Testing for Online Bookstore Application

Conduct end-to-end testing for an Online Bookstore application using React Testing Library.

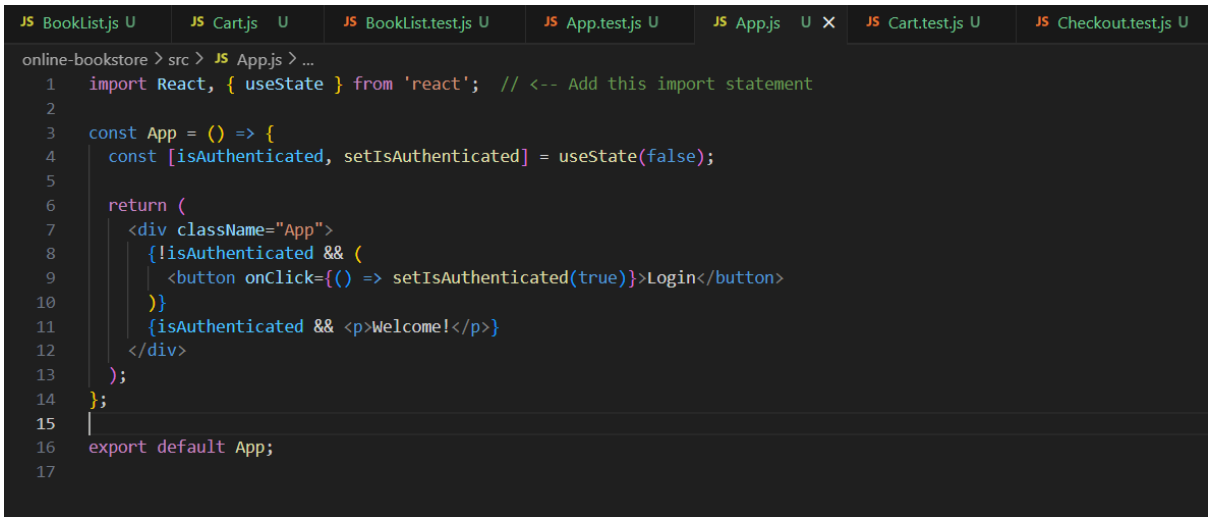
1. Create a React application for the Online Bookstore.
2. Develop components for browsing books, managing the cart, and completing purchases.
3. Write end-to-end tests to cover user flows such as adding books to the cart, updating quantities, and checking out.
4. Test form submissions, input validations, and error handling.
5. Use React Testing Library along with Jest for test assertions and interactions.
6. Ensure proper cleanup and teardown of test environments.
7. Implement tests for user authentication and authorization scenarios.
8. Explore the use of testing utilities like waitFor and waitForElementToBeRemoved for handling asynchronous operations.
9. Integrate code coverage tools to monitor testing effectiveness and identify areas for improvement.

Solution –

App.js

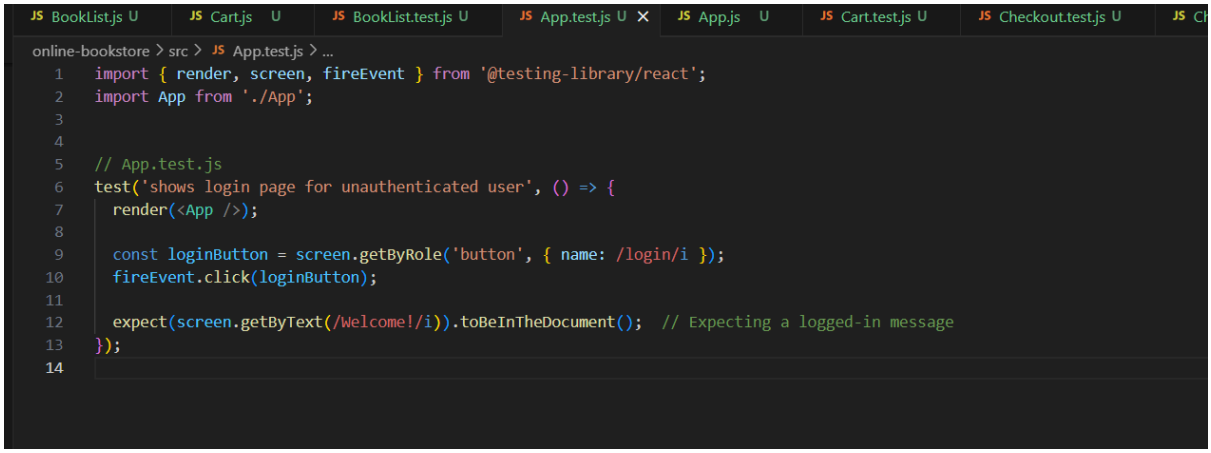
```
import React, { useState } from 'react';  // <-- Add this import statement


const App = () => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);


  return (
    <div className="App">
      {!isAuthenticated && (
        <button onClick={() => setIsAuthenticated(true)}>Login</button>
      )}
```

```
      {isAuthenticated && <p>Welcome!</p>}
    </div>
  );
};

export default App;
```



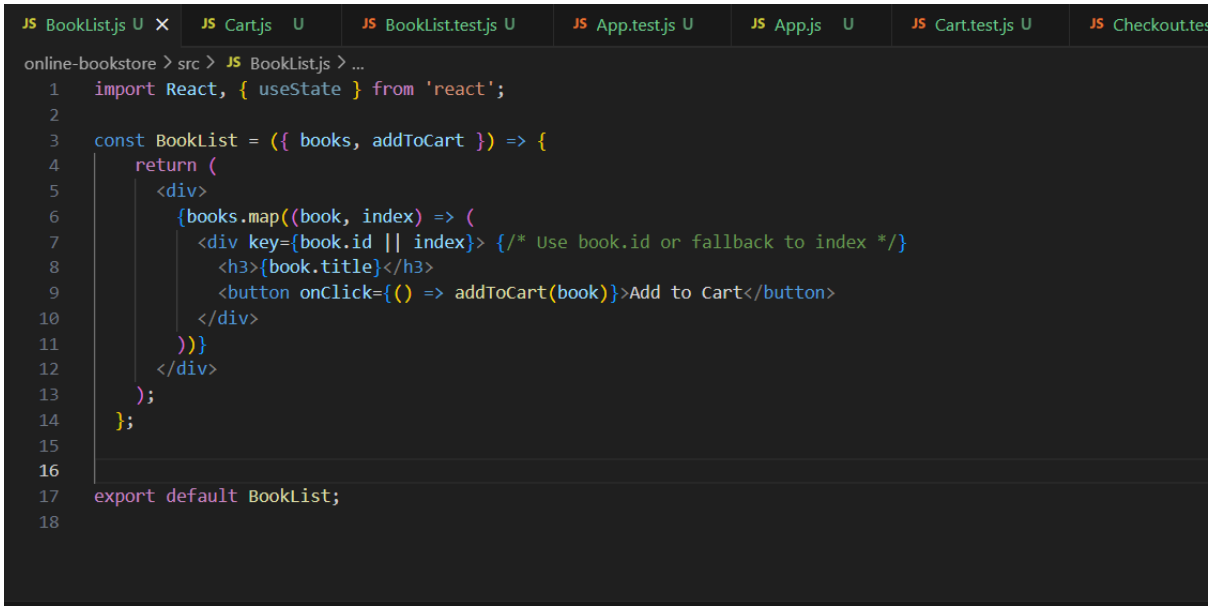
App.test.js

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

// App.test.js
test('shows login page for unauthenticated user', () => {
  render(<App />);

  const loginButton = screen.getByRole('button', { name: /login/i });
  fireEvent.click(loginButton);
  expect(screen.getByText(/Welcome!/i)).toBeInTheDocument();  // Expecting a logged-in message
});
```



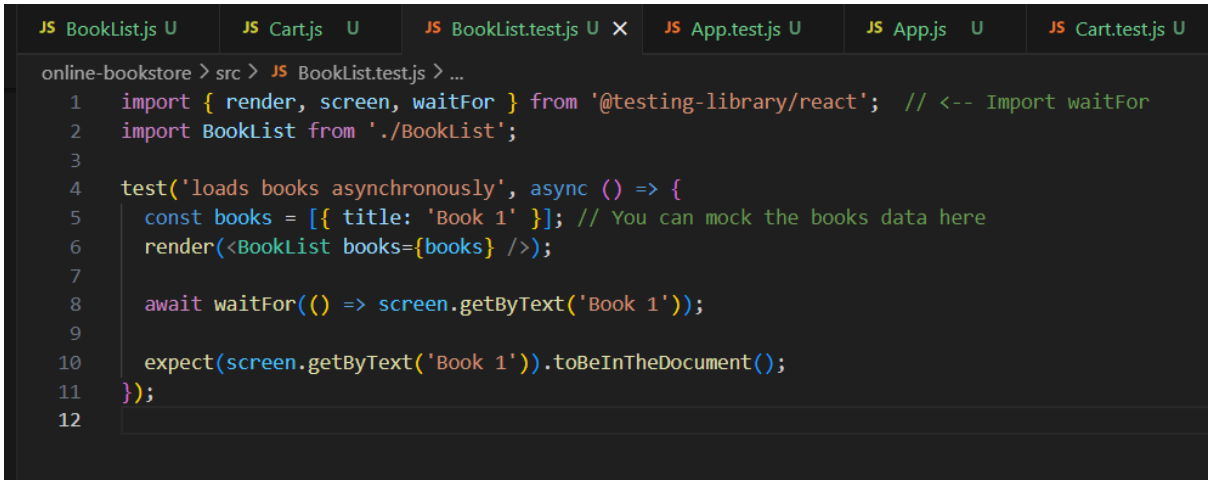
BookList.js

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

const BookList = ({ books, addToCart }) => {
  return (
    <div>
      {books.map((book, index) => (
        <div key={book.id || index}> {/* Use book.id or fallback to index */}
          <h3>{book.title}</h3>
          <button onClick={() => addToCart(book)}>Add to Cart</button>
        </div>
      ))}
    </div>
  );
};

export default BookList;
```



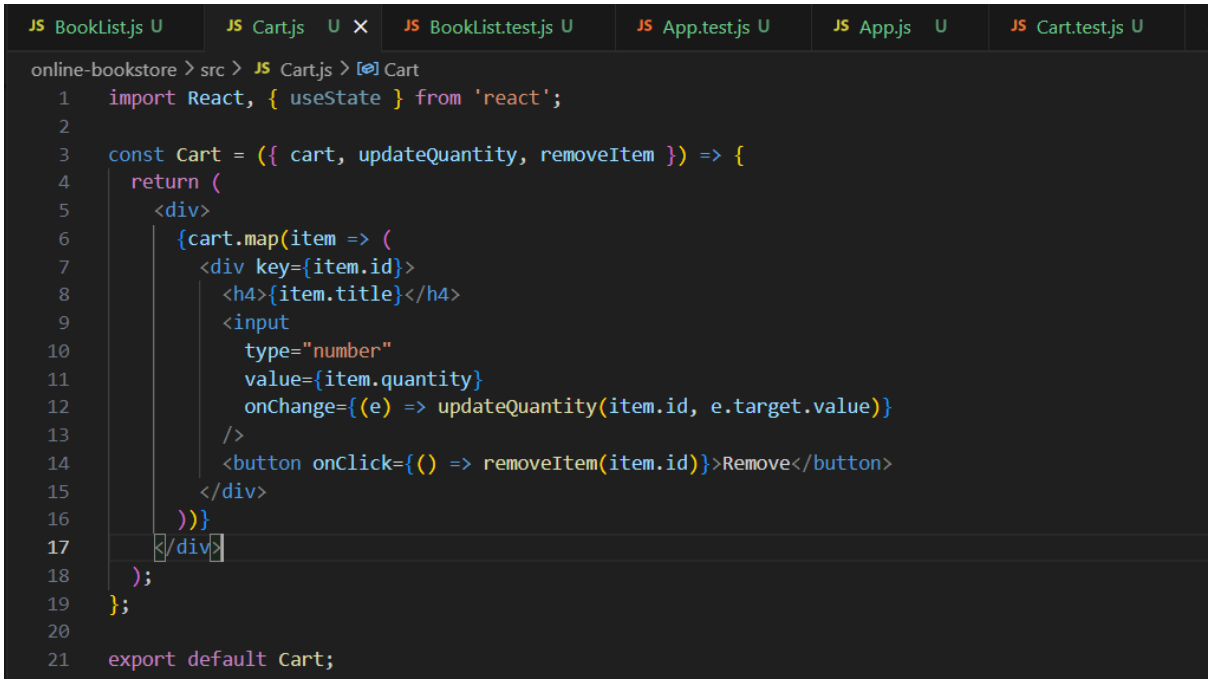
BookList.test.js

```
import { render, screen, waitFor } from '@testing-library/react';
import BookList from './BookList';
test('loads books asynchronously', async () => {
  const books = [{ title: 'Book 1' }]; // You can mock the books data here
  render(<BookList books={books} />);
  await waitFor(() => screen.getByText('Book 1'));
  expect(screen.getByText('Book 1')).toBeInTheDocument();
});
```



Cart.js

```
import React, { useState } from 'react';
const Cart = ({ cart, updateQuantity, removeItem }) => {
  return (
    <div>
      {cart.map(item => (
        <div key={item.id}>
          <h4>{item.title}</h4>
          <input
            type="number"
            value={item.quantity}
            onChange={(e) => updateQuantity(item.id, e.target.value)}
          />
          <button onClick={() => removeItem(item.id)}>Remove</button>
        </div>
      ))}
    </div>
  );
};
export default Cart;
```

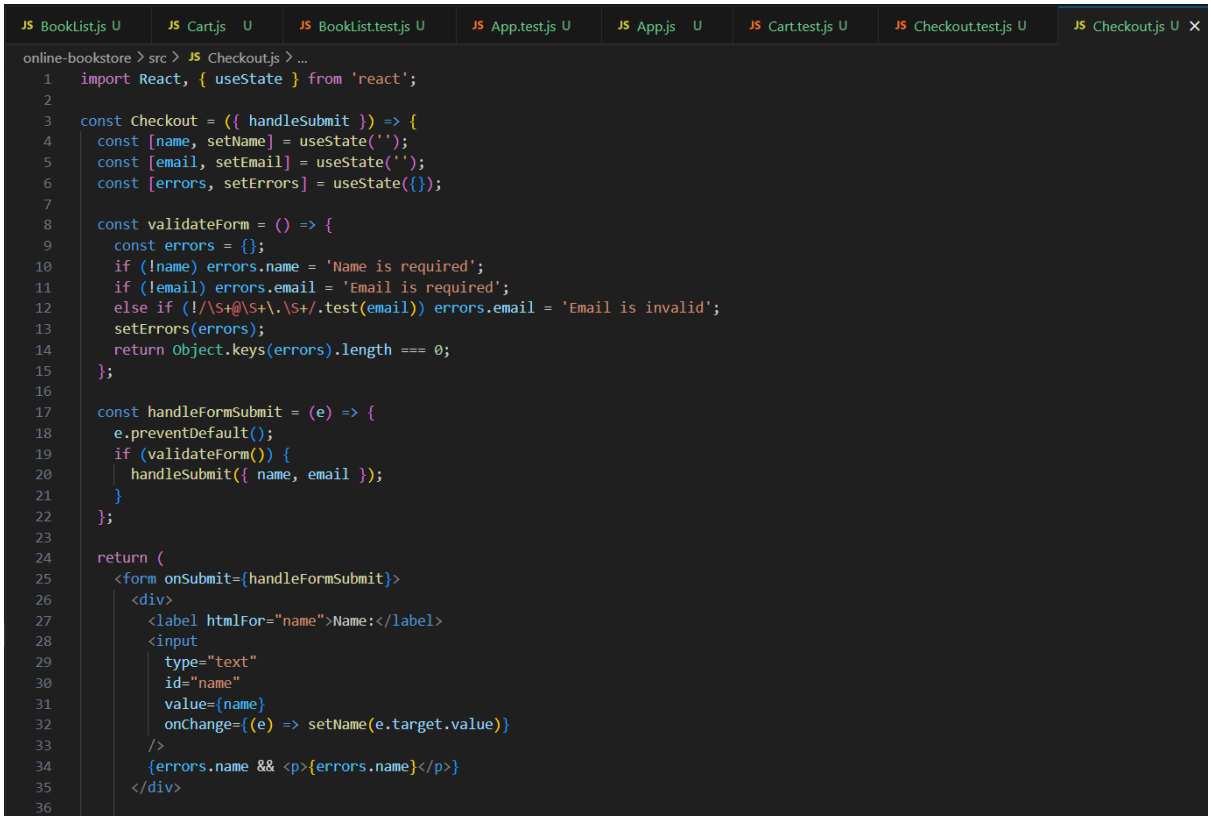


Checkout.js

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

const Checkout = ({ handleSubmit }) => {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [errors, setErrors] = useState({});

  const validateForm = () => {
    const errors = {};
    if (!name) errors.name = 'Name is required';
    if (!email) errors.email = 'Email is required';
    else if (!/\S+@\S+\.\S+/.test(email)) errors.email = 'Email is invalid';
    setErrors(errors);
    return Object.keys(errors).length === 0;
  };
```

```
const handleFormSubmit = (e) => {
  e.preventDefault();
  if (validateForm()) {
    handleSubmit({ name, email });
  }
};

return (
  <form onSubmit={handleFormSubmit}>
    <div>
      <label htmlFor="name">Name:</label>
      <input
        type="text"
        id="name"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      {errors.name && <p>{errors.name}</p>}
    </div>

    <div>
      <label htmlFor="email">Email:</label>
      <input
        type="email"
        id="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
```

```
        />
        {errors.email && <p>{errors.email}</p>}
      </div>

      <button type="submit">Submit</button>
    </form>
  );
};

export default Checkout;
```



Checkout.test.js

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

test('form submits with valid data', () => {
  const handleSubmit = jest.fn();

  render(<Checkout handleSubmit={handleSubmit} />);

  fireEvent.change(screen.getByLabelText(/Name/i), { target: { value: 'John Doe' } });
  fireEvent.change(screen.getByLabelText(/Email/i), { target: { value: 'john@example.com' } });
  fireEvent.click(screen.getByText(/Submit/i));

  expect(handleSubmit).toHaveBeenCalledWith({
    name: 'John Doe',
    email: 'john@example.com',
  });
});
```

BookList.js U | Cart.js U | BookList.test.js U | App.test.js U | App.js U | Cart.test.js U | Checkout.test.js U

online-bookstore > src > Checkout.test.js > ...

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

test('form submits with valid data', () => {
  const handleSubmit = jest.fn();

  render(<Checkout handleSubmit={handleSubmit} />);

  fireEvent.change(screen.getByLabelText(/Name/i), { target: { value: 'John Doe' } });
  fireEvent.change(screen.getByLabelText(/Email/i), { target: { value: 'john@example.com' } });
  fireEvent.click(screen.getByText(/Submit/i));

  expect(handleSubmit).toHaveBeenCalledWith({
    name: 'John Doe',
    email: 'john@example.com',
  });
});

```

OUTPUT-

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Test Suites: 4 passed, 4 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        4.999 s
Ran all test suites related to changed files.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```