**Aim:** Implementing data communication between multiple screens using props and call-back functions.

**Description:**

In React, props and callback functions are commonly used to manage communication between components, especially when handling data flow from parent to child components and vice versa. This technique is essential for building dynamic and interactive user interfaces. We will focus on passing data between different screens (components) using props and callback functions to facilitate real-time updates and data management across screens.

 **Main Features**:

* A product list with an "Add to Cart" button.
* A shopping cart screen where you can:
  + View products added to the cart.
  + Adjust quantities of products.
  + Remove products from the cart.
* Total price updates dynamically.

 **Data Flow**:

* Use props to pass data and callback functions for communication between components.

**Component Structure**

1. **App**:
   * Parent component that holds the state and manages data.
2. **ProductList**:
   * Displays available products and lets the user add products to the cart.
3. **Cart**:
   * Displays products in the cart, allows quantity adjustment, and calculates the total.

**Objective:**

* To learn how to pass data between different components (screens) in React.
* To understand how to use props to send data from a parent component to child components.
* To use callback functions for passing data from child components back to the parent.
* To implement a practical example where multiple screens interact using props and callback functions.

**Implementation:**

1. Basic Setup

Let’s create a simple React app where we have two screens:

* ParentScreen: This will be the main screen managing the state.
* ChildScreen: This will be the secondary screen that communicates with the parent via props and callback functions.

Explanation:

1. Parent to Child Communication (Props):
   * The ParentScreen component defines a piece of state (data) and passes it down to the ChildScreen component as a prop.
   * The ChildScreen can access and display this data.
2. Child to Parent Communication (Callback Functions):
   * The ParentScreen component also defines a callback function (handleDataChange) that updates its state.
   * The ChildScreen component accepts this function as a prop (onDataChange), and when the user interacts with the input field and clicks the "Send Data to Parent" button, it triggers the callback to update the parent's state.

**Index.html**

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="utf-8" />*

*<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500&display=swap" rel="stylesheet">*

*<link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />*

*<meta name="viewport" content="width=device-width, initial-scale=1" />*

*<meta name="theme-color" content="#000000" />*

*<meta*

*name="description"*

*content="Web site created using create-react-app"*

*/>*

*<link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />*

*<!--*

*manifest.json provides metadata used when your web app is installed on a*

*user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/*

*-->*

*<link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />*

*<!--*

*Notice the use of %PUBLIC\_URL% in the tags above.*

*It will be replaced with the URL of the `public` folder during the build.*

*Only files inside the `public` folder can be referenced from the HTML.*

*Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will*

*work correctly both with client-side routing and a non-root public URL.*

*Learn how to configure a non-root public URL by running `npm run build`.*

*-->*

*<title>React App</title>*

*</head>*

*<body>*

*<noscript>You need to enable JavaScript to run this app.</noscript>*

*<div id="root"></div>*

*<!--*

*This HTML file is a template.*

*If you open it directly in the browser, you will see an empty page.*

*You can add webfonts, meta tags, or analytics to this file.*

*The build step will place the bundled scripts into the <body> tag.*

*To begin the development, run `npm start` or `yarn start`.*

*To create a production bundle, use `npm run build` or `yarn build`.*

*-->*

*</body>*

*</html>*

**Cart.css**

*/\* Cart Styles \*/*

*.cart {*

*margin-top: 30px;*

*font-family: 'Roboto', sans-serif;*

*background-color: #ffffff; /\* White background for the Cart \*/*

*padding: 20px;*

*border-radius: 8px;*

*box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);*

*}*

*.cart h2 {*

*color: #333;*

*font-size: 24px;*

*margin-bottom: 15px;*

*}*

*.cart ul {*

*list-style: none;*

*padding: 0;*

*}*

*.cart li {*

*display: flex;*

*justify-content: space-between;*

*margin-bottom: 15px;*

*padding: 15px;*

*border: 1px solid #ddd;*

*border-radius: 8px;*

*background-color: #fafafa; /\* Light gray background for cart items \*/*

*transition: background-color 0.3s ease;*

*}*

*.cart li:hover {*

*background-color: #f0f0f0; /\* Slightly darker background when hovered \*/*

*}*

*.cart .product-name {*

*font-size: 18px;*

*color: #555;*

*flex: 2;*

*}*

*.cart .product-quantity {*

*display: flex;*

*justify-content: center;*

*align-items: center;*

*flex: 1;*

*}*

*.cart .quantity-input {*

*width: 60px;*

*text-align: center;*

*padding: 5px;*

*margin: 0 10px;*

*border: 1px solid #ddd;*

*border-radius: 5px;*

*font-size: 14px;*

*}*

*.cart button {*

*background-color: #dc3545; /\* Red background for remove buttons \*/*

*color: white;*

*padding: 8px 15px;*

*border: none;*

*border-radius: 5px;*

*cursor: pointer;*

*font-size: 14px;*

*transition: background-color 0.3s ease;*

*}*

*.cart button:hover {*

*background-color: #c82333; /\* Darker red on hover \*/*

*}*

*.cart h3 {*

*margin-top: 20px;*

*font-size: 20px;*

*color: #333;*

*font-weight: bold;*

*}*

**Cart.js**

*import React from 'react';*

*import './Cart.css';*

*const Cart = ({ cartItems, onRemoveFromCart, onUpdateQuantity }) => {*

*return (*

*<div className="cart">*

*<h2>Your Cart</h2>*

*<ul>*

*{cartItems.map((item) => (*

*<li key={item.id}>*

*<span className="product-name">{item.name}</span>*

*<span className="product-quantity">*

*<button onClick={() => onUpdateQuantity(item, 'decrease')}>-</button>*

*<input*

*type="number"*

*className="quantity-input"*

*value={item.quantity}*

*onChange={(e) => onUpdateQuantity(item, 'update', e.target.value)}*

*min="1"*

*/>*

*<button onClick={() => onUpdateQuantity(item, 'increase')}>+</button>*

*</span>*

*<span className="remove-button">*

*<button onClick={() => onRemoveFromCart(item)}>Remove</button>*

*</span>*

*</li>*

*))}*

*</ul>*

*<h3>Total: ${cartItems.reduce((total, item) => total + item.price \* item.quantity, 0)}</h3>*

*</div>*

*);*

*};*

*export default Cart;*

**ProductList.css**

*/\* Product List Styles \*/*

*.product-list {*

*margin-top: 20px;*

*font-family: 'Roboto', sans-serif;*

*background-color: #ffffff; /\* White background for the Product List \*/*

*padding: 20px;*

*border-radius: 8px;*

*box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);*

*}*

*.product-list h2 {*

*color: #333;*

*font-size: 24px;*

*margin-bottom: 15px;*

*}*

*.product-list ul {*

*list-style: none;*

*padding: 0;*

*}*

*.product-list li {*

*display: flex;*

*justify-content: space-between;*

*margin-bottom: 15px;*

*padding: 15px;*

*border: 1px solid #ddd;*

*border-radius: 8px;*

*background-color: #fafafa; /\* Light gray background for product items \*/*

*transition: background-color 0.3s ease;*

*}*

*.product-list li:hover {*

*background-color: #f0f0f0; /\* Slightly darker background when hovered \*/*

*}*

*.product-list .product-name {*

*font-size: 18px;*

*color: #555;*

*}*

*.product-list .product-price {*

*font-size: 16px;*

*color: #007bff; /\* Blue price \*/*

*}*

*.product-list button {*

*background-color: #007bff;*

*color: white;*

*padding: 10px 20px;*

*border: none;*

*border-radius: 5px;*

*cursor: pointer;*

*font-size: 14px;*

*transition: background-color 0.3s ease;*

*}*

*.product-list button:hover {*

*background-color: #0056b3; /\* Darker blue on hover \*/*

*}*

**ProductList.js**

*import React from 'react';*

*import './ProductList.css';*

*const ProductList = ({ products, onAddToCart }) => {*

*return (*

*<div className="product-list">*

*<h2>Available Products</h2>*

*<ul>*

*{products.map((product) => (*

*<li key={product.id}>*

*<span className="product-name">{product.name}</span>*

*<span className="product-price">${product.price}</span>*

*<button onClick={() => onAddToCart(product)}>Add to Cart</button>*

*</li>*

*))}*

*</ul>*

*</div>*

*);*

*};*

*export default ProductList;*

**App.css**

*/\* Global Styles \*/*

*@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500&display=swap');*

*/\* Body background color \*/*

*body {*

*font-family: 'Roboto', sans-serif;*

*background-color: #f4f6f9; /\* Light gray background for the whole app \*/*

*margin: 0;*

*padding: 0;*

*box-sizing: border-box;*

*color: #333;*

*}*

*/\* App header \*/*

*h1 {*

*font-size: 28px;*

*text-align: center;*

*color: #343a40;*

*margin-top: 20px;*

*background-color: #ffffff; /\* White background for the header \*/*

*padding: 15px;*

*box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);*

*margin-bottom: 20px;*

*}*

***App.js***

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

*import ProductList from './components/ProductList';*

*import Cart from './components/Cart';*

*const App = () => {*

*// Initial product list*

*const products = [*

*{ id: 1, name: 'Laptop', price: 1000 },*

*{ id: 2, name: 'Phone', price: 500 },*

*{ id: 3, name: 'Headphones', price: 100 }*

*];*

*// Cart state to store added items*

*const [cartItems, setCartItems] = useState([]);*

*// Handle adding products to the cart*

*const handleAddToCart = (product) => {*

*const existingItemIndex = cartItems.findIndex((item) => item.id === product.id);*

*if (existingItemIndex !== -1) {*

*// Update quantity if product already exists in the cart*

*const updatedCart = [...cartItems];*

*updatedCart[existingItemIndex].quantity += 1;*

*setCartItems(updatedCart);*

*} else {*

*// Add new product to cart with quantity 1*

*setCartItems([...cartItems, { ...product, quantity: 1 }]);*

*}*

*};*

*// Handle removing product from the cart*

*const handleRemoveFromCart = (product) => {*

*const updatedCart = cartItems.filter((item) => item.id !== product.id);*

*setCartItems(updatedCart);*

*};*

*// Handle updating the product quantity*

*const handleUpdateQuantity = (product, action, newQuantity = null) => {*

*const updatedCart = [...cartItems];*

*const index = updatedCart.findIndex((item) => item.id === product.id);*

*if (index !== -1) {*

*if (action === 'increase') {*

*updatedCart[index].quantity += 1;*

*} else if (action === 'decrease' && updatedCart[index].quantity > 1) {*

*updatedCart[index].quantity -= 1;*

*} else if (action === 'update' && newQuantity >= 1) {*

*updatedCart[index].quantity = parseInt(newQuantity, 10);*

*}*

*setCartItems(updatedCart);*

*}*

*};*

*return (*

*<div>*

*<h1>Shopping Cart App</h1>*

*<ProductList products={products} onAddToCart={handleAddToCart} />*

*<Cart*

*cartItems={cartItems}*

*onRemoveFromCart={handleRemoveFromCart}*

*onUpdateQuantity={handleUpdateQuantity}*

*/>*

*</div>*

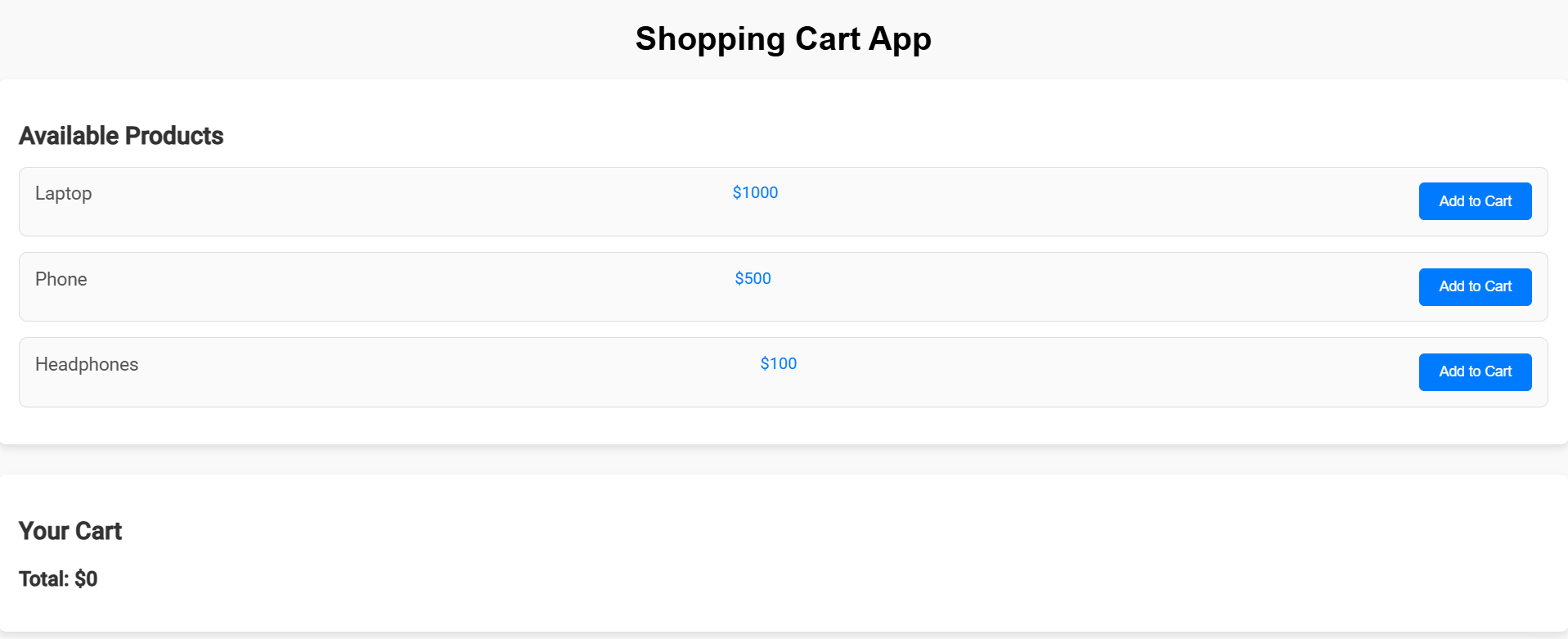
*);*

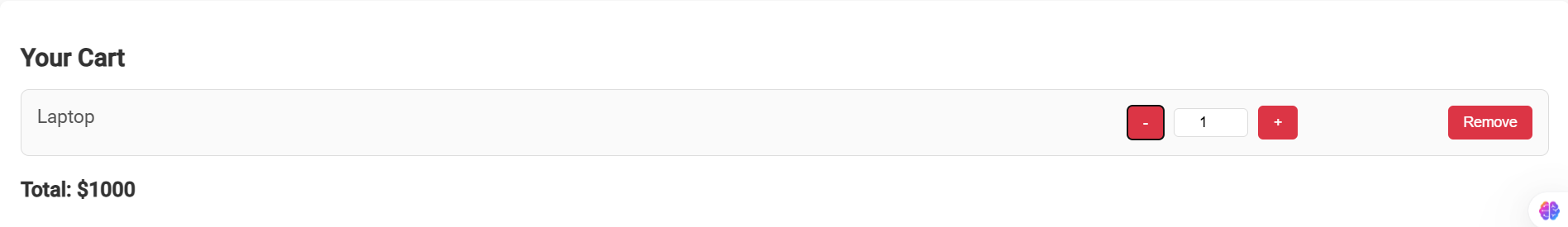
*};*

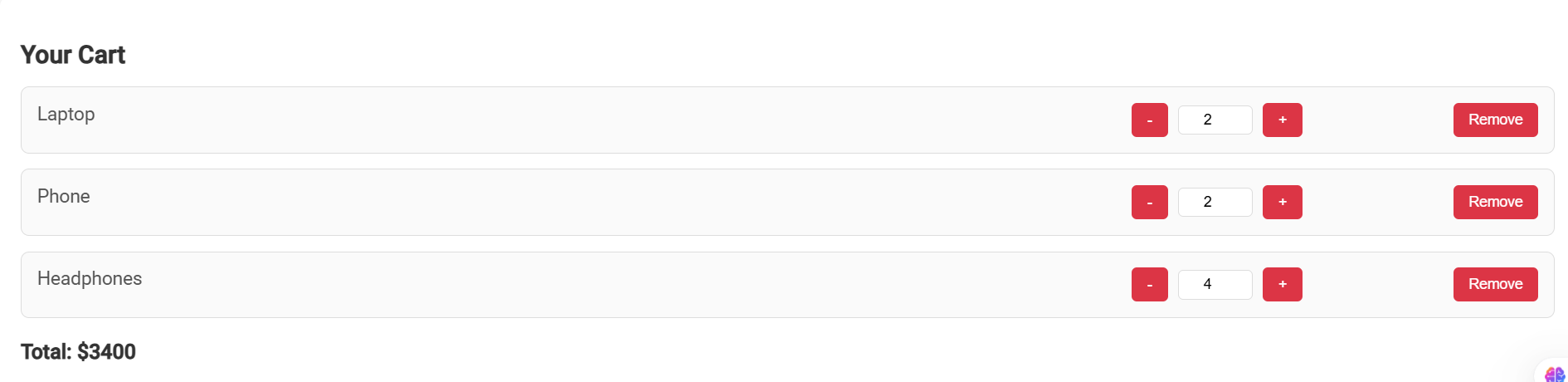
*export default App;*

**Key Concepts**

1. **Props for Data Transfer**:
   * The products list is passed to the ProductList component.
   * The cart state is passed to the Cart component.
2. **Callback Functions for Communication**:
   * onAddToCart, onUpdateQuantity, and onRemoveFromCart are callbacks to modify the parent state.







**How to Run**

1. Create a React app using npx create-react-app shopping-cart.
2. Replace the default files with the above code.
3. Start the app using npm start.

**Conclusion:**

we successfully demonstrated how to communicate between multiple screens (components) using **props** and **callback functions** in React. This pattern is commonly used in React applications to manage state and data flow between components, ensuring a seamless user experience. By understanding these concepts, developers can build more interactive and dynamic applications where data is shared and updated between different parts of the app.