1 Front-end Pistas Deportivas

Front en React para la API REST Spring de gestión de reservas de instalaciones deportivas.

1.1 Diseño de la aplicación

En este enlace puedes ver cómo hemos diseñado la aplicación.

En este enlace puedes ver el back-end.

1.2 Creación del proyecto

Creamos el proyecto y añadimos las dependencias con:

```
npm create vite@latest front-pistas-deportivas
cd front-pistas-deportivas
npm install react-router-dom axios react-bootstrap bootstrap
npm install
```

Creamos las carpetas necesarias para el proyecto:

Carpeta	Uso
pages	para las páginas
components	para los componentes
services	para la api (peticiones al back)

Buscamos cómo hacer un NavBar con react-bootstrap y de este ejemplo sacamos la nuestra:

```
import Container from 'react-bootstrap/Container';
import Nav from 'react-bootstrap/Nav';
import Navbar from 'react-bootstrap/Navbar';

function NavBar() {
  return (
    <>
      <Navbar bg="primary" data-bs-theme="dark">
        <Container>
          <Navbar.Brand href="#home">Navbar</Navbar.Brand>
          <Nav className="me-auto">
            <Nav.Link href="/">Home</Nav.Link>
          </Nav>
        </Container>
```

```
        </Navbar>
      </>
    );
}

export default NavBar;
```

Aquí tienes el componente NavBar.

Modificamos el App.js para que cargue bootstrap (tal y como viene en la documentación oficial) y además contenga el router:

```
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
import NavBar from './components/NavBar';
import HomePage from './pages/HomePage';

import 'bootstrap/dist/css/bootstrap.min.css';

/**
 * Con React Router-v7 siempre usaremos esta manera de definir rutas.
 * No obstante es compatible (hacia atrás) con la manera antigua de v5.
 */
const router = createBrowserRouter([
  {
    path: "/",
    element: <HomePage />,
  }
]);

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

export default App
```

Borramos los CSS de App.jsx, main.jsx e index.html porque queremos usar sólo react-bootstrap (por eso el import que ves en el código anterior para cargarlo).

Creamos la página HomePage.

Creamos la página LoginPage y su componente Login.

Para que el NavBar funcione bien (no haga que la página se recargue) necesitamos que esté dentro del Router, por eso ahora vamos a crear un componente “RootLayout” que será la base del resto de componentes:

```jsx

import { Outlet } from "react-router-dom";
import NavBar from "./NavBar";
import Container from "react-bootstrap/Container";
import Row from "react-bootstrap/Row";
import Col from "react-bootstrap/Col";

const RootLayout = () => {
  return (
    <>
      <NavBar />

      {/* Contenedor principal para el contenido de la página */}
      <Container className="my-4">
        <Outlet />
      </Container>

      {/* Footer */}
      <footer className="bg-dark text-white py-3 mt-auto">
        <Container>
          <Row>
            <Col className="text-center">
              &copy; {new Date().getFullYear()} IES Virgen del Carmen.
                  CFGS Desarrollo de Aplicaciones Multiplataforma.
            </Col>
          </Row>
        </Container>
      </footer>
    </>
  );
};

export default RootLayout;
```

Ya podemos empezar con el sistema de Login.

1.3 El sistema de Login

Para el sistema de login vamos a necesitar crear, por un lado un par de servicios que se encarguen de comunicarse con el backend, obtener el token JWT y almacenarlo en el cliente, y por otro el servicio que se encargue de autenticar todas las peticiones gracias al token almacenado.

Adicionalmente necesitamos un componente `Login.jsx` que es un formulario de inicio de sesión que permite a los usuarios autenticarse en la aplicación. Este componente se encuentra dentro de la página `LoginPage.jsx`, la cual es una de las rutas definidas en el enrutador principal de la aplicación en `App.jsx`.

1.3.1 Servicio `api.js`

El archivo `api.js` se encarga de configurar una instancia de Axios para realizar peticiones HTTP a la API de la aplicación. Este archivo se encuentra en la ruta `src/services/api.js`.

1.3.1.1 Configuración de Axios Se importa Axios y se crea una instancia con una configuración base que incluye la URL base de la API.

```
// filepath: src/services/api.js
import axios from 'axios';

const api = axios.create({
    baseURL: 'http://localhost:5000/api', // URL base de la API
});

export default api;
```

1.3.2 Servicio `auth.js`

El archivo `auth.js` se encarga de manejar el token de autenticación en el almacenamiento local del navegador. Este archivo se encuentra en la ruta `src/services/auth.js`.

1.3.2.1 Funciones de Autenticación El archivo define dos funciones principales: `setToken` y `clearToken`.

- `setToken(token)`: Almacena el token JWT en el almacenamiento local.
- `clearToken()`: Elimina el token JWT del almacenamiento local.

```
// filepath: src/services/auth.js
export const setToken = (token) => {
    localStorage.setItem('token', token);
};

export const clearToken = () => {
    localStorage.removeItem('token');
};
```

1.3.3 Componente `Login.jsx`

1.3.3.1 Importaciones En el componente `Login.jsx` cargamos estas librerías y módulos:

- `react-bootstrap` para los componentes de formulario y botones.

- `react` para manejar el estado del componente.
- `react-router-dom` para la navegación.
- `api` de `src/services/api.js` para realizar peticiones HTTP con axios.
- `clearToken` y `setToken` de `src/services/auth.js` para manejar el token de autenticación en el almacenamiento local.

1.3.3.2 Estado del Componente

El componente utiliza el hook `useState` para manejar tres estados:

- `username`: Almacena el nombre de usuario ingresado.
- `password`: Almacena la contraseña ingresada.
- `error`: Almacena cualquier mensaje de error que ocurra durante el proceso de inicio de sesión.

1.3.3.3 Navegación

El hook `useNavigate` de `react-router-dom` se utiliza para redirigir al usuario a diferentes rutas después de un intento de inicio de sesión.

1.3.3.4 Función `manejaLogin`

Esta función se ejecuta cuando el usuario envía el formulario de inicio de sesión. Realiza los siguientes pasos:

1. Previene el comportamiento por defecto del formulario.
2. Limpia cualquier mensaje de error previo.
3. Intenta realizar una petición POST a la ruta `/auth/login` con el `username` y `password` ingresados.
4. Si la petición es exitosa, almacena el token JWT recibido en el almacenamiento local y redirige al usuario a la página principal (`/`).
5. Si ocurre un error, limpia el token del almacenamiento local y muestra un mensaje de error.

1.3.3.5 Renderizado del Formulario

El formulario incluye dos campos de entrada para el nombre de usuario y la contraseña, y un botón para enviar el formulario. Si hay un mensaje de error, este se muestra debajo del formulario.

```
1  import { Button, Form } from "react-bootstrap";
2  import { useState } from 'react';
3  import { useNavigate } from 'react-router-dom';
4  import api from "../services/api";
5  import { clearToken, setToken } from "../services/auth";
6
7  const Login = () => {
8      const [username, setUsername] = useState('');
9      const [password, setPassword] = useState('');
```

```
    const [error, setError] = useState('');
    const navigate = useNavigate();

    const manejaLogin = async (event) => {
        event.preventDefault();
        setError('');
        try {
            const response = await api.post('/auth/login', { username,
                password });
            setToken(response.data.jwt);
            navigate('/');
        } catch (err) {
            console.log(err);
            clearToken();
            setError('Credenciales no válidas');
        }
    }

    return(
        <Form>
            <Form.Group className="mb-3">
                <Form.Label>Username:</Form.Label>
                <Form.Control
                    type="text"
                    placeholder="Nombre de Usuario"
                    aria-label="Nombre de Usuario"
                    value={username}
                    onChange={(e) => setUsername(e.target.value)}
                />
            </Form.Group>
            <Form.Group className="mb-3">
                <Form.Label>Password</Form.Label>
                <Form.Control
                    type="password"
                    placeholder="Contraseña"
                    aria-label="Password"
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                />
            </Form.Group>
            <Form.Group className="mb-3">
                <Button onClick={manejaLogin}>
                    ¡Enviar!
                </Button>
            </Form.Group>
            {error && <p style={{ color: 'red' }}>{error}</p>}
        </Form>
    );
};

export default Login;
```

1.3.4 Integración en `LoginPage.jsx`

El componente `Login.jsx` se utiliza en la página `LoginPage.jsx`, la cual se define de la siguiente manera:

```jsx
import { Container } from "react-bootstrap";
import Login from "../components/Login";

const LoginPage = () => {
    return(
        <Container>
            <h3>Inicio de Sesión</h3>
            <Login />
        </Container>
    );
};

export default LoginPage;
```

1.3.5 Ruta en `App.jsx`

La página `LoginPage.jsx` está configurada como una de las rutas en el enrutador principal definido en `App.jsx`:

```jsx
const router = createBrowserRouter([
  {
    path: "/",
    element: <RootLayout />,
    children: [
      {
        index: true,
        element: <HomePage />,
      },
      {
        path: "login",
        element: <LoginPage />,
      },
      // Otras rutas...
    ],
  },
]);

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

export default App;
```

Con esta configuración, cuando el usuario navega a la ruta `/login`, se renderiza la página `LoginPage.jsx`, que a su vez muestra el componente `Login.jsx`.

1.3.6 Integración de `api.jsx` y `auth.jsx` en `Login.jsx`

El componente `Login.jsx` utiliza estos servicios para manejar el proceso de autenticación, puedes verlo en las importaciones:

```
import api from "../services/api";
import { clearToken, setToken } from "../services/auth";
```

1.3.6.1 Uso en la Función `manejaLogin` En la función `manejaLogin`, se realiza una petición POST a la ruta `/auth/login` utilizando la instancia de Axios configurada en `api.js`. Si la petición es exitosa, se almacena el token JWT utilizando la función `setToken` de `auth.js`. Si ocurre un error, se limpia el token utilizando la función `clearToken`.

```
const manejaLogin = async (event) => {
    event.preventDefault();
    setError('');
    try {
        const response = await api.post('/auth/login', { username,
            password });
        setToken(response.data.jwt);
        navigate('/');
    } catch (err) {
        console.log(err);
        clearToken();
        setError('Credenciales no válidas');
    }
};
```

Con esta configuración, el componente `Login.jsx` puede manejar el proceso de autenticación de manera efectiva utilizando los servicios `api.js` y `auth.js`.

1.4 CRUD de Instalaciones

El CRUD de instalaciones se maneja a través de tres páginas principales: `InstalacionesPage`, `InstalacionFormPage` e `InstalacionDeletePage`. Estas páginas se han añadido al enrutador en el archivo `App.js` y reutilizan el componente `InstalacionForm` para añadir, borrar y editar instalaciones.

1.4.1 Páginas del CRUD

1.4.1.1 `InstalacionesPage` Esta página muestra una lista de todas las instalaciones disponibles. Permite a los usuarios ver los detalles de cada instalación y proporciona enlaces para editar o eliminar instalaciones.

1.4.1.2 `InstalacionFormPage` Esta página se utiliza tanto para añadir nuevas instalaciones como para editar instalaciones existentes. Reutiliza el componente `InstalacionForm` y muestra diferentes partes del formulario en función de la ruta y los parámetros proporcionados.

1.4.1.3 `InstalacionDeletePage` Esta página se utiliza para confirmar y realizar la eliminación de una instalación. Muestra un mensaje de confirmación y un botón para proceder con la eliminación.

1.4.2 Configuración del Router en `App.js`

Las rutas para estas páginas se configuran en el archivo `App.js` de la siguiente manera:

```
// filepath: src/App.js
const router = createBrowserRouter([
  {
    path: "/",
    element: <RootLayout />,
    children: [
      {
        index: true,
        element: <HomePage />,
      },
      {
        path: "login",
        element: <LoginPage />,
      },
      {
        path: "instalaciones",
        element: <InstalacionesPage />,
      },
      {
        path: "instalaciones/add",
        element: <InstalacionFormPage />,
      },
      {
        path: "instalaciones/edit/:id",
        element: <InstalacionFormPage />,
      },
```

```
      {
        path: "instalaciones/del/:id",
        element: <InstalacionDeletePage />,
      },
      // Otras rutas...
    ],
  },
]);

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

export default App;
```

1.4.3 Componente `InstalacionForm`

El componente `InstalacionForm` se reutiliza tanto para añadir como para editar instalaciones. Dependiendo de la ruta, se muestran diferentes partes del formulario.

1.4.3.1 Ejemplo de `InstalacionForm`

```
// filepath: src/components/InstalacionForm.jsx
import { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import api from '../services/api';

const InstalacionForm = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const [nombre, setNombre] = useState('');
  const [descripcion, setDescripcion] = useState('');

  useEffect(() => {
    if (id) {
      // Si hay un ID, estamos editando una instalación existente
      api.get(`/instalaciones/${id}`).then(response => {
        setNombre(response.data.nombre);
        setDescripcion(response.data.descripcion);
      });
    }
  }, [id]);

  const handleSubmit = async (event) => {
    event.preventDefault();
    if (id) {
      // Editar instalación existente
      await api.put(`/instalaciones/${id}`, { nombre, descripcion });
    } else {
```

```
      // Añadir nueva instalación
      await api.post('/instalaciones', { nombre, descripcion });
    }
    navigate('/instalaciones');
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>Nombre:</label>
        <input
          type="text"
          value={nombre}
          onChange={(e) => setNombre(e.target.value)}
        />
      </div>
      <div>
        <label>Descripción:</label>
        <textarea
          value={descripcion}
          onChange={(e) => setDescripcion(e.target.value)}
        />
      </div>
      <button type="submit">{id ? 'Editar' : 'Añadir'} Instalación</
        button>
    </form>
  );
};

export default InstalacionForm;
```

1.4.4 Uso de `InstalacionForm` en `InstalacionFormPage`

La página `InstalacionFormPage` reutiliza el componente `InstalacionForm` para manejar tanto la creación como la edición de instalaciones.

```
// filepath: src/pages/InstalacionFormPage.jsx
import { Container } from 'react-bootstrap';
import InstalacionForm from '../components/InstalacionForm';

const InstalacionFormPage = () => {
  return (
    <Container>
      <h3>{useParams().id ? 'Editar Instalación' : 'Nueva Instalación'
        }</h3>
      <InstalacionForm />
    </Container>
  );
};
```

```
13
14 export default InstalacionFormPage;
```

1.4.5 Confirmación de Eliminación en `InstalacionDeletePage`

La página `InstalacionDeletePage` maneja la eliminación de una instalación mostrando un mensaje de confirmación.

```jsx
// filepath: src/pages/InstalacionDeletePage.jsx
import { useParams, useNavigate } from 'react-router-dom';
import api from '../services/api';

const InstalacionDeletePage = () => {
  const { id } = useParams();
  const navigate = useNavigate();

  const handleDelete = async () => {
    await api.delete(`/instalaciones/${id}`);
    navigate('/instalaciones');
  };

  return (
    <div>
      <h3>?`Estás seguro de que deseas eliminar esta instalación?</h3>
      <button onClick={handleDelete}>Eliminar</button>
      <button onClick={() => navigate('/instalaciones')}>Cancelar</
         button>
    </div>
  );
};

export default InstalacionDeletePage;
```