1. ProductForm Tests

`src/components/ProductForm.test.js`

Test that the form renders correctly with all inputs.
Test that `onAddProduct` is called when the form is submitted.
Test that `onCancel` is called when the Cancel button is clicked.

```
import React from 'react';
import { render, fireEvent } from '@testing-library/react';
import ProductForm from './ProductForm';

describe('ProductForm', () => {
  it('renders the form correctly', () => {
    const { getByLabelText, getByText } =
render(<ProductForm onAddProduct={() => {}} onCancel={() =>
{}} />);
    expect(getByLabelText(/Name:/i)).toBeInTheDocument();
    expect(getByLabelText(/Description:/
i)).toBeInTheDocument();
    expect(getByLabelText(/Price:/i)).toBeInTheDocument();
    expect(getByLabelText(/Stock:/i)).toBeInTheDocument();
    expect(getByText(/Add Product/i)).toBeInTheDocument();
    expect(getByText(/Cancel/i)).toBeInTheDocument();
  });

  it('calls onAddProduct when the form is submitted', () =>
{
    const mockAddProduct = jest.fn();
    const { getByLabelText, getByText } =
render(<ProductForm onAddProduct={mockAddProduct}
onCancel={() => {}} />);

    fireEvent.change(getByLabelText(/Name:/i), { target:
{ value: 'New Product' } });
    fireEvent.change(getByLabelText(/Description:/i),
{ target: { value: 'New Description' } });
    fireEvent.change(getByLabelText(/Price:/i), { target: {
value: '$100' } });
```

```
    fireEvent.change(getByLabelText(/Stock:/i), { target: {
value: '10' } });
    fireEvent.click(getByText(/Add Product/i));

    expect(mockAddProduct).toHaveBeenCalled();
  });

  it('calls onCancel when the Cancel button is clicked', ()
=> {
    const mockCancel = jest.fn();
    const { getByText } = render(<ProductForm
onAddProduct={() => {}} onCancel={mockCancel} />);
    fireEvent.click(getByText(/Cancel/i));
    expect(mockCancel).toHaveBeenCalled();
  });
});
```

Explanation

1. **`renders the form correctly`**:
    - This test checks that the form is rendered correctly by verifying the presence of all required form fields (Name, Description, Price, Stock) and buttons (Add Product, Cancel).
    - Uses `getByLabelText` to find input fields and `getByText` to find buttons.
2. **`calls onAddProduct when the form is submitted`**:
    - This test simulates user interaction by filling out the form and submitting it.
    - Uses `fireEvent.change` to simulate typing into input fields.
    - Uses `fireEvent.click` to simulate clicking the Add Product button.
    - Verifies that the `onAddProduct` callback is called when the form is submitted.
3. **`calls onCancel when the Cancel button is clicked`**:
    - This test simulates clicking the Cancel button.
    - Uses `fireEvent.click` to simulate clicking the Cancel button.
    - Verifies that the `onCancel` callback is called when the Cancel button is clicked.

2. Product Tests

src/components/Product.test.js

Test that the product details render correctly.
Test that `onBack` is called when the Back button is clicked.
Test that the product is updated when Save is clicked.

```
import React from 'react';
import { render, fireEvent } from '@testing-library/react';
import Product from './Product';

describe('Product', () => {
  const mockProduct = {
    id: 1,
    name: 'React Book',
    description: 'Great book about React.',
    price: '$120',
    stock: 10,
  };

  it('renders the product details correctly', () => {
    const { getByLabelText, getByText } = render(<Product
product={mockProduct} onBack={() => {}} />);
    expect(getByLabelText(/Name:/
i)).toHaveValue(mockProduct.name);
    expect(getByLabelText(/Description:/
i)).toHaveValue(mockProduct.description);
    expect(getByLabelText(/Price:/
i)).toHaveValue(mockProduct.price);
    expect(getByLabelText(/Stock:/
i)).toHaveValue(mockProduct.stock.toString());
    expect(getByText(/Save/i)).toBeInTheDocument();
    expect(getByText(/Back to list/i)).toBeInTheDocument();
  });

  it('calls onBack when the Back button is clicked', () =>
{
    const mockBack = jest.fn();
    const { getByText } = render(<Product
product={mockProduct} onBack={mockBack} />);
    fireEvent.click(getByText(/Back to list/i));
    expect(mockBack).toHaveBeenCalled();
  });
  it('updates the product when Save is clicked', () => {
    const { getByLabelText, getByText } = render(<Product 
product={mockProduct} onBack={() => {}} />);
    fireEvent.change(getByLabelText(/Name:/i), { target: 
{ value: 'Updated Product' } });
    fireEvent.click(getByText(/Save/i));
    expect(getByLabelText(/Name:/i)).toHaveValue('Updated 
Product');
  });
});
```

Explanation

1. **`renders the product details correctly`**:
    - This test checks that the product details are rendered correctly by verifying the values of the form fields (Name, Description, Price, Stock) match the provided product data.
    - Uses `getByLabelText` to find input fields and `getByText` to find buttons.
2. **calls `onBack` when the Back button is clicked**:
    - This test simulates clicking the Back button.
    - Uses `fireEvent.click` to simulate clicking the Back button.
    - Verifies that the `onBack` callback is called when the Back button is clicked.
3. **updates the product when Save is clicked**:
    - This test simulates editing a product by changing the value of the Name field and clicking the Save button.
    - Uses `fireEvent.change` to simulate typing into the input field.
    - Uses `fireEvent.click` to simulate clicking the Save button.
    - Verifies that the value of the Name field is updated after saving.

3. ProductList Tests

`src/components/ProductList.test.js`

Test that products are fetched and displayed.
Test that `onSelectProduct` is called when a product is clicked.
Test that a product is deleted when the Delete button is clicked.

```
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-
library/react';
import ProductList from './ProductList';

describe('ProductList', () => {
  beforeEach(() => {
    fetch.resetMocks();
  });

  it('fetches and displays products', async () => {
    fetch.mockResponseOnce(JSON.stringify([
      { id: 1, name: 'React Book' },
      { id: 2, name: 'ES6 Book' },
    ]));

    const { getByText } = render(<ProductList
onSelectProduct={() => {}} />);
    await waitFor(() => expect(getByText(/React Book/
i)).toBeInTheDocument());
    expect(getByText(/ES6 Book/i)).toBeInTheDocument();
  });

  it('calls onSelectProduct when a product is clicked',
async () => {
    fetch.mockResponseOnce(JSON.stringify([
      { id: 1, name: 'React Book' },
    ]));

    const mockSelectProduct = jest.fn();
    const { getByText } = render(<ProductList
onSelectProduct={mockSelectProduct} />);
    await waitFor(() => fireEvent.click(getByText(/React
Book/i)));
    expect(mockSelectProduct).toHaveBeenCalledWith({ id: 1,
name: 'React Book' });
  });

  it('deletes a product when the Delete button is clicked',
async () => {
    fetch.mockResponseOnce(JSON.stringify([
      { id: 1, name: 'React Book' },
    ]));
    fetch.mockResponseOnce(JSON.stringify({}), { status:
200 });

    const { getByText, queryByText } = render(<ProductList
onSelectProduct={() => {}} />);
    await waitFor(() => fireEvent.click(getByText(/Delete/
i)));
    await waitFor(() => expect(queryByText(/React Book/
i)).not.toBeInTheDocument());
  });
});
```

Explanation

1. **`fetches and displays products`**:
   - This test checks that the products are fetched from the API and displayed in the list.
   - Uses `fetch.mockResponseOnce` to mock the API response with a list of products.
   - Uses `waitFor` to wait for the products to be rendered in the DOM.
   - Verifies that the product names are displayed in the list.
2. **calls `onSelectProduct` when a product is clicked**:
   - This test simulates clicking a product in the list.
   - Uses fetch.mockResponseOnce to mock the API response with a single product.
   - Uses fireEvent.click to simulate clicking the product.
   - Verifies that the onSelectProduct callback is called with the selected product when a product is clicked.
3. **deletes a product when the Delete button is clicked**:
   - This test simulates deleting a product.
   - Uses `fetch.mockResponseOnce` to mock the API response with a list of products and a successful delete response.
   - Uses `fireEvent.click` to simulate clicking the Delete button.
   - Uses `waitFor` to wait for the product to be removed from the DOM.
   - Verifies that the product is no longer in the list after deletion.

Running the Tests

To run the tests, use the following command:

```
npm test
```

This command will run all the tests in your project and display the results in the terminal. The tests ensure that your components render correctly, handle user interactions, and interact with the API as expected.