuscribirse al servicio modalDialogService al desmontar el componente
    modalDialogService.subscribeShow(null);
  };
}, []);


let classHeader = "";
```

```
let faIcon = "";
switch (tipo) {
  case "success":
    classHeader = "bg-success";
    faIcon = "fa-regular fa-circle-check";
    break;
  case "danger":
    classHeader = "bg-danger";
    faIcon = "fa-solid fa-triangle-exclamation";
    break;
  case "info":
    classHeader = "bg-info";
    faIcon = "fa-solid fa-circle-info";
    break;
  case "warning":
    classHeader = "bg-warning";
    faIcon = "fa-solid fa-triangle-exclamation";
    break;
  default:
    classHeader = "bg-success";
    break;
}


if (mensaje === "") return null;


return (
  <>
    <Modal
      show
      onHide={handleClose}
      backdrop="static"
      keyboard={mensaje === "BloquearPantalla" ? false : true}
    >
      <Modal.Header
        className={classHeader}
        closeButton={mensaje !== "BloquearPantalla"}
      >
        <Modal.Title>{titulo}</Modal.Title>
      </Modal.Header>


      <Modal.Body style={{ fontSize: "1.2em" }}>
        {mensaje === "BloquearPantalla" ? (
          <div className="progress">
            <div
              className="progress-bar progress-bar-striped progress-bar-animated"
              role="progressbar"
              aria-valuenow="100"
              aria-valuemin="0"
              aria-valuemax="100"
              style={{ flex: 1 }}
```

```
            ></div>
          </div>
        ) : (
          <p>
            <i
              style={{ fontSize: "1.6em", margin: "0.5em" }}
              className={faIcon}
            ></i>
            {mensaje}
          </p>
        )}
      </Modal.Body>


      <Modal.Footer>
        {boton1 !== "" && (
          <button
            type="button"
            className="btn btn-primary"
            onClick={handleAccionBoton1}
          >
            {boton1}
          </button>
        )}
        {boton2 !== "" && (
          <button
            type="button"
            className="btn btn-secondary"
            onClick={handleAccionBoton2}
          >
            {boton2}
          </button>
        )}
      </Modal.Footer>
    </Modal>
  </>
 );
}


export { ModalDialog};
```

Observe:

- que la librería “react-bootstrap” ofrece un extenso conjunto de componentes visuales, en nuestro caso comenzaremos a usar uno de ellos llamado Modal y la funcionalidad que ofrece es la de formulario modal (formulario que no permite iteracion con el resto de la pagina hasta que este se cierre) que usaremos como contenedor de nuestro componente ModalDialog.

2. Crearemos un servicio llamado services/modalDialog.service.js y le

asignaremos el siguiente código:

```
let ModalDialog_Show = null;  //apunta a la funcion show del componente ModalDialog


const Alert = (
  _mensaje,
  _titulo = "Atención",
  _boton1 = "Aceptar",
  _boton2 = "",
  _accionBoton1 = null,
  _accionBoton2 = null,
  _tipo = 'info'
) => {
  if (ModalDialog_Show)
    ModalDialog_Show(
      _mensaje,
      _titulo,
      _boton1,
      _boton2,
      _accionBoton1,
      _accionBoton2,
      _tipo
    );
};


const Confirm = (
  _mensaje,
  _titulo = "Confirmar",
  _boton1 = "Aceptar",
  _boton2 = "Cancelar",
  _accionBoton1 = null,
  _accionBoton2 = null,
  _tipo = 'warning'
) => {
  if (ModalDialog_Show)
    ModalDialog_Show(
      _mensaje,
      _titulo,
      _boton1,
      _boton2,
      _accionBoton1,
      _accionBoton2,
      _tipo
    );
};


let cntBloquearPantalla = 0;
const BloquearPantalla = (blnBloquear) => {
```

```
  if (blnBloquear) {
    cntBloquearPantalla++;
  } else {
    cntBloquearPantalla--;
  }
  if (ModalDialog_Show) {
    if (cntBloquearPantalla === 1) {
      ModalDialog_Show(
        "BloquearPantalla",
        "Espere por favor...",
        "",
        "",
        null,
        null,
        'info'
      );
    }
    if (cntBloquearPantalla === 0) {
      ModalDialog_Show("", "", "", "", null, null);
    }
  }
};


const subscribeShow = (_ModalDialog_Show) => {
  ModalDialog_Show = _ModalDialog_Show;
};


const modalDialogService = { Alert, Confirm, BloquearPantalla, subscribeShow };


export default modalDialogService;
```

Observe:

- La funcionalidad Alert
- La funcionalidad Confirm
- La funcionalidad BloquearPantalla

3. Terminando la configuración y para dejar disponible el componente ModalDialog que será manipulado mediante el servicio ModalDialogService, importamos en el componente App.jsx, con la siguiente instrucción en las primeras líneas del mismo:

```
import { ModalDialog } from "./components/ModalDialog";
```

4. Luego lo agregamos al html de nuestro componente App.jsx, poniéndolo arriba del componente “Menu”

```
...
          <ModalDialog/>
          <Menu />
...
```

Ahora empezaremos a hacer uso del servicio y componente recién definidos, en el componente **Articulos**, importamos el servicio:

```
import modalDialogService from "../../services/modalDialog.service";
```

- reemplazamos todos los alert de javascript por los la función Alert provista por el nuevo servicio, por ej:

reemplazar:

alert("No puede modificarse un registro Inactivo.");

por:

modalDialogService.Alert("No puede modificarse un registro Inactivo.");

- reemplazamos los confirm de javascript por la función Confirm provista por el nuevo servicio, haciendo las adaptaciones necesarias; debido a que confirm de js es sincrono y el nuevo Confirm es asincrono y espera funciones callback para ejecutar la acciones definidas. por ej:

reemplazar:

```
const resp = window.confirm(
  "Esta seguro que quiere " +
    (item.Activo ? "desactivar" : "activar") +
    " el registro?"
);
if (resp) {
  await articulosService.ActivarDesactivar(item);
  await Buscar();
}
```

por:

```
modalDialogService.Confirm(
      "Esta seguro que quiere " +
        (item.Activo ? "desactivar" : "activar") +
        " el registro?",
      undefined,
      undefined,
      undefined,
      async () => {
        await articulosService.ActivarDesactivar(item);
        await Buscar();
      }
    );
```

- Usar la función BloquearPantalla() del servicio para evitar que se llame más de una vez a la misma acción por error . En este caso la usaremos para el botón Buscar del FormBusqueda que trae los datos del servidor. Para lo cual haga la siguientes modificaciones en la función Buscar() del componente Articulos.jsx

```
      async function Buscar(_pagina) {
          if (_pagina && _pagina !== Pagina) {
            setPagina(_pagina);
          }
           // OJO Pagina (y cualquier estado...) se actualiza para el proximo
render, para buscar usamos el parametro _pagina
          else {
            _pagina = Pagina;
          }
          modalDialogService.BloquearPantalla(true);
```

```
            const data = await articulosService.Buscar(Nombre, Activo,
_pagina);
          modalDialogService.BloquearPantalla(false);

    setItems(data.Items);
          setRegistrosTotal(data.RegistrosTotal);

          //generar array de las paginas para mostrar en select del paginador
          const arrPaginas = [];
          for (let i = 1; i <= Math.ceil(data.RegistrosTotal / 10); i++) {
            arrPaginas.push(i);
          }
          setPaginas(arrPaginas);
        }
```

Ahora iremos un paso más allá implementado el BloquearPantalla para que se ejecute siempre que ejecutemos una comunicación con el servidor (llamada ajax). Lo hacemos mediante un interceptor que nos provee la librería Axios. Para cumplir nuestro objetivo vamos a crear un servicio que nos proveerá la funcionalidad de la librería Axios con la configuración necesaria para que antes de cada petición ajax y después de la misma invoque a la funcionalidad BloquearPantalla. Entonces vamos a crear el servicio: services/http.service.js, con el siguiente código:

```
import axios from "axios";
import modalService from "./modalDialog.service";

const httpService = axios.create({
  headers: {
    "Content-type": "application/json",
  },
});
```

```
httpService.interceptors.request.use(
  (request) => {
    modalService.BloquearPantalla(true);
    const accessToken = sessionStorage.getItem("accessToken");
    if (accessToken) {
      request.headers["Authorization"] = "Bearer " + accessToken;
    }
    return request;
  },
  (error) => {
    console.log("error en axios request", error);
    return Promise.reject(error);
  }
);

httpService.interceptors.response.use(
  (response) => {
    modalService.BloquearPantalla(false);
    return response;
  },
  (error) => {
    // loguear el error
    console.log("error en axios response ", error);
    modalService.BloquearPantalla(false);

    if (error.response.status === 401) {
      // no auntenticado
      error.message = "debe loguearse para acceder a esta funcionalidad";
```

```
    } else if (error.response.status === 403) {
      // no auntenticado
            error.message = "usuario no autorizado para acceder a esta
funcionalidad";
    } else {
      error.message =
        error?.response?.data?.message ??
         "Actualmente tenemos inconvenientes en el servidor, por favor intente
más tarde";
    }
    modalService.Alert(error.message);

    return Promise.reject(error);

    //return error
    //throw new Error(error?.response?.data?.Message ?? 'Ocurrio un error');
  }
);

export default httpService;
```

Observe:

- Se hace uso de una funcionalidad ofrecida por axios llamada interceptor
- Invocamos BloquearPantalla(true) antes de cada petición, para bloquear la pantalla.
- Invocamos BloquearPantalla(false) después de cada petición (tanto para las respuestas correctas o con error), para desbloquear la pantalla.
- Estamos incluyendo algunos elementos de seguridad que en la próxima etapa terminaremos de implementar.

Finalmente en nuestro código debemos reemplazar el uso habitual que hacíamos de la librería Axios, por este nuevo servicio que es análogo salvo que agrega la funcionalidad de BloquearPantalla en forma automática.

Implica 2 cambios: 1) importar el nuevo servicio http.service y 2) reemplazar el uso de la librería axios por este nuevo servicio que es una función envolvente de axios (wrapper)

Nuestro articulos.service.js quedaría de la siguiente manera:

```
import httpService from "./http.service";
const urlResource =
"https://labsys.frc.utn.edu.ar/dds-backend-2025/api/articulos";
// mas adelante podemos usar un archivo de configuración para el urlResource
// import {config} from "../config";
// const urlResource = config.urlResourceArticulos;



async function Buscar(Nombre, Activo, Pagina) {
  const resp = await httpService.get(urlResource, {
    params: { Nombre, Activo, Pagina },
  });
  return resp.data;
}



async function BuscarPorId(item) {
  const resp = await httpService.get(urlResource + "/" + item.IdArticulo);
  return resp.data;
}



async function ActivarDesactivar(item) {
  await httpService.delete(urlResource + "/" + item.IdArticulo);
}



async function Grabar(item) {
  if (item.IdArticulo === 0) {
    await httpService.post(urlResource, item);
  } else {
    await httpService.put(urlResource + "/" + item.IdArticulo, item);
  }
}


export const articulosService = {
  Buscar,BuscarPorId,ActivarDesactivar,Grabar
};
```

Observe:

- que este servicio evita que tengamos que BloquearPantalla(true) y BloquearPantalla(false) antes de cada llamada a una webapi
- que la configuración de las url de los servidores y/o recursos suelen estar en un archivo de global de configuración denominado generalmente config.js (dentro de la carpeta /src), que en nuestro caso podría tener el siguiente contenido:

```
// opcion 1 cuando se hacen pruebas locales
//const urlServidor = "http://localhost:3000"


// opcion 2 cuando se despliega el frontend en un servidor distinto al
backend
const urlServidor = "https://labsys.frc.utn.edu.ar/dds-backend-2025"
//const urlServidor = "https://dds-backend.azurewebsites.net"
//const urlServidor = "https://webapi.pymes.net.ar"


// opcion 3 cuando se despliega el frontend, en el mismo servidor que el
backend
//const urlServidor = ""




const urlResourceArticulos = urlServidor + "/api/articulos";
const urlResourceCategorias = urlServidor + "/api/categorias";


export const config = {
    urlServidor,
    urlResourceArticulos,
    urlResourceCategorias,
}
```

Etapa 8
Seguridad JWT

En está etapa implementaremos en nuestro frontend un componente que estará protegido con seguridad mediante Json Web Token, verificando autenticación y autorización.

En nuestro backend tenemos definidos 2 usuarios con 2 roles distintos:

Usuario	Clave	Rol	Permisos
admin	123	jefe	SI debe tener permisos para el componente Usuarios (y su webapi asociada)
juan	123	empleado	NO debe tener permisos para el componente Usuarios (y su webapi asociada)

Para construir nuestro frontend nos valdremos de los siguientes elementos:

1. Un servicio que llamaremos auth.service.js, que nos ofrecerá la funcionalidad de login, logout e identificación del usuario logueado.
2. un componente login.jsx y su respectivo archivo de estilo, que serán la interface visual del servicio anterior.
3. un componente auxiliar que llamaremos RequireAuth, que nos servirá para “envolver” a los componentes de nuestra aplicación, a los cuales queramos acceder exigiendo autenticación y autorización, es decir todos los componentes que no sean públicos.

Siguiendo los puntos anteriores construiremos un ejemplo de un componente y su respectivo servicio que estén protegidos seguridad JWT

a. un componente llamdado Usuarios.jsx que carga un listado de articulos provenientes de una webapi con seguridad JWT: https://labsys.frc.utn.edu.ar/dds-backend-2025/api/usuarios
b. el servicio correspondiente al componente con seguridad: usuarios.service.js

A continuación vamos con todo el código necesario:

Código de services/auth.service.js:

```
import httpService from "./http.service";
```

```
import { config } from "../config";
import modalService from "./modalDialog.service";


const login = async (usuario, clave, navigateToComponent) => {
   let resp = await httpService.post(config.urlServidor + "/api/login",
{
    usuario,
    clave,
  });

  if (resp.data?.accessToken) {
    sessionStorage.setItem("usuarioLogueado", usuario);
    sessionStorage.setItem("accessToken", resp.data.accessToken);
    sessionStorage.setItem("refreshToken", resp.data.refreshToken);
    if (CambioUsuarioLogueado) CambioUsuarioLogueado(usuario);
    {
      //navigate("/Inicio");
     navigateToComponent();
    }

  } else {
    if (CambioUsuarioLogueado) CambioUsuarioLogueado(null);
    //alert("Usuario o clave incorrectos");
    modalService.Alert("Usuario o clave incorrectos");
  }
};


const logout = () => {
  sessionStorage.removeItem("usuarioLogueado");
```

```
  sessionStorage.removeItem("accessToken");
  sessionStorage.removeItem("refreshToken");
  if (CambioUsuarioLogueado) CambioUsuarioLogueado(null);
};

const getUsuarioLogueado = () => {
  return sessionStorage.getItem("usuarioLogueado");
};

let CambioUsuarioLogueado = null;
const subscribeUsuarioLogueado = (x) => (CambioUsuarioLogueado = x);

const AuthService = {
  login,
  logout,
  getUsuarioLogueado,
  subscribeUsuarioLogueado
};

export default AuthService;
```

Observe:

- cómo se recupera al llamar la api login el accessToken y refreshToken y se los guarda localmente para usarlo en las peticiones siguientes.
- el código hecho en etapas anteriores: “http.service.js” que ***debemos*** usar en todas las peticiones y es el que verifica si está guardado localmente el accessToken, lo envía en todas las peticiones para permitir al servidor realizar la autenticación/autorización pertinente.

- como en el método logout se eliminan los elementos de seguridad que estaban guardados localmente en el cliente web.

Código de components/login/Login.jsx:

```
import React, { useState, useEffect } from "react";
import "./Login.css"; //css global
import { useNavigate } from "react-router-dom";
import { useParams } from 'react-router-dom';
import AuthService from "../../services/auth.service";


function Login() {
  const [usuario, setUsuario] = useState("");
  const [clave, setClave] = useState("");
  const navigate = useNavigate();
  const {componentFrom} = useParams();


  const navigateToComponent = () => {
    navigate(`/${componentFrom}`);
  };


  const handleIngresar = async () => {
    //AuthService.login(usuario, clave, navigate);
    AuthService.login(usuario, clave, navigateToComponent);
  };



  useEffect(() => {
    // lo primero que hacemos al ingresar al login es desloguearnos
    // borrando los datos de sessionStorage y el state usuarioLogueado
```

```
          AuthService.logout();
        });

        return (
          <>
          <div className="divbody text-center">
            <main className="form-signin w-100 m-auto">
              <form className="p-5">
                <img
                  className="mb-4"
src="https://getbootstrap.com/docs/5.2/assets/brand/bootstrap-logo.svg"
                  alt=""
                  width="72"
                  height="57"
                />
                <h1 className="h3 mb-3 fw-normal">Por favor ingrese</h1>

                <div className="form-floating">
                  <input
                    type="text"
                    autoComplete="off"
                    placeholder="usuario"
                    onChange={(e) => setUsuario(e.target.value)}
                    value = {usuario}
                    autoFocus
                    className="form-control"
                    id="usuario"
                  />
```

```
                <label className="custom-control" for="usuario">
                  Usuario
                </label>
              </div>
              <div className="form-floating">
                <input
                 type="password"
                 autoComplete="off"
                 placeholder="Clave"
                 onChange={(e) => setClave(e.target.value)}
                 value = {clave}
                 className="form-control"
                 id="clave"

                />
                <label className="custom-control" htmlFor="clave">
                  Clave
                </label>
              </div>

              <div className="checkbox mb-3">
                <label className="custom-control">
                  <input type="checkbox" value="remember-me" /> Recordarme
                </label>
              </div>
                        <button  className="w-100  btn  btn-lg  btn-primary"
type="button"  onClick={(e) => handleIngresar()}>
                Ingresar
              </button>
```

```
            <p className="mt-5 mb-3 text-muted">© 2025</p>
          </form>
        </main>
      </div>

    </>
    );
}
export {Login};
```

Código de components/login/Login.css:

```
divbody {
  display: flex;
  align-items: center;
  padding-top: 2em;
  padding-bottom: 2em;
  background-color: #f5f5f5;
}

.form-signin {
  max-width: 500px;
  padding: 15px;
}

.form-signin .form-floating:focus-within {
  z-index: 2;
}
```

```
.form-signin input[type="email"] {
  margin-bottom: -1px;
  border-bottom-right-radius: 0;
  border-bottom-left-radius: 0;
}

.form-signin input[type="password"] {
  margin-bottom: 10px;
  border-top-left-radius: 0;
  border-top-right-radius: 0;
}
.form-signin input{
  text-transform: none;
}
```

Código de components/RequiereAuth.jsx:

```
import React from "react";
import { Navigate } from "react-router-dom";
import AuthService from "../services/auth.service";

function RequireAuth({ children }) {
  let usuarioLogueado = AuthService.getUsuarioLogueado();

  // verificar la autenticacion
  if (!usuarioLogueado) {
    // no funciona en el build
    // return <Navigate to={"/login/" + children.type.name} />;
```

```
                                return      <Navigate      to={"/login/"      +
children.type.NombreComponenteNoOfuscado} />;
        }



        // un nivel mas de seguridad seria verificar la autorizacion...
        return children;
      }

      export { RequireAuth };
```

Código del archivo config.js (lo ponemos en la misma carpeta de App.jsx):

```
      // opcion 1 cuando se hacen pruebas locales
      //const urlServidor = "http://localhost:3000"

      // opcion 2 cuando se despliega el frontend en un servidor distinto al
backend
      const urlServidor = "https://labsys.frc.utn.edu.ar/dds-backend-2025"
      //const urlServidor = "https://dds-backend.azurewebsites.net"
      //const urlServidor = "https://webapi.pymes.net.ar"

      // opcion 3 cuando se despliega el frontend, en el mismo servidor que
el backend
      //const urlServidor = ""


      const urlResourceArticulos = urlServidor + "/api/articulos";
      const urlResourceCategorias = urlServidor + "/api/categorias";
      const urlResourceUsuarios = urlServidor + "/api/usuarios";
```

```
export const config = {
    urlServidor,
    urlResourceArticulos,
    urlResourceCategorias,
    urlResourceUsuarios,
}
```

Código de components/Usuarios.jsx:

```
import React, { useState, useEffect } from "react";
import { usuariosService } from "../services/usuarios.service";



function Usuarios() {
  const tituloPagina = "Usuarios JWT (solo para jefes)";
  const [usuarios, setUsuarios] = useState(null);

  // cargar al iniciar el componente, solo una vez
  useEffect(() => {
    BuscarUsuarios();
  }, []);

  async function BuscarUsuarios() {
     try {
      let data = await usuariosService.Buscar();
      setUsuarios(data);
    } catch (error) {
      console.log("error al buscar datos en el servidor!")
```

```
    }
  }



  return (
    <>
      <div className="tituloPagina">{tituloPagina}</div>
      <table className="table table-bordered table-striped">
        <thead>
          <tr>
            <th style={{ width: "20%" }}>IdUsuario</th>
            <th style={{ width: "50%" }}>Nombre</th>
            <th style={{ width: "30%" }}>Rol</th>
          </tr>
        </thead>
        <tbody>
          {usuarios &&
            usuarios.map((item) => (
              <tr key={item.IdUsuario}>
                <td>{item.IdUsuario}</td>
                <td>{item.Nombre}</td>
                <td>{item.Rol}</td>
              </tr>
            ))}
        </tbody>
      </table>
    </>
  );
}
```

```
      Usuarios.NombreComponenteNoOfuscado = "Usuarios";

      export { Usuarios };
```

Código de services/usuarios.service.js:

```
import {config} from "../config";

import httpService from "./http.service";


const urlResource = config.urlResourceUsuarios;


async function Buscar() {

  const resp = await httpService.get(urlResource    );

  return resp.data;

}


export const usuariosService = {

  Buscar

};
```

Integrando los elementos anteriores a nuestra aplicación: a nuestro componente Menu.jsx le agregamos el link de nuevo componente Usuarios, del login/logout y la visualización del Usuario logueado, quedándonos de la siguiente manera:

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

      import { NavLink} from "react-router-dom";

      import AuthService from "../services/auth.service";


      function Menu() {

        const [usuarioLogueado, setUsuarioLogueado] = useState(
```

```
          AuthService.getUsuarioLogueado()
        );

        function CambioUsuarioLogueado(_usuarioLogueado) {
          setUsuarioLogueado(_usuarioLogueado);
        }



        useEffect(() => {
          AuthService.subscribeUsuarioLogueado(CambioUsuarioLogueado);
          return () => {
            AuthService.subscribeUsuarioLogueado(null);
          }
        }, []);

        return (
              <nav  className="navbar  navbar-dark  bg-dark  navbar-expand-md
no-print px-2">
              <a className="navbar-brand" href="#!">
                <i className="fa fa-industry"></i>
                 <i>Pymes</i>
              </a>
              <button
                className="navbar-toggler"
                type="button"
                data-bs-toggle="collapse"
                data-bs-target="#navbarSupportedContent"
                aria-controls="navbarSupportedContent"
                aria-expanded="false"
```

```
                aria-label="Toggle navigation"
              >
                <span className="navbar-toggler-icon"></span>
              </button>
                                <div   className="collapse   navbar-collapse"
id="navbarSupportedContent">
                <ul className="navbar-nav mr-auto">
                  <li className="nav-item">
                    <NavLink className="nav-link" to="/inicio">
                      Inicio
                    </NavLink>
                  </li>
                  <li className="nav-item">
                    <NavLink className="nav-link" to="/categorias">
                      Categorias
                    </NavLink>
                  </li>
                  <li className="nav-item">
                    <NavLink className="nav-link" to="/articulos">
                      Articulos
                    </NavLink>
                  </li>
                  <li className="nav-item">
                        <NavLink  className="nav-link"  title="exclusivo  para
jefes" to="/usuarios">
                      Usuarios JWT
                    </NavLink>
                  </li>
```

```
                <li className="nav-item dropdown bg-dark">
                  <a
                    className="nav-link dropdown-toggle"
                    href="#!"
                    id="navbarDropdown"
                    role="button"
                    data-bs-toggle="dropdown"
                    aria-expanded="false"
                  >
                    Informes
                  </a>
                        <ul  className="dropdown-menu  dropdown-menu-dark"
aria-labelledby="navbarDropdown">
                    <li>
                      <a className="dropdown-item" href="#!">
                        Ventas
                      </a>
                    </li>
                    <li>
                          <a  className="dropdown-item   dropdown-menu-dark"
href="#!">
                        Compras
                      </a>
                    </li>
                    <li>
                      <hr className="dropdown-divider" />
                    </li>
                    <li>
```

```
                          <a  className="dropdown-item   dropdown-menu-dark"
href="#!">
                            Libro de IVA
                          </a>
                        </li>
                      </ul>
                    </li>
                  </ul>



                  <ul className="navbar-nav ms-auto">
                    {usuarioLogueado && (
                      <li className="nav-item">
                                <a  className="nav-link"  href="#!">Bienvenido:
{usuarioLogueado}</a>
                      </li>
                    )}
                    <li className="nav-item">
                      <NavLink className="nav-link" to="/login/Inicio">
                        <span
                          className={
                            usuarioLogueado ? "text-warning" : "text-success"
                          }
                        >
                          <i
                            className={
                                    usuarioLogueado ? "fa fa-sign-out" : "fa
fa-sign-in"
                            }
```

```
                        ></i>
                      </span>
                      {usuarioLogueado ? " Logout" : " Login"}
                    </NavLink>
                  </li>
                </ul>
              </div>
            </nav>
          );
        }
        export {Menu};
```

y finalmente ajustamos en el componente raíz: App.jsx

a) agregamos los import necesarios al inicio del mismo:

```
import { Usuarios } from "./components/Usuarios";
import {RequireAuth} from "./components/RequiereAuth" ;
import { Login } from "./components/login/Login";
```

b) modificamos parcialmente el html para actualizar las rutas de navegación con los nuevos componentes: RequireAuth, Usuarios y Login; para simplificar copiamos a continuación solo el html de la etiqueta <Routes> que es donde está el cambio:

```
<Routes>
 <Route path="/inicio" element={<Inicio />} />
 <Route path="/categorias" element={<Categorias />} />
 <Route path="/articulos" element={<Articulos />} />
 <Route
   path="/usuarios"
   element={
     <RequireAuth>
       <Usuarios />
```

```
        </RequireAuth>
      }
    />
    <Route path="/login/:componentFrom" element={<Login />} />
    <Route path="*" element={<Navigate to="/inicio" replace />} />
  </Routes>
```

A continuación pruebe la aplicación, verificando:

- que si no hay usuarios logueado (se denomina acceso anónimo) no se debe permitir el acceso al recurso: Usuarios
- que si está logueado con el usuario “juan”, aún no se debe permitir el acceso al recurso del backend: “Usuarios”. Porque aunque se pasó la autenticación, no puede pasar la autorización que indica que solo los usuarios con el rol “Admin” pueden acceder a este recurso.
- que si está logueado con el usuario “admin”, si se pueda acceder al recurso: “Usuarios”.

Siguientes pasos:

- ¿Qué pasa si el usuario se autentica, está trabajando con la aplicación y le expira el token? Aquí tendríamos que detectar la respuesta de 401 y solicitar un nuevo token usando el refresh token.
- ¿Cómo asegurar que se use una clave segura, con una cantidad mínima de caracteres que incluya mayúsculas, minúsculas y signos especiales?
- ¿Cómo evitar que mediante un ataque de fuerza bruta descubran las claves de los usuarios?
- ¿Que pasa cuando un usuario logueado accede a una ruta protegida (actualmente validada con RequiereAuth), pero no cumple con la autorización?: se muestra la interface gráfica pero no se llama a la webapi. Pero aunque no permita llamar a la webapi del servidor, ni siquiera debería poder ver la interface gráfica de la página. ¿Cómo logramos esto?