CRUD React para 'articles' - Mejoras implementadas

Implementaciones: 1) react-hook-form (validación) 2) Modal de confirmación 3) Búsqueda y paginación (cliente)

Generado: 2025-10-21 12:39:56

Referencia al fichero de rutas del backend subido por el usuario: ■filecite■turn0file0■

== File: src/utils/api/API.js ==

```
// 1. Obtener URL desde la variable de entorno
const BASE_URL = import.meta.env.VITE_API_BASE_URL;

// 2. Función Genérica de conexión a la API
export const API = async({ endpoint, method = "GET", body, hasBody = false }) => {
    // Para este caso no hacen falta los headers. Únicamente para cuando hay que enviar credenciales
.
    // let headers = {};
    // if(localStorage.getItem("hhToken")) {
    //     headers["Authorization"] = `Bearer ${localStorage.getItem("hhToken")}`;
    // }


    let requestBody = null;

    // Manejor del content-type
    if(hasBody) {
        headers["Content-Type"] = "application/json";
        requestBody = JSON.stringify(body);
    }

    // Construir la URL
    const url = BASE_URL + endpoint;

    try {
        const res = await fetch(url, {
            method,
            // headers,
            body: requestBody
        });
        // Comprobar error
        if(!res.ok) {
            return { error: response };
        }
        // Parsear resultado a json
        const response = await res.json();
        // Devolver resultado
        return { response };
    } catch (error) {
        return { error:{ message: error || "Error de conexión" } };
    }
};
```

== File: src/state/article.actions.js ==

```
// Archivo de acciones para el reducer de artículos
```

```
// Exportamos constantes que representan los tipos de acción
export const ACTIONS = {
    // Inicializa la carga de artículos
    FETCH_INIT: 'FETCH_INIT',
    // Cuando la carga fue exitosa
    FETCH_SUCCESS: 'FETCH_SUCCESS',
    // Cuando ocurre un error en la carga
    FETCH_FAILURE: 'FETCH_FAILURE',
    // Añadir un artículo nuevo
    ADD_ARTICLE: 'ADD_ARTICLE',
    // Actualizar un artículo existente
    UPDATE_ARTICLE: 'UPDATE_ARTICLE',
    // Eliminar un artículo por id
    DELETE_ARTICLE: 'DELETE_ARTICLE',
    // Seleccionar un artículo para edición
    SET_EDITING: 'SET_EDITING',
    // Limpiar el artículo en edición
    CLEAR_EDITING: 'CLEAR_EDITING'
};


== File: src/state/article.reducer.js ==

import { ACTIONS } from './article.actions.js';

// Reducer para manejar el estado de artículos
export const articleReducer = (state, action) => {
    // Switch para cada tipo de acción
    switch(action.type) {
        // Iniciamos la carga: ponemos loading true y limpiamos error
        case ACTIONS.FETCH_INIT:
            return { ...state, loading: true, error: null };
        // Carga exitosa: colocamos los datos en articles y apagamos loading
        case ACTIONS.FETCH_SUCCESS:
            return { ...state, loading: false, articles: action.payload, error: null };
        // Error en carga: almacenamos el error y apagamos loading
        case ACTIONS.FETCH_FAILURE:
            return { ...state, loading: false, error: action.payload };
        // Añadir artículo: lo añadimos al array articles
        case ACTIONS.ADD_ARTICLE:
            return { ...state, articles: [ action.payload, ...state.articles ] };
        // Actualizar artículo: reemplazamos por id
        case ACTIONS.UPDATE_ARTICLE:
            return {
                ...state,
                articles: state.articles.map(a => a._id === action.payload._id ? action.payload : a)
,
                editing: null
            };
        // Eliminar artículo: filtramos por id
        case ACTIONS.DELETE_ARTICLE:
            return { ...state, articles: state.articles.filter(a => a._id !== action.payload) };
        // Seleccionar artículo para edición
        case ACTIONS.SET_EDITING:
            return { ...state, editing: action.payload };
        // Limpiar edición
        case ACTIONS.CLEAR_EDITING:
            return { ...state, editing: null };
        // Default: devolvemos el estado actual
        default:
```

```
            return state;
    }
};


== File: src/services/articleService.js ==

// Servicio que usa la función API proporcionada para interactuar con el backend
import { API } from '../utils/api/API.js';

// Obtener todos los artículos
export const fetchArticles = async () => {
    // Llamada a API genérica usando async/await
    const { response, error } = await API({ endpoint: '/articles', method: 'GET' });
    // Devolvemos los datos o lanzamos error para manejarlo en el componente
    if(error) throw error;
    return response;
};

// Obtener artículo por id
export const fetchArticleById = async (id) => {
    const { response, error } = await API({ endpoint: `/articles/${id}`, method: 'GET' });
    if(error) throw error;
    return response;
};

// Crear nuevo artículo (hasBody true)
export const createArticle = async (body) => {
    const { response, error } = await API({ endpoint: '/articles', method: 'POST', body, hasBody: tr
ue });
    if(error) throw error;
    return response;
};

// Actualizar artículo por id
export const updateArticleById = async (id, body) => {
    const { response, error } = await API({ endpoint: `/articles/${id}`, method: 'PUT', body, hasBod
y: true });
    if(error) throw error;
    return response;
};

// Eliminar artículo por id
export const deleteArticleById = async (id) => {
    const { response, error } = await API({ endpoint: `/articles/${id}`, method: 'DELETE' });
    if(error) throw error;
    return response;
};


== File: src/components/Modal/Modal.jsx ==

import React from 'react'; // Import React para JSX
import './Modal.css'; // Estilos del modal

// Componente modal genérico: title, message, onConfirm, onCancel
const Modal = ({ title, message, onConfirm, onCancel }) => {
    // Render del modal con overlay y cuadro central
    return (
        <div className='modal-overlay'>
```

```
            <div className='modal'>
                <h3>{title}</h3>
                <p>{message}</p>
                <div className='modal-actions'>
                    <button onClick={onCancel}>Cancelar</button>
                    <button onClick={onConfirm}>Confirmar</button>
                </div>
            </div>
        </div>
    );
};

export default Modal;


== File: src/components/Modal/Modal.css ==

.modal-overlay {
    position: fixed;
    top: 0;
    left: 0;
    right:0;
    bottom:0;
    display:flex;
    align-items:center;
    justify-content:center;
    background: rgba(0,0,0,0.4);
    z-index: 1000;
}
.modal {
    background: #fff;
    padding: 1rem;
    border-radius:8px;
    width: 320px;
    box-shadow: 0 8px 20px rgba(0,0,0,0.15);
}
.modal-actions {
    display:flex;
    justify-content:flex-end;
    gap: 0.5rem;
    margin-top:1rem;
}


== File: src/components/ArticleList/ArticleList.jsx ==

import React, { useEffect, useState, useMemo } from 'react'; // Import React y hooks necesarios
import './ArticleList.css'; // Importar estilos del componente
import Modal from '../Modal/Modal.jsx'; // Modal para confirmación

// Componente funcional para listar artículos con búsqueda y paginación cliente
const ArticleList = ({ state, dispatch, actions, services }) => {
    // Estado local para búsqueda, paginación y control del modal
    const [query, setQuery] = useState(''); // Texto de búsqueda
    const [currentPage, setCurrentPage] = useState(1); // Página actual
    const [pageSize] = useState(5); // Tamaño de página (constante por ahora)
    const [showModal, setShowModal] = useState(false); // Mostrar modal de confirmación
    const [toDelete, setToDelete] = useState(null); // Artículo pendiente de eliminar

    // useEffect para cargar los artículos al montar el componente
```

```
useEffect(() => {
    // Definimos una función async dentro del efecto
    const load = async () => {
        // Disparamos la acción de inicio de fetch
        dispatch({ type: actions.FETCH_INIT });
        try {
            // Llamada al servicio para obtener artículos (async/await)
            const data = await services.fetchArticles();
            // Dispatch success con los datos
            dispatch({ type: actions.FETCH_SUCCESS, payload: data });
        } catch (err) {
            // Dispatch failure con el error
            dispatch({ type: actions.FETCH_FAILURE, payload: err });
        }
    };
    // Ejecutar carga
    load();
}, [dispatch, actions, services]); // Dependencias del useEffect

// useMemo para calcular artículos filtrados según la query (optimización)
const filtered = useMemo(() => {
    // Si no hay query devolvemos todos
    if(!query) return state.articles;
    // Buscar en título y contenido (case-insensitive)
    const q = query.toLowerCase();
    return state.articles.filter(a => {
        const title = (a.title || '').toLowerCase();
        const content = (a.content || '').toLowerCase();
        return title.includes(q) || content.includes(q);
    });
}, [state.articles, query]);

// Calcular total de páginas
const totalPages = Math.max(1, Math.ceil(filtered.length / pageSize));

// Asegurar que currentPage está en rango cuando cambian filtered/totalPages
useEffect(() => {
    if(currentPage > totalPages) setCurrentPage(totalPages);
    if(currentPage < 1) setCurrentPage(1);
}, [currentPage, totalPages]);

// Artículos a mostrar en la página actual
const paginated = useMemo(() => {
    const start = (currentPage - 1) * pageSize;
    return filtered.slice(start, start + pageSize);
}, [filtered, currentPage, pageSize]);

// Handler para solicitar eliminación: abre modal
const requestDelete = (article) => {
    setToDelete(article);
    setShowModal(true);
};

// Confirmación de eliminación: llama al servicio y actualiza estado
const confirmDelete = async () => {
    if(!toDelete) return;
    try {
        await services.deleteArticleById(toDelete._id);
        dispatch({ type: actions.DELETE_ARTICLE, payload: toDelete._id });
        // Ajuste: si la página quedó sin elementos, retroceder una página
```

```
            if(paginated.length === 1 && currentPage > 1) {
                setCurrentPage(currentPage - 1);
            }
        } catch(e) {
            alert('Error eliminando: ' + (e.message || JSON.stringify(e)));
        } finally {
            setShowModal(false);
            setToDelete(null);
        }
    };

    // Cancelar eliminación
    const cancelDelete = () => {
        setShowModal(false);
        setToDelete(null);
    };

    // Renderizado del componente
    return (
        <div className='article-list'>
            {/* Controles: búsqueda y paginación */}
            <div className='controls'>
                <input
                    placeholder='Buscar por título o contenido...'
                    value={query}
                    onChange={(e) => { setQuery(e.target.value); setCurrentPage(1); }}
                />
                <div className='pagination-info'>
                    <span>Página {currentPage} de {totalPages}</span>
                </div>
            </div>

            {/* Si loading true mostramos mensaje */}
            {state.loading && <p>Cargando artículos...</p>}
            {/* Si hay error mostramos mensaje */}
            {state.error && <p className='error'>Error: {state.error.message || JSON.stringify(state
.error)}</p>}

            {/* Listado de artículos paginados */}
            <ul>
                {paginated.map(article => (
                    <li key={article._id}>
                        <div className='article-item'>
                            <h3>{article.title}</h3>
                            <p>{article.content}</p>
                            <div className='actions'>
                                <button onClick={() => dispatch({ type: actions.SET_EDITING, payload
: article })}>Editar</button>
                                <button onClick={() => requestDelete(article)}>Eliminar</button>
                            </div>
                        </div>
                    </li>
                ))}
            </ul>

            {/* Paginación: controles prev/next y números */}
            <div className='pagination'>
                <button onClick={() => setCurrentPage(p => Math.max(1, p - 1))} disabled={currentPag
e === 1}>Anterior</button>
                {/* Mostrar algunos botones de página (simple) */}
```

```
                {Array.from({ length: totalPages }).map((_, idx) => {
                    const page = idx + 1;
                    return (
                        <button
                            key={page}
                            className={page === currentPage ? 'active' : ''}
                            onClick={() => setCurrentPage(page)}
                        >
                            {page}
                        </button>
                    );
                })}
                <button onClick={() => setCurrentPage(p => Math.min(totalPages, p + 1))} disabled={c
urrentPage === totalPages}>Siguiente</button>
            </div>

            {/* Modal de confirmación */}
            {showModal && (
                <Modal
                    title='Confirmar eliminación'
                    message={`¿Deseas eliminar el artículo "${toDelete?.title}"? Esta acción no se p
uede deshacer.`}
                    onConfirm={confirmDelete}
                    onCancel={cancelDelete}
                />
            )}
        </div>
    );
};

export default ArticleList;


== File: src/components/ArticleList/ArticleList.css ==

.article-list {
    padding: 1rem;
}
.controls {
    display:flex;
    justify-content:space-between;
    margin-bottom: 0.5rem;
    gap: 0.5rem;
}
.controls input {
    flex:1;
    padding:0.5rem;
}
.article-item {
    border: 1px solid #ddd;
    padding: 0.75rem;
    margin-bottom: 0.5rem;
    border-radius: 6px;
}
.article-item h3 {
    margin: 0 0 0.25rem 0;
}
.article-item .actions {
    margin-top: 0.5rem;
}
```

```
.article-item .actions button {
    margin-right: 0.5rem;
}
.pagination {
    display:flex;
    gap:0.25rem;
    margin-top:0.5rem;
    flex-wrap:wrap;
}
.pagination button.active {
    font-weight:bold;
    text-decoration: underline;
}


== File: src/components/ArticleForm/ArticleForm.jsx ==

import React, { useEffect } from 'react'; // Import React y hooks necesarios
import './ArticleForm.css'; // Importar estilos
import { useForm } from 'react-hook-form'; // react-hook-form para validación y manejo de formulario
s

// Formulario para crear/editar artículos usando react-hook-form.
// Recibe dispatch, estado y acciones por props.
const ArticleForm = ({ state, dispatch, actions, services }) => {
    // Inicializamos useForm con valores por defecto vacíos
    const { register, handleSubmit, reset, formState: { errors, isSubmitting } } = useForm({
        defaultValues: { title: '', content: '' }
    });

    // useEffect para rellenar el formulario cuando editing cambia
    useEffect(() => {
        // Si hay un artículo en edición, reseteamos el formulario con sus valores
        if(state.editing) {
            reset({
                title: state.editing.title || '',
                content: state.editing.content || ''
            });
        } else {
            // Si no, limpiamos el formulario
            reset({ title: '', content: '' });
        }
    }, [state.editing, reset]);

    // Función para guardar el formulario (async/await)
    const onSubmit = async (data) => {
        // isSubmitting lo maneja react-hook-form para evitar dobles envíos
        try {
            // Si hay editing llamamos a update, si no a create
            if(state.editing) {
                const updated = await services.updateArticleById(state.editing._id, data);
                // Dispatch acción de actualización con el artículo devuelto
                dispatch({ type: actions.UPDATE_ARTICLE, payload: updated });
            } else {
                const created = await services.createArticle(data);
                // Dispatch para añadir el nuevo artículo
                dispatch({ type: actions.ADD_ARTICLE, payload: created });
            }
            // Limpiar formulario y edición
            dispatch({ type: actions.CLEAR_EDITING });
```

```
            reset({ title: '', content: '' });
        } catch (err) {
            // Mostrar error al usuario
            alert('Error guardando artículo: ' + (err.message || JSON.stringify(err)));
        }
    };

    // Render del formulario con validación y mensajes de error
    return (
        <form className='article-form' onSubmit={handleSubmit(onSubmit)}>
            <h2>{state.editing ? 'Editar artículo' : 'Crear artículo'}</h2>

            <label>
                Título
                {/* register() conecta el input con react-hook-form y añade validaciones */}
                <input {...register('title', { required: 'El título es obligatorio', maxLength: { va
lue: 120, message: 'Máx 120 caracteres' } })} />
                {/* Mostrar error si existe */}
                {errors.title && <span className='error'>{errors.title.message}</span>}
            </label>

            <label>
                Contenido
                <textarea {...register('content', { required: 'El contenido es obligatorio', minLeng
th: { value: 10, message: 'Mínimo 10 caracteres' } })} />
                {errors.content && <span className='error'>{errors.content.message}</span>}
            </label>

            <div className='buttons'>
                <button type='submit' disabled={isSubmitting}>{isSubmitting ? 'Guardando...' : 'Guar
dar'}</button>
                {state.editing && (
                    <button type='button' onClick={() => { dispatch({ type: actions.CLEAR_EDITING })
; reset({ title: '', content: '' }); }}>Cancelar</button>
                )}
            </div>
        </form>
    );
};

export default ArticleForm;


== File: src/components/ArticleForm/ArticleForm.css ==

.article-form {
    padding: 1rem;
    border: 1px solid #eee;
    border-radius: 6px;
}
.article-form label {
    display: block;
    margin-bottom: 0.5rem;
}
.article-form input, .article-form textarea {
    width: 100%;
    padding: 0.5rem;
    margin-top: 0.25rem;
    box-sizing: border-box;
}
```

```
.article-form .buttons {
    margin-top: 0.5rem;
}
.article-form .buttons button {
    margin-right: 0.5rem;
}
.error {
    color: #b00020;
    font-size: 0.9rem;
    display:block;
    margin-top:0.25rem;
}


== File: src/App.jsx ==

import React, { useReducer } from 'react'; // Import React y useReducer
import './App.css'; // Import styles globales
import ArticleList from './components/ArticleList/ArticleList.jsx'; // Componente lista
import ArticleForm from './components/ArticleForm/ArticleForm.jsx'; // Componente formulario
import { articleReducer } from './state/article.reducer.js'; // Reducer
import { ACTIONS } from './state/article.actions.js'; // Acciones
import * as services from './services/articleService.js'; // Servicios de API

// Estado inicial para el reducer
const initialState = {
    articles: [], // Array de artículos
    loading: false, // Indicador de carga
    error: null, // Error en carga
    editing: null // Artículo en edición
};

// Componente principal de la aplicación
const App = () => {
    // Uso avanzado del hook useReducer para manejar el estado complejo
    const [state, dispatch] = useReducer(articleReducer, initialState);

    // Render principal: pasamos state, dispatch, acciones y servicios a los componentes hijos
    return (
        <div className='app'>
            <header className='app-header'>
                <h1>Gestión de Artículos</h1>
            </header>
            <main className='app-main'>
                {/* Columna de formulario */}
                <section className='left'>
                    <ArticleForm state={state} dispatch={dispatch} actions={ACTIONS} services={servi
ces} />
                </section>
                {/* Columna de lista */}
                <section className='right'>
                    <ArticleList state={state} dispatch={dispatch} actions={ACTIONS} services={servi
ces} />
                </section>
            </main>
        </div>
    );
};

export default App;
```

== File: src/App.css ==

```
.app {
    font-family: Arial, sans-serif;
    max-width: 1000px;
    margin: 0 auto;
    padding: 1rem;
}
.app-header {
    text-align: center;
    margin-bottom: 1rem;
}
.app-main {
    display: grid;
    grid-template-columns: 1fr 2fr;
    gap: 1rem;
}
```

== File: src/main.jsx ==

```
import React from 'react'; // Import React
import { createRoot } from 'react-dom/client'; // Import función para montar la app
import App from './App.jsx'; // Importar componente App

// Montaje de la aplicación en el elemento con id 'root'
const container = document.getElementById('root');
const root = createRoot(container);
root.render(<App />);
```

== Explicación de las mejoras implementadas ==

1) Validación con react-hook-form:
- Se ha reemplazado el control manual de inputs por 'react-hook-form' en ArticleForm.
- Ventajas: menor re-render, manejo de isSubmitting, validación declarativa y mensajes de error.
- Campos validados: title (required, maxLength 120), content (required, minLength 10).
- Uso: register conecta inputs al formulario; handleSubmit envuelve la función onSubmit; reset reset ea valores.

2) Modal de confirmación para eliminar:
- Componente Modal genérico en src/components/Modal con overlay y acciones Confirmar/Cancelar.
- ArticleList usa estado local (showModal y toDelete) para gestionar la lógica de confirmación.
- confirmDelete llama al servicio deleteArticleById y dispatch({ DELETE_ARTICLE }) al completar.

3) Búsqueda y paginación cliente:
- Búsqueda: input que actualiza 'query' y filtra artículos por título o contenido (case-insensitive) .
- useMemo se usa para memorizar el array filtrado y la porción paginada (optimización).
- Paginación: cliente, tamaño de página 5, controles Prev/Next y botones por página.
- Ajuste: si al eliminar un artículo la página queda vacía, retrocede de página automáticamente.

Notas adicionales:
- Todas las llamadas a la API siguen usando async/await y los servicios en services/articleService.j s.
- No asumimos paginación en el backend; por tanto la paginación es cliente. Si el backend soporta pa ginación, el servicio puede adaptarse para enviar parámetros query (page, limit, q).