Backend (Java Example using Spring/Hibernate)

// --- Product Entity (Model) ---

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String category;

private double price;

// ... other fields ...

// Getters and Setters

}

// --- Product Repository (Data Access) ---

public interface ProductRepository extends JpaRepository<Product, Long> {

}

// --- Product Service (Business Logic) ---

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

public Product updateProduct(Long id, Product productDetails) {

Product product = productRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundException("Product not found with id: " + id)); // Handle not found

// Update only allowed fields

product.setName(productDetails.getName());

product.setCategory(productDetails.getCategory());

product.setPrice(productDetails.getPrice());

// ... update other fields ...

return productRepository.save(product);

}

public Product getProductById(Long id) {

return productRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundException("Product not found with id: " + id));

}

}

// --- Product Controller (API Endpoint) ---

@RestController

@RequestMapping("/api/products")

public class ProductController {

@Autowired

private ProductService productService;

@GetMapping("/{id}")

public Product getProductById(@PathVariable Long id) {

return productService.getProductById(id);

}

@PutMapping("/{id}")

public ResponseEntity<Product> updateProduct(@PathVariable Long id, @RequestBody Product productDetails) {

Product updatedProduct = productService.updateProduct(id, productDetails);

return ResponseEntity.ok(updatedProduct);

}

}

// --- Exception Handling ---

@ResponseStatus(HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

Frontend (React Example)

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function EditProduct() {

const [product, setProduct] = useState({ name: '', category: '', price: 0 });

const [productId, setProductId] = useState(''); // For searching/selecting

const [message, setMessage] = useState('');

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

// --- Fetch Product Details ---

useEffect(() => {

if (productId) {

axios.get(`/api/products/${productId}`)

.then(response => setProduct(response.data))

.catch(err => setError("Error fetching product: " + err.message));

}

}, [productId]);

// --- Handle Form Input Changes ---

const handleInputChange = (e) => {

const { name, value } = e.target;

setProduct({ ...product, [name]: value });

};

// --- Handle Update Submission ---

const handleSubmit = async (e) => {

e.preventDefault();

try {

await axios.put(`/api/products/${productId}`, product);

setMessage('Product updated successfully!');

setError('');

} catch (err) {

setError('Error updating product: ' + err.response.data.message);

setMessage('');

}

};

return (

<div>

<h2>Edit Product</h2>

{/\* --- Search for Product --- \*/}

<div>

<label>Enter Product ID to Edit: </label>

<input type="text" value={productId} onChange={(e) => setProductId(e.target.value)} />

</div>

{productId && error && <p style={{ color: 'red' }}>{error}</p>}

{productId && !error && (

<form onSubmit={handleSubmit}>

<div>

<label>Name: </label>

<input type="text" name="name" value={product.name} onChange={handleInputChange} />

</div>

<div>

<label>Category: </label>

<input type="text" name="category" value={product.category} onChange={handleInputChange} />

</div>

<div>

<label>Price: </label>

<input type="number" name="price" value={product.price} onChange={handleInputChange} />

</div>

<button type="submit">Update Product</button>

</form>

)}

{message && <p style={{ color: 'green' }}>{message}</p>}

</div>

);

}

export default EditProduct;

**Searching Products/Buyers**

**A. Backend (Java/Spring/Hibernate with Search)**

// ... (Product Entity and Repository as above)

// --- Product Service (with Search) ---

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

public List<Product> searchProducts(String query) {

// Simple search (can be enhanced with more advanced techniques)

return productRepository.findByNameContainingIgnoreCaseOrCategoryContainingIgnoreCase(query, query);

}

public List<Product> getAllProducts() {

return productRepository.findAll();

}

// ... (updateProduct, getProductById as above)

}

public interface ProductRepository extends JpaRepository<Product, Long> {

List<Product> findByNameContainingIgnoreCaseOrCategoryContainingIgnoreCase(String name, String category);

}

// --- Product Controller (API Endpoint) ---

@RestController

@RequestMapping("/api/products")

public class ProductController {

@Autowired

private ProductService productService;

@GetMapping("/search")

public List<Product> searchProducts(@RequestParam String query) {

return productService.searchProducts(query);

}

@GetMapping

public List<Product> getAllProducts() {

return productService.getAllProducts();

}

// ... (Other endpoints)

}

**Frontend (React Example)**

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function SearchProducts() {

const [products, setProducts] = useState([]);

const [searchTerm, setSearchTerm] = useState('');

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

// --- Fetch Products on Initial Load and Search ---

useEffect(() => {

const fetchProducts = async () => {

try {

const result = searchTerm

? await axios.get(`/api/products/search?query=${searchTerm}`)

: await axios.get('/api/products');

setProducts(result.data);

setError('');

} catch (err) {

setError('Error fetching products: ' + err.message);

setProducts([]);

}

};

fetchProducts();

}, [searchTerm]);

const handleSearchChange = (e) => {

setSearchTerm(e.target.value);

};

const handleClearSearch = () => {

setSearchTerm('');

};

return (

<div>

<h2>Search Products</h2>

<div>

<input

type="text"

placeholder="Search by name or category"

value={searchTerm}

onChange={handleSearchChange}

/>

<button onClick={handleClearSearch}>Clear</button>

</div>

{error && <p style={{ color: 'red' }}>{error}</p>}

{products.length > 0 && (

<table>

<thead>

<tr>

<th>Name</th>

<th>Category</th>

<th>Price</th>

{/\* ... other headers ... \*/}

<th>Actions</th>

</tr>

</thead>

<tbody>

{products.map(product => (

<tr key={product.id}>

<td>{product.name}</td>

<td>{product.category}</td>

<td>{product.price}</td>

{/\* ... other data ... \*/}

<td>

<button onClick={() => {/\* Implement edit \*/}}>Edit</button>

<button onClick={() => {/\* Implement view \*/}}>View</button>

<button onClick={() => {/\* Implement delete \*/}}>Delete</button>

</td>

</tr>

))}

</tbody>

</table>

)}

{products.length === 0 && !error && <p>No products found.</p>}

</div>

);

}

export default SearchProducts;