Implémentation d'une barre de recherche avec Laravel et React

Voici un guide étape par étape pour créer une barre de recherche qui permet de filtrer des produits par nom et par catégorie.

Backend (Laravel)

1. Créer les modèles nécessaires

Assurez-vous d'avoir les modèles `Product` et `Category` :

php

```php
// app/Models/Product.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    protected $fillable = ['name', 'description', 'price', 'category_id'];

    public function category()
    {
        return $this->belongsTo(Category::class);
    }
}

// app/Models/Category.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    protected $fillable = ['name'];

    public function products()
    {
        return $this->hasMany(Product::class);
    }
}
```

2. Créer une API pour la recherche

php

```php
// routes/api.php
Route::get('/products/search', [ProductController::class, 'search']);
```

3. Implémenter le contrôleur

php

```php
// app/Http/Controllers/ProductController.php
namespace App\Http\Controllers;

use App\Models\Product;
use Illuminate\Http\Request;

class ProductController extends Controller
{
    public function search(Request $request)
    {
        $query = $request->input('query');

        // Recherche par nom de produit et catégorie
        $products = Product::with('category')
            ->where('name', 'LIKE', "%{$query}%")
            ->orWhereHas('category', function($q) use ($query) {
                $q->where('name', 'LIKE', "%{$query}%");
            })
            ->get();

        return response()->json($products);
    }
}
```

Frontend (React)

1. Créer un composant de barre de recherche

jsx

```jsx
// SearchBar.jsx
import { useState, useEffect } from 'react';
import axios from 'axios';

const SearchBar = () => {
    const [query, setQuery] = useState('');
    const [products, setProducts] = useState([]);
    const [loading, setLoading] = useState(false);

    useEffect(() => {
        // Debounce pour éviter trop de requêtes
        const timeoutId = setTimeout(() => {
            if (query.trim() !== '') {
                searchProducts();
            } else {
                setProducts([]);
            }
        }, 500);

        return () => clearTimeout(timeoutId);
    }, [query]);

    const searchProducts = async () => {
        setLoading(true);
        try {
            const response = await axios.get(`/api/products/search?query=${query}`);
            setProducts(response.data);
        } catch (error) {
            console.error('Erreur lors de la recherche:', error);
        } finally {
            setLoading(false);
        }
    };

    return (
        <div className="search-container">
            <div className="search-input">
                <input
                    type="text"
                    placeholder="Rechercher par nom de produit ou catégorie..."
                    value={query}
                    onChange={(e) => setQuery(e.target.value)}
                />
                {loading && <span className="loader">Chargement...</span>}
            </div>
```

```
            {products.length > 0 && (
                <div className="search-results">
                    <h3>Résultats ({products.length})</h3>
                    <ul>
                        {products.map(product => (
                            <li key={product.id} className="product-item">
                                <h4>{product.name}</h4>
                                <span className="category-badge">
                                    Catégorie: {product.category.name}
                                </span>
                                <p>{product.description}</p>
                                <p className="price">{product.price} €</p>
                            </li>
                        ))}
                    </ul>
                </div>
            )}
        </div>
    );
};

export default SearchBar;
```

2. Ajouter des styles CSS

css

```css
/* styles.css */
.search-container {
    width: 100%;
    max-width: 800px;
    margin: 0 auto;
}

.search-input {
    position: relative;
    margin-bottom: 20px;
}

.search-input input {
    width: 100%;
    padding: 12px 15px;
    border: 1px solid #ddd;
    border-radius: 4px;
    font-size: 16px;
}

.loader {
    position: absolute;
    right: 15px;
    top: 50%;
    transform: translateY(-50%);
    color: #888;
}

.search-results {
    border: 1px solid #eee;
    border-radius: 4px;
    padding: 15px;
}

.product-item {
    border-bottom: 1px solid #eee;
    padding: 15px 0;
}

.product-item:last-child {
    border-bottom: none;
}

.category-badge {
    background-color: #e2f0ff;
    color: #0066cc;
```

```
    padding: 3px 8px;
    border-radius: 12px;
    font-size: 12px;
    margin-left: 10px;
}

.price {
    font-weight: bold;
    color: #e83e8c;
}
```

3. Intégrer le composant dans votre application React

jsx

```jsx
// App.jsx
import SearchBar from './components/SearchBar';

function App() {
    return (
        <div className="App">
            <header>
                <h1>Notre catalogue de produits</h1>
            </header>
            <main>
                <SearchBar />
                {/* Autres composants de votre application */}
            </main>
        </div>
    );
}

export default App;
```

Configuration CSRF pour Laravel et React

Pour que les requêtes API fonctionnent correctement, configurez axios :

jsx

```jsx
// Dans votre fichier principal (index.jsx ou App.jsx)
import axios from 'axios';

// Configuration d'Axios pour Laravel
axios.defaults.baseURL = 'http://votre-domaine.com'; // Remplacez par votre URL
axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';
axios.defaults.withCredentials = true;

// Si vous utilisez Laravel Sanctum
axios.get('/sanctum/csrf-cookie').then(() => {
    // Les requêtes pourront ensuite être effectuées avec le token CSRF
});
```

Tests

Pour tester votre implémentation :

1. Assurez-vous que votre base de données contient des produits et des catégories
2. Lancez votre serveur Laravel : `php artisan serve`
3. Démarrez votre application React : `npm run dev` ou `npm start` (selon votre configuration)
4. Testez la barre de recherche avec différents termes

Optimisations possibles

- Ajouter une pagination pour gérer un grand nombre de résultats
- Implémenter des filtres supplémentaires (prix, date, etc.)
- Ajouter un système de tri des résultats
- Mettre en surbrillance les termes recherchés dans les résultats