Complete React Hooks Guide with E-commerce Application

Table of Contents

Introduction to React Hooks {#introduction}

React Hooks are functions that let you use state and other React features in functional components. They were introduced in React 16.8 and have revolutionized how we write React applications.

Why Hooks?

- **Simpler Code**: No need for class components for state management
- **Reusable Logic**: Custom hooks allow sharing stateful logic between components
- **Better Testing**: Easier to test individual pieces of functionality
- **Performance**: Better optimization opportunities

Rules of Hooks

1. Only call hooks at the top level of React functions
2. Only call hooks from React function components or custom hooks
3. Hook calls must be in the same order every time the component renders

useState Hook {#usestate}

The `useState` hook allows you to add state to functional components.

Syntax

javascript

```
const [state, setState] = useState(initialValue);
```

E-commerce Example: Shopping Cart Counter

```javascript
import React, { useState } from 'react';

function ProductCard({ product }) {
  const [quantity, setQuantity] = useState(1);
  const [isInCart, setIsInCart] = useState(false);

  const handleAddToCart = () => {
    setIsInCart(true);
    // Add to cart logic here
  };

  const incrementQuantity = () => {
    setQuantity(prev => prev + 1);
  };

  const decrementQuantity = () => {
    setQuantity(prev => prev > 1 ? prev - 1 : 1);
  };

  return (
    <div className="product-card">
      <h3>{product.name}</h3>
      <p>${product.price}</p>

      <div className="quantity-selector">
        <button onClick={decrementQuantity}>-</button>
        <span>{quantity}</span>
        <button onClick={incrementQuantity}>+</button>
      </div>

      <button
        onClick={handleAddToCart}
        disabled={isInCart}
      >
        {isInCart ? 'Added to Cart' : 'Add to Cart'}
      </button>
    </div>
  );
}
```

Key Points:

- State updates are asynchronous
- Use functional updates when new state depends on previous state
- useState can hold any type of value (primitive, object, array)

useEffect Hook {#useeffect}

The `useEffect` hook lets you perform side effects in functional components. It serves the same purpose as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` combined.

Syntax

javascript

```javascript
useEffect(() => {
  // Side effect code
  return () => {
    // Cleanup code (optional)
  };
}, [dependencies]); // Dependency array (optional)
```

E-commerce Example: Product Data Fetching

javascript

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

function ProductList() {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [searchTerm, setSearchTerm] = useState('');

  // Effect for fetching products (runs once on mount)
  useEffect(() => {
    const fetchProducts = async () => {
      try {
        setLoading(true);
        const response = await fetch('/api/products');
        const data = await response.json();
        setProducts(data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchProducts();
  }, []); // Empty dependency array = run once on mount

  // Effect for search (runs when searchTerm changes)
  useEffect(() => {
    if (searchTerm) {
      const timeoutId = setTimeout(() => {
        const filtered = products.filter(product =>
          product.name.toLowerCase().includes(searchTerm.toLowerCase())
        );
        setProducts(filtered);
      }, 300); // Debounce search

      return () => clearTimeout(timeoutId); // Cleanup
    }
  }, [searchTerm, products]); // Runs when searchTerm or products change

  // Effect for cleanup (component unmount)
  useEffect(() => {
    return () => {
      // Cleanup subscriptions, cancel requests, etc.
      console.log('Component unmounting');
    };
  }, []);

  if (loading) return <div>Loading products...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <div>
      <input
        type="text"
        placeholder="Search products..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
      />
      <div className="products-grid">
        {products.map(product => (
          <ProductCard key={product.id} product={product} />
        ))}
      </div>
    </div>
  );
}
```

useEffect Patterns:

1. **No dependency array**: Runs after every render
2. **Empty dependency array []**: Runs once on mount
3. **With dependencies [dep1, dep2]**: Runs when dependencies change
4. **Return cleanup function**: Runs on unmount or before next effect

useContext Hook {#usecontext}

The `useContext` hook provides a way to pass data through the component tree without having to pass props down manually at every level.

E-commerce Example: Shopping Cart Context

```javascript
```

```
import React, { createContext, useContext, useReducer } from 'react';

// Create Cart Context
const CartContext = createContext();

// Cart actions
const cartActions = {
  ADD_ITEM: 'ADD_ITEM',
  REMOVE_ITEM: 'REMOVE_ITEM',
  UPDATE_QUANTITY: 'UPDATE_QUANTITY',
  CLEAR_CART: 'CLEAR_CART'
};

// Cart reducer
function cartReducer(state, action) {
  switch (action.type) {
    case cartActions.ADD_ITEM:
      const existingItem = state.items.find(item => item.id === action.payload.id);
      if (existingItem) {
        return {
          ...state,
          items: state.items.map(item =>
            item.id === action.payload.id
              ? { ...item, quantity: item.quantity + 1 }
              : item
          )
        };
      }
      return {
        ...state,
        items: [...state.items, { ...action.payload, quantity: 1 }]
      };

    case cartActions.REMOVE_ITEM:
      return {
        ...state,
        items: state.items.filter(item => item.id !== action.payload)
      };

    case cartActions.UPDATE_QUANTITY:
      return {
        ...state,
        items: state.items.map(item =>
          item.id === action.payload.id
            ? { ...item, quantity: action.payload.quantity }
            : item
      )
    };

  case cartActions.CLEAR_CART:
    return { ...state, items: [] };

  default:
    return state;
  }
}

// Cart Provider Component
export function CartProvider({ children }) {
  const [cart, dispatch] = useReducer(cartReducer, { items: [] });

  const addItem = (product) => {
    dispatch({ type: cartActions.ADD_ITEM, payload: product });
  };

  const removeItem = (productId) => {
    dispatch({ type: cartActions.REMOVE_ITEM, payload: productId });
  };

  const updateQuantity = (productId, quantity) => {
    dispatch({
      type: cartActions.UPDATE_QUANTITY,
      payload: { id: productId, quantity }
    });
  };

  const clearCart = () => {
    dispatch({ type: cartActions.CLEAR_CART });
  };

  const getCartTotal = () => {
    return cart.items.reduce((total, item) =>
      total + (item.price * item.quantity), 0
    );
  };

  const getItemCount = () => {
    return cart.items.reduce((count, item) => count + item.quantity, 0);
  };

  const value = {
    cart,
    addItem,
    removeItem,
    updateQuantity,
    clearCart,
    getCartTotal,
    getItemCount
  };

  return (
    <CartContext.Provider value={value}>
      {children}
    </CartContext.Provider>
  );
}

// Custom hook to use cart context
export function useCart() {
  const context = useContext(CartContext);
  if (!context) {
    throw new Error('useCart must be used within a CartProvider');
  }
  return context;
}

// Usage in components
function CartIcon() {
  const { getItemCount } = useCart();

  return (
    <div className="cart-icon">
      🛒 ({getItemCount()})
    </div>
  );
}

function ProductCard({ product }) {
  const { addItem } = useCart();

  return (
    <div className="product-card">
      <h3>{product.name}</h3>
      <p>${product.price}</p>
      <button onClick={() => addItem(product)}>
        Add to Cart
      </button>
    </div>
  );
}
```

useReducer Hook {#usereducer}

The `useReducer` hook is an alternative to `useState` for managing complex state logic. It's especially useful when you have complex state transitions.

When to use useReducer vs useState:

- **useState**: Simple state updates
- **useReducer**: Complex state logic, multiple sub-values, state transitions depend on previous state

E-commerce Example: Order Management

```
javascript
```

```
import React, { useReducer } from 'react';

const orderActions = {
  SET_LOADING: 'SET_LOADING',
  SET_ERROR: 'SET_ERROR',
  SET_SUCCESS: 'SET_SUCCESS',
  RESET: 'RESET',
  UPDATE_SHIPPING: 'UPDATE_SHIPPING',
  UPDATE_PAYMENT: 'UPDATE_PAYMENT'
};

function orderReducer(state, action) {
  switch (action.type) {
    case orderActions.SET_LOADING:
      return {
        ...state,
        loading: true,
        error: null
      };

    case orderActions.SET_ERROR:
      return {
        ...state,
        loading: false,
        error: action.payload
      };

    case orderActions.SET_SUCCESS:
      return {
        ...state,
        loading: false,
        error: null,
        orderComplete: true,
        orderId: action.payload
      };

    case orderActions.UPDATE_SHIPPING:
      return {
        ...state,
        shippingInfo: { ...state.shippingInfo, ...action.payload }
      };

    case orderActions.UPDATE_PAYMENT:
      return {
        ...state,
        paymentInfo: { ...state.paymentInfo, ...action.payload }
      };

    case orderActions.RESET:
      return initialState;

    default:
      return state;
  }
}

const initialState = {
  loading: false,
  error: null,
  orderComplete: false,
  orderId: null,
  shippingInfo: {
    address: '',
    city: '',
    zipCode: '',
    country: ''
  },
  paymentInfo: {
    cardNumber: '',
    expiryDate: '',
    cvv: ''
  }
};

function CheckoutForm() {
  const [orderState, dispatch] = useReducer(orderReducer, initialState);

  const handleSubmitOrder = async () => {
    dispatch({ type: orderActions.SET_LOADING });

    try {
      const response = await fetch('/api/orders', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          shipping: orderState.shippingInfo,
          payment: orderState.paymentInfo
        })
      });

      const data = await response.json();
      dispatch({ type: orderActions.SET_SUCCESS, payload: data.orderId });
    } catch (error) {
      dispatch({ type: orderActions.SET_ERROR, payload: error.message });
    }
  };

  if (orderState.orderComplete) {
    return (
      <div className="order-success">
        <h2>Order Complete!</h2>
        <p>Order ID: {orderState.orderId}</p>
      </div>
    );
  }

  return (
    <div className="checkout-form">
      <h2>Checkout</h2>

      {orderState.error && (
        <div className="error">{orderState.error}</div>
      )}

      <div className="shipping-section">
        <h3>Shipping Information</h3>
        <input
          type="text"
          placeholder="Address"
          value={orderState.shippingInfo.address}
          onChange={(e) =>
            dispatch({
              type: orderActions.UPDATE_SHIPPING,
              payload: { address: e.target.value }
            })
          }
        />
        {/* More shipping fields... */}
      </div>

      <button
        onClick={handleSubmitOrder}
        disabled={orderState.loading}
      >
        {orderState.loading ? 'Processing...' : 'Place Order'}
      </button>
    </div>
  );
}
```

useMemo Hook {#usememo}

The `useMemo` hook returns a memoized value. It's used for expensive calculations that shouldn't run on every render.

E-commerce Example: Price Calculations

javascript

```
import React, { useState, useMemo } from 'react';

function OrderSummary({ items, discountCode, shippingMethod }) {
  const [taxRate] = useState(0.08); // 8% tax rate

  // Expensive calculation - only recalculate when items change
  const subtotal = useMemo(() => {
    console.log('Calculating subtotal...'); // This should only log when items change
    return items.reduce((total, item) => {
      return total + (item.price * item.quantity);
    }, 0);
  }, [items]);

  // Calculate discount - only when subtotal or discountCode changes
  const discount = useMemo(() => {
    if (!discountCode) return 0;

    console.log('Calculating discount...');
    switch (discountCode.toUpperCase()) {
      case 'SAVE10':
        return subtotal * 0.1;
      case 'SAVE20':
        return subtotal * 0.2;
      case 'WELCOME':
        return Math.min(subtotal * 0.15, 50); // Max $50 discount
      default:
        return 0;
    }
  }, [subtotal, discountCode]);

  // Calculate shipping - only when subtotal or shipping method changes
  const shipping = useMemo(() => {
    console.log('Calculating shipping...');
    if (subtotal > 100) return 0; // Free shipping over $100

    switch (shippingMethod) {
      case 'standard':
        return 5.99;
      case 'express':
        return 12.99;
      case 'overnight':
        return 24.99;
      default:
        return 5.99;
    }
  }, [subtotal, shippingMethod]);
  // Calculate tax on discounted subtotal
  const tax = useMemo(() => {
    const taxableAmount = subtotal - discount;
    return taxableAmount * taxRate;
  }, [subtotal, discount, taxRate]);

  // Final total calculation
  const total = useMemo(() => {
    return subtotal - discount + shipping + tax;
  }, [subtotal, discount, shipping, tax]);

  return (
    <div className="order-summary">
      <h3>Order Summary</h3>
      <div className="summary-line">
        <span>Subtotal:</span>
        <span>${subtotal.toFixed(2)}</span>
      </div>
      {discount > 0 && (
        <div className="summary-line discount">
          <span>Discount ({discountCode}):</span>
          <span>-${discount.toFixed(2)}</span>
        </div>
      )}
      <div className="summary-line">
        <span>Shipping:</span>
        <span>{shipping === 0 ? 'FREE' : `$${shipping.toFixed(2)}`}</span>
      </div>
      <div className="summary-line">
        <span>Tax:</span>
        <span>${tax.toFixed(2)}</span>
      </div>
      <div className="summary-line total">
        <span>Total:</span>
        <span>${total.toFixed(2)}</span>
      </div>
    </div>
  );
}
```

When to use useMemo:

- Expensive calculations
- Complex object/array transformations
- Preventing unnecessary re-renders of child components

- Avoiding infinite loops in useEffect dependencies

useCallback Hook {#usecallback}

The `useCallback` hook returns a memoized callback function. It's useful for preventing unnecessary re-renders when passing callbacks to child components.

E-commerce Example: Product List with Filtering

```
javascript
```

```
import React, { useState, useCallback, useMemo } from 'react';

function ProductList({ products }) {
  const [sortBy, setSortBy] = useState('name');
  const [filterCategory, setFilterCategory] = useState('all');
  const [searchTerm, setSearchTerm] = useState('');

  // Memoized filter function - only recreated when dependencies change
  const handleSearch = useCallback((term) => {
    setSearchTerm(term);
  }, []); // No dependencies, function never changes

  // Memoized sort function
  const handleSort = useCallback((sortType) => {
    setSortBy(sortType);
  }, []);

  // Memoized category filter function
  const handleCategoryFilter = useCallback((category) => {
    setFilterCategory(category);
  }, []);

  // Filtered and sorted products
  const filteredProducts = useMemo(() => {
    let filtered = products;

    // Filter by category
    if (filterCategory !== 'all') {
      filtered = filtered.filter(product => product.category === filterCategory);
    }

    // Filter by search term
    if (searchTerm) {
      filtered = filtered.filter(product =>
        product.name.toLowerCase().includes(searchTerm.toLowerCase()) ||
        product.description.toLowerCase().includes(searchTerm.toLowerCase())
      );
    }

    // Sort products
    filtered.sort((a, b) => {
      switch (sortBy) {
        case 'name':
          return a.name.localeCompare(b.name);
        case 'price-low':
          return a.price - b.price;
        case 'price-high':
          return b.price - a.price;
        case 'rating':
          return b.rating - a.rating;
        default:
          return 0;
      }
    });

    return filtered;
  }, [products, filterCategory, searchTerm, sortBy]);

  return (
    <div className="product-list">
      <ProductFilters
        onSearch={handleSearch}
        onSort={handleSort}
        onCategoryFilter={handleCategoryFilter}
        sortBy={sortBy}
        filterCategory={filterCategory}
      />

      <div className="products-grid">
        {filteredProducts.map(product => (
          <ProductCard key={product.id} product={product} />
        ))}
      </div>
    </div>
  );
}

// Child component that receives memoized callbacks
const ProductFilters = React.memo(({
  onSearch,
  onSort,
  onCategoryFilter,
  sortBy,
  filterCategory
}) => {
  console.log('ProductFilters rendered'); // This will only log when props actually change

  return (
    <div className="product-filters">
      <input
        type="text"
        placeholder="Search products..."
        onChange={(e) => onSearch(e.target.value)}
      />

      <select
        value={sortBy}
        onChange={(e) => onSort(e.target.value)}
      >
        <option value="name">Sort by Name</option>
        <option value="price-low">Price: Low to High</option>
        <option value="price-high">Price: High to Low</option>
        <option value="rating">Rating</option>
      </select>

      <select
        value={filterCategory}
        onChange={(e) => onCategoryFilter(e.target.value)}
      >
        <option value="all">All Categories</option>
        <option value="electronics">Electronics</option>
        <option value="clothing">Clothing</option>
        <option value="books">Books</option>
      </select>
    </div>
  );
});
```

useCallback vs useMemo:

- **useCallback**: Memoizes functions
- **useMemo**: Memoizes values (including computed values)

useRef Hook {#useref}

The `useRef` hook returns a mutable ref object. It's commonly used for accessing DOM elements directly or storing mutable values that don't cause re-renders.

E-commerce Example: Focus Management and Timers

```javascript
```

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

function SearchWithSuggestions() {
  const [searchTerm, setSearchTerm] = useState('');
  const [suggestions, setSuggestions] = useState([]);
  const [showSuggestions, setShowSuggestions] = useState(false);
  const [selectedIndex, setSelectedIndex] = useState(-1);

  // Refs for DOM manipulation
  const searchInputRef = useRef(null);
  const suggestionsRef = useRef(null);

  // Ref to store timeout ID (doesn't cause re-renders)
  const debounceTimeoutRef = useRef(null);

  // Ref to store previous search term
  const previousSearchTermRef = useRef('');

  // Focus search input on component mount
  useEffect(() => {
    searchInputRef.current?.focus();
  }, []);

  // Handle search with debouncing
  useEffect(() => {
    if (searchTerm !== previousSearchTermRef.current) {
      // Clear previous timeout
      if (debounceTimeoutRef.current) {
        clearTimeout(debounceTimeoutRef.current);
      }

      // Set new timeout
      debounceTimeoutRef.current = setTimeout(async () => {
        if (searchTerm.length > 2) {
          const response = await fetch(`/api/search-suggestions?q=${searchTerm}`);
          const data = await response.json();
          setSuggestions(data);
          setShowSuggestions(true);
        } else {
          setSuggestions([]);
          setShowSuggestions(false);
        }
      }, 300);

      previousSearchTermRef.current = searchTerm;
    }
  // Cleanup timeout on unmount
  return () => {
    if (debounceTimeoutRef.current) {
      clearTimeout(debounceTimeoutRef.current);
    }
  };
}, [searchTerm]);

const handleKeyDown = (e) => {
  if (!showSuggestions || suggestions.length === 0) return;

  switch (e.key) {
    case 'ArrowDown':
      e.preventDefault();
      setSelectedIndex(prev =>
        prev < suggestions.length - 1 ? prev + 1 : 0
      );
      break;

    case 'ArrowUp':
      e.preventDefault();
      setSelectedIndex(prev =>
        prev > 0 ? prev - 1 : suggestions.length - 1
      );
      break;

    case 'Enter':
      e.preventDefault();
      if (selectedIndex >= 0) {
        selectSuggestion(suggestions[selectedIndex]);
      }
      break;

    case 'Escape':
      setShowSuggestions(false);
      setSelectedIndex(-1);
      break;
  }
};

const selectSuggestion = (suggestion) => {
  setSearchTerm(suggestion.name);
  setShowSuggestions(false);
  setSelectedIndex(-1);
  // Navigate to search results or product page
  window.location.href = `/search?q=${encodeURIComponent(suggestion.name)}`;
  };

  return (
    <div className="search-container">
      <input
        ref={searchInputRef}
        type="text"
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        onKeyDown={handleKeyDown}
        placeholder="Search products..."
        className="search-input"
      />

      {showSuggestions && suggestions.length > 0 && (
        <div ref={suggestionsRef} className="suggestions-list">
          {suggestions.map((suggestion, index) => (
            <div
              key={suggestion.id}
              className={`suggestion-item ${
                index === selectedIndex ? 'selected' : ''
              }`}
              onClick={() => selectSuggestion(suggestion)}
            >
              <img
                src={suggestion.image}
                alt={suggestion.name}
                className="suggestion-image"
              />
              <div className="suggestion-content">
                <div className="suggestion-name">{suggestion.name}</div>
                <div className="suggestion-price">${suggestion.price}</div>
              </div>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}

// Another useRef example: Scroll to element
function ProductReviews({ reviews }) {
  const reviewsEndRef = useRef(null);

  const scrollToBottom = () => {
    reviewsEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  };

  return (
    <div className="product-reviews">
      <h3>Customer Reviews</h3>
      <div className="reviews-list">
        {reviews.map(review => (
          <div key={review.id} className="review-item">
            <h4>{review.title}</h4>
            <p>{review.content}</p>
            <div className="review-rating">
              {'★'.repeat(review.rating)}{'☆'.repeat(5 - review.rating)}
            </div>
          </div>
        ))}
        <div ref={reviewsEndRef} />
      </div>
      <button onClick={scrollToBottom}>
        Scroll to Latest Review
      </button>
    </div>
  );
}
```

useRef Use Cases:

1. **DOM Access**: Focus elements, scroll positions, measuring elements
2. **Storing Mutable Values**: Timers, previous values, counters
3. **Avoiding Re-renders**: Values that change but shouldn't trigger updates
4. **Integration with Third-party Libraries**: Storing library instances

Custom Hooks {#custom-hooks}

Custom hooks allow you to extract component logic into reusable functions. They must start with "use" and can call other hooks.

E-commerce Example: Custom Hooks for Common Functionality

```
javascript
```

```javascript
// Custom hook for API calls with loading and error states
import { useState, useEffect } from 'react';

function useApi(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true);
        setError(null);
        const response = await fetch(url);
        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }
        const result = await response.json();
        setData(result);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error };
}

// Custom hook for local storage
function useLocalStorage(key, initialValue) {
  const [storedValue, setStoredValue] = useState(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      console.error(`Error reading localStorage key "${key}":`, error);
      return initialValue;
    }
  });

  const setValue = (value) => {
    try {
      const valueToStore = value instanceof Function ? value(storedValue) : value;
      setStoredValue(valueToStore);
      window.localStorage.setItem(key, JSON.stringify(valueToStore));
    } catch (error) {
      console.error(`Error setting localStorage key "${key}":`, error);
    }
  };

  return [storedValue, setValue];
}

// Custom hook for wishlist functionality
function useWishlist() {
  const [wishlistItems, setWishlistItems] = useLocalStorage('wishlist', []);

  const addToWishlist = (product) => {
    setWishlistItems(prev => {
      const exists = prev.find(item => item.id === product.id);
      if (exists) return prev;
      return [...prev, product];
    });
  };

  const removeFromWishlist = (productId) => {
    setWishlistItems(prev => prev.filter(item => item.id !== productId));
  };

  const isInWishlist = (productId) => {
    return wishlistItems.some(item => item.id === productId);
  };

  const toggleWishlist = (product) => {
    if (isInWishlist(product.id)) {
      removeFromWishlist(product.id);
    } else {
      addToWishlist(product);
    }
  };

  return {
    wishlistItems,
    addToWishlist,
    removeFromWishlist,
    isInWishlist,
    toggleWishlist
  };
}
```

```
// Custom hook for form handling
function useForm(initialValues, validate) {
  const [values, setValues] = useState(initialValues);
  const [errors, setErrors] = useState({});
  const [touched, setTouched] = useState({});

  const handleChange = (name, value) => {
    setValues(prev => ({ ...prev, [name]: value }));

    // Clear error when user starts typing
    if (errors[name]) {
      setErrors(prev => ({ ...prev, [name]: '' }));
    }
  };

  const handleBlur = (name) => {
    setTouched(prev => ({ ...prev, [name]: true }));

    if (validate) {
      const fieldErrors = validate({ ...values });
      setErrors(prev => ({ ...prev, [name]: fieldErrors[name] || '' }));
    }
  };

  const handleSubmit = (onSubmit) => {
    return (e) => {
      e.preventDefault();

      // Mark all fields as touched
      const touchedFields = {};
      Object.keys(values).forEach(key => {
        touchedFields[key] = true;
      });
      setTouched(touchedFields);

      // Validate all fields
      if (validate) {
        const formErrors = validate(values);
        setErrors(formErrors);

        // If there are errors, don't submit
        if (Object.keys(formErrors).length > 0) {
          return;
        }
      }
      // Submit the form
      onSubmit(values);
    };
  };

  const resetForm = () => {
    setValues(initialValues);
    setErrors({});
    setTouched({});
  };

  return {
    values,
    errors,
    touched,
    handleChange,
    handleBlur,
    handleSubmit,
    resetForm
  };
}

// Usage of custom hooks in components
function ProductPage({ productId }) {
  // Using useApi custom hook
  const { data: product, loading, error } = useApi(`/api/products/${productId}`);

  // Using useWishlist custom hook
  const { isInWishlist, toggleWishlist } = useWishlist();

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;
  if (!product) return <div>Product not found</div>;

  return (
    <div className="product-page">
      <h1>{product.name}</h1>
      <p>${product.price}</p>
      <button
        onClick={() => toggleWishlist(product)}
        className={isInWishlist(product.id) ? 'in-wishlist' : ''}
      >
        {isInWishlist(product.id) ? '♥ Remove from Wishlist' : '♡ Add to Wishlist'}
      </button>
    </div>
  );
}
```

```
function ContactForm() {
  const initialValues = {
    name: '',
    email: '',
    message: ''
  };

  const validate = (values) => {
    const errors = {};
    if (!values.name) errors.name = 'Name is required';
    if (!values.email) errors.email = 'Email is required';
    else if (!/\S+@\S+\.\S+/.test(values.email)) errors.email = 'Email is invalid';
    if (!values.message) errors.message = 'Message is required';
    return errors;
  };

  const { values, errors, touched, handleChange, handleBlur, handleSubmit, resetForm } =
    useForm(initialValues, validate);

  const onSubmit = async (formData) => {
    try {
      await fetch('/api/contact', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData)
      });
      alert('Message sent successfully!');
      resetForm();
    } catch (error) {
      alert('Failed to send message');
    }
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <input
          type="text"
          placeholder="Your Name"
          value={values.name}
          onChange={(e) => handleChange('name', e.target.value)}
          onBlur={() => handleBlur('name')}
        />
        {touched.name && errors.name && <span className="error">{errors.name}</span>}
      </div>
      <div>
        <input
          type="email"
          placeholder="Your Email"
          value={values.email}
          onChange={(e) => handleChange('email', e.target.value)}
          onBlur={() => handleBlur('email')}
        />
        {touched.email && errors.email && <span className="error">{errors.email}</span>}
      </div>

      <div>
        <textarea
          placeholder="Your Message"
          value={values.message}
          onChange={(e) => handleChange('message', e.target.value)}
          onBlur={() => handleBlur('message')}
        />
        {touched.message && errors.message && <span className="error">{errors.message}</span>}
      </div>

      <button type="submit">Send Message</button>
    </form>
  );
}
```

Complete E-commerce Application {#complete-app}

Now let's put everything together in a complete e-commerce application that demonstrates all the hooks we've covered:

```javascript
```

```
import React, { createContext, useContext, useReducer, useState, useEffect, useMemo, useCallback, useRef } from 'react'

// ============= CONTEXT AND REDUCER =============
const EcommerceContext = createContext();

const initialState = {
  products: [],
  cart: { items: [] },
  user: null,
  loading: false,
  error: null,
  filters: {
    category: 'all',
    priceRange: [0, 1000],
    sortBy: 'name'
  }
};

function ecommerceReducer(state, action) {
  switch (action.type) {
    case 'SET_LOADING':
      return { ...state, loading: action.payload };

    case 'SET_ERROR':
      return { ...state, error: action.payload, loading: false };

    case 'SET_PRODUCTS':
      return { ...state, products: action.payload, loading: false };

    case 'ADD_TO_CART':
      const existingItem = state.cart.items.find(item => item.id === action.payload.id);
      if (existingItem) {
        return {
          ...state,
          cart: {
            ...state.cart,
            items: state.cart.items.map(item =>
              item.id === action.payload.id
                ? { ...item, quantity: item.quantity + 1 }
                : item
            )
          }
        };
      }
      return {
        ...state,
        cart: {
          ...state.cart,
          items: [...state.cart.items, { ...action.payload, quantity: 1 }]
        }
      };

    case 'REMOVE_FROM_CART':
      return {
        ...state,
        cart: {
          ...state.cart,
          items: state.cart.items.filter(item => item.id !== action.payload)
        }
      };

    case 'UPDATE_FILTERS':
      return {
        ...state,
        filters: { ...state.filters, ...action.payload }
      };

    case 'SET_USER':
      return { ...state, user: action.payload };

    default:
      return state;
  }
}

// ============= PROVIDER COMPONENT =============
export function EcommerceProvider({ children }) {
  const [state, dispatch] = useReducer(ecommerceReducer, initialState);

  // Load products on mount
  useEffect(() => {
    const loadProducts = async () => {
      dispatch({ type: 'SET_LOADING', payload: true });
      try {
        // Simulated API call
        const response = await new Promise(resolve => {
          setTimeout(() => {
            resolve([
              { id: 1, name: 'Laptop', price: 999, category: 'electronics', rating: 4.5, image: '/laptop.jpg' },
              { id: 2, name: 'T-Shirt', price: 29, category: 'clothing', rating: 4.0, image: '/tshirt.jpg' },
              { id: 3, name: 'Book', price: 15, category: 'books', rating: 4.8, image: '/book.jpg' },
              { id: 4, name: 'Headphones', price: 199, category: 'electronics', rating: 4.3, image: '/headphones.jpg' },
              { id: 5, name: 'Jeans', price: 79, category: 'clothing', rating: 4.2, image: '/jeans.jpg' }
            ]);
          }, 1000);
        });
        dispatch({ type: 'SET_PRODUCTS', payload: response });
      } catch (error) {
        dispatch({ type: 'SET_ERROR', payload: error.message });
      }
    };

    loadProducts();
  }, []);

  // Memoized cart calculations
  const cartTotal = useMemo(() => {
    return state.cart.items.reduce((total, item) => total + (item.price * item.quantity), 0);
  }, [state.cart.items]);

  const cartItemCount = useMemo(() => {
    return state.cart.items.reduce((count, item) => count + item.quantity, 0);
  }, [state.cart.items]);

  // Filtered products based on current filters
  const filteredProducts = useMemo(() => {
    let filtered = state.products;

    if (state.filters.category !== 'all') {
      filtered = filtered.filter(product => product.category === state.filters.category);
    }

    filtered = filtered.filter(product =>
      product.price >= state.filters.priceRange[0] &&
      product.price <= state.filters.priceRange[1]
    );

    filtered.sort((a, b) => {
      switch (state.filters.sortBy) {
        case 'price-low':
          return a.price - b.price;
        case 'price-high':
          return b.price - a.price;
        case 'rating':
          return b.rating - a.rating;
        default:
          return a.name.localeCompare(b.name);
      }
    });
```

```
    return filtered;
  }, [state.products, state.filters]);

  const value = {
    state,
    dispatch,
    cartTotal,
    cartItemCount,
    filteredProducts
  };

  return (
    <EcommerceContext.Provider value={value}>
      {children}
    </EcommerceContext.Provider>
  );
}

// ============= CUSTOM HOOK =============
function useEcommerce() {
  const context = useContext(EcommerceContext);
  if (!context) {
    throw new Error('useEcommerce must be used within EcommerceProvider');
  }
  return context;
}

// ============= COMPONENTS =============

// Header with cart icon
function Header() {
  const { cartItemCount } = useEcommerce();

  return (
    <header className="header">
      <h1>My E-Store</h1>
      <div className="cart-icon">
        🛒 ({cartItemCount})
      </div>
    </header>
  );
}

// Product filters
function ProductFilters() {
  const { state, dispatch } = useEcommerce();
  const handleCategoryChange = useCallback((category) => {
    dispatch({ type: 'UPDATE_FILTERS', payload: { category } });
  }, [dispatch]);

  const handleSortChange = useCallback((sortBy) => {
    dispatch({ type: 'UPDATE_FILTERS', payload: { sortBy } });
  }, [dispatch]);

  const handlePriceRangeChange = useCallback((priceRange) => {
    dispatch({ type: 'UPDATE_FILTERS', payload: { priceRange } });
  }, [dispatch]);

  return (
    <div className="filters">
      <select
        value={state.filters.category}
        onChange={(e) => handleCategoryChange(e.target.value)}
      >
        <option value="all">All Categories</option>
        <option value="electronics">Electronics</option>
        <option value="clothing">Clothing</option>
        <option value="books">Books</option>
      </select>

      <select
        value={state.filters.sortBy}
        onChange={(e) => handleSortChange(e.target.value)}
      >
        <option value="name">Sort by Name</option>
        <option value="price-low">Price: Low to High</option>
        <option value="price-high">Price: High to Low</option>
        <option value="rating">Rating</option>
      </select>

      <div className="price-range">
        <label>Max Price: ${state.filters.priceRange[1]}</label>
        <input
          type="range"
          min="0"
          max="1000"
          value={state.filters.priceRange[1]}
          onChange={(e) => handlePriceRangeChange([0, parseInt(e.target.value)])}
        />
      </div>
    </div>
  );
}
```

```
// Individual product card
function ProductCard({ product }) {
  const { dispatch } = useEcommerce();
  const [isAdding, setIsAdding] = useState(false);

  const handleAddToCart = useCallback(async () => {
    setIsAdding(true);
    // Simulate API call delay
    await new Promise(resolve => setTimeout(resolve, 500));
    dispatch({ type: 'ADD_TO_CART', payload: product });
    setIsAdding(false);
  }, [dispatch, product]);

  return (
    <div className="product-card">
      <img src={product.image} alt={product.name} />
      <h3>{product.name}</h3>
      <p className="price">${product.price}</p>
      <div className="rating">
        {'★'.repeat(Math.floor(product.rating))}
        {'☆'.repeat(5 - Math.floor(product.rating))}
        ({product.rating})
      </div>
      <button
        onClick={handleAddToCart}
        disabled={isAdding}
        className="add-to-cart-btn"
      >
        {isAdding ? 'Adding...' : 'Add to Cart'}
      </button>
    </div>
  );
}

// Product list with search
function ProductList() {
  const { state, filteredProducts } = useEcommerce();
  const [searchTerm, setSearchTerm] = useState('');
  const searchInputRef = useRef(null);

  // Focus search input on mount
  useEffect(() => {
    searchInputRef.current?.focus();
  }, []);

  // Filter products by search term
  const searchedProducts = useMemo(() => {
    if (!searchTerm) return filteredProducts;

    return filteredProducts.filter(product =>
      product.name.toLowerCase().includes(searchTerm.toLowerCase())
    );
  }, [filteredProducts, searchTerm]);

  if (state.loading) {
    return <div className="loading">Loading products...</div>;
  }

  if (state.error) {
    return <div className="error">Error: {state.error}</div>;
  }

  return (
    <div className="product-list">
      <div className="search-bar">
        <input
          ref={searchInputRef}
          type="text"
          placeholder="Search products..."
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
        />
      </div>

      <ProductFilters />

      <div className="products-grid">
        {searchedProducts.map(product => (
          <ProductCard key={product.id} product={product} />
        ))}
      </div>

      {searchedProducts.length === 0 && !state.loading && (
        <div className="no-products">No products found</div>
      )}
    </div>
  );
}

// Shopping cart
function ShoppingCart() {
  const { state, dispatch, cartTotal } = useEcommerce();
  const [isOpen, setIsOpen] = useState(false);
  const removeFromCart = useCallback((productId) => {
    dispatch({ type: 'REMOVE_FROM_CART', payload: productId });
  }, [dispatch]);

  return (
    <div className="shopping-cart">
      <button
        className="cart-toggle"
        onClick={() => setIsOpen(!isOpen)}
      >
        Cart ({state.cart.items.length})
      </button>

      {isOpen && (
        <div className="cart-dropdown">
          <h3>Shopping Cart</h3>
          {state.cart.items.length === 0 ? (
            <p>Your cart is empty</p>
          ) : (
            <>
              {state.cart.items.map(item => (
                <div key={item.id} className="cart-item">
                  <span>{item.name}</span>
                  <span>Qty: {item.quantity}</span>
                  <span>${(item.price * item.quantity).toFixed(2)}</span>
                  <button onClick={() => removeFromCart(item.id)}>Remove</button>
                </div>
              ))}
              <div className="cart-total">
                <strong>Total: ${cartTotal.toFixed(2)}</strong>
              </div>
              <button className="checkout-btn">Proceed to Checkout</button>
            </>
          )}
        </div>
      )}
    </div>
  );
}

// Main App component
function App() {
  return (
    <EcommerceProvider>
      <div className="app">
        <Header />
        <main className="main-content">
          <ProductList />
          <ShoppingCart />
        </main>
      </div>
    </EcommerceProvider>
  );
}

export default App;
```

Key Takeaways and Best Practices

1. Hook Selection Guidelines

- **useState**: Simple state values and updates
- **useReducer**: Complex state logic with multiple related values
- **useEffect**: Side effects, API calls, subscriptions, cleanup
- **useContext**: Avoiding prop drilling, global state
- **useMemo**: Expensive calculations, preventing unnecessary re-computations
- **useCallback**: Preventing unnecessary re-renders of child components
- **useRef**: DOM access, storing mutable values without re-renders

2. Performance Optimization

- Use `React.memo` with `useCallback` and `useMemo` for component optimization
- Don't overuse `useMemo` and `useCallback` - they have their own overhead
- Profile your app to identify actual performance bottlenecks

3. Common Pitfalls to Avoid

- **Stale Closures**: Always use functional updates in useState when depending on previous state
- **Missing Dependencies**: Include all dependencies in useEffect dependency arrays
- **Infinite Loops**: Be careful with object/array dependencies in useEffect
- **Overusing Context**: Don't put everything in context - it can cause unnecessary re-renders

4. Testing Hooks

- Test custom hooks with React Testing Library's `renderHook`
- Test components that use hooks as integration tests
- Mock API calls and external dependencies

5. TypeScript with Hooks

```typescript
// Type your state
const [user, setUser] = useState<User | null>(null);

// Type your custom hooks
function useApi<T>(url: string): { data: T | null; loading: boolean; error: string | null } {
  // Implementation
}
```

This comprehensive guide covers all major React Hooks with practical e-commerce examples. Each hook is demonstrated with real-world use cases, showing when and how to use them effectively. The complete application at the end ties everything together, showing how hooks work in harmony to create a functional, maintainable React application.