React Lab Exercise — Café Order Builder

Level: Beginner

Duration: 45–75 minutes

Overview

Build a small single-page React exercise that models a simplified café ordering flow. Students will create functional components, pass **props**, and manage UI state with **useState**. The app focuses on component composition, event handling, and basic state updates (add/remove/update quantity).

This is a compact, realistic case study: a customer browses a café menu, adds items to an order, adjusts quantities, and sees the total price.

Learning objectives

By the end of the lab you should be able to:

- Create and export functional React components.
- Pass data and callbacks between components using **props**.
- Use the **useState** hook to store and update component state.
- Map arrays into component lists with `map()` and add `key` props.
- Implement simple UI interactions (add, remove, increment/decrement).
- Calculate derived data (order total) from state.

Prerequisites

- Node.js (14+)
- npm or yarn
- Basic JavaScript (arrays/objects/functions)
- Familiarity with terminal/command line and a code editor (VS Code recommended)

Setup (choose one)

Create React App (simpler)

```
npx create-react-app cafe-order
cd cafe-order
npm start
```

Vite (faster dev server)

```
npm create vite@latest cafe-order -- --template react
cd cafe-order
npm install
npm run dev
```

Open `http://localhost:3000` (CRA) or the dev URL Vite prints.

Project structure suggestion

```
cafe-order/
├─ src/
│  ├─ components/
│  │  ├─ MenuItem.jsx
│  │  ├─ MenuList.jsx
│  │  └─ OrderSummary.jsx
│  ├─ data/
│  │  └─ menu.js
│  ├─ App.jsx
│  └─ index.jsx
└─ package.json
```

Starter data — `src/data/menu.js`

```
export const MENU = [
  { id: 1, name: 'Espresso', desc: 'Strong, small shot', price: 120 },
  { id: 2, name: 'Cappuccino', desc: 'Espresso, steamed milk, foam', price:
180 },
  { id: 3, name: 'Latte', desc: 'Espresso with steamed milk', price: 200 },
  { id: 4, name: 'Cold Brew', desc: 'Slow-brewed cold coffee', price: 150 }
];
```

Step-by-step tasks

Step 1 — Create a MenuItem component (props)

Goal: Practice creating a functional component and using props.

File: `src/components/MenuItem.jsx`

```
import React from 'react';

export default function MenuItem({ item, onAdd }) {
  // `item` is a prop (object). `onAdd` is a callback prop.
  return (
```

```
    <div className="menu-item">
      <h3>{item.name}</h3>
      <p>{item.desc}</p>
      <div>
        <strong>₹{item.price}</strong>
        <button onClick={() => onAdd(item)}>Add</button>
      </div>
    </div>
  );
}
```

What to check: Confirm `MenuItem` renders name, description and that clicking Add calls the `onAdd` callback.

Step 2 — Create MenuList (mapping props to children)

Goal: Map an array of items to multiple `MenuItem` components.

File: `src/components/MenuList.jsx`

```
import React from 'react';
import MenuItem from './MenuItem';

export default function MenuList({ items, onAdd }) {
  return (
    <div className="menu-list">
      <h2>Menu</h2>
      {items.map((it) => (
        <MenuItem key={it.id} item={it} onAdd={onAdd} />
      ))}
    </div>
  );
}
```

What to check: Every menu item appears and Add works for each.

Step 3 — Manage order state in App.jsx (useState)

Goal: Keep the order in top-level state and implement `addToOrder`.

File: `src/App.jsx`

```
import React, { useState } from 'react';
import MenuList from './components/MenuList';
import OrderSummary from './components/OrderSummary';
import { MENU } from './data/menu';

export default function App() {
  const [order, setOrder] = useState([]); // array of {id,name,price,qty}

  function addToOrder(item) {
```

```
    setOrder((prev) => {
      const found = prev.find((p) => p.id === item.id);
      if (found) {
        // increment qty
        return prev.map((p) =>
          p.id === item.id ? { ...p, qty: p.qty + 1 } : p
        );
      }
      // add new item with qty 1
      return [...prev, { ...item, qty: 1 }];
    });
  }

  function removeFromOrder(id) {
    setOrder((prev) => prev.filter((p) => p.id !== id));
  }

  function updateQty(id, qty) {
    if (qty <= 0) return removeFromOrder(id);
    setOrder((prev) => prev.map((p) => (p.id === id ? { ...p, qty } : p)));
  }

  return (
    <div className="app">
      <h1>Café Order Builder</h1>
      <div className="layout">
        <MenuList items={MENU} onAdd={addToOrder} />
        <OrderSummary order={order} onRemove={removeFromOrder}
onUpdateQty={updateQty} />
      </div>
    </div>
  );
}
```

What to check: Add the same item multiple times and see its quantity increase. The `order` state lives in `App` and is passed down.

Step 4 — Create OrderSummary component (derived data)

Goal: Read the `order` prop and calculate a total. Allow quantity changes and removal.

File: `src/components/OrderSummary.jsx`

```
import React from 'react';

export default function OrderSummary({ order, onRemove, onUpdateQty }) {
  const total = order.reduce((sum, it) => sum + it.price * it.qty, 0);

  return (
    <div className="order-summary">
      <h2>Order Summary</h2>
      {order.length === 0 ? (
        <p>Your order is empty.</p>
      ) : (
        <ul>
          {order.map((it) => (
```

```
            <li key={it.id}>
              <span>{it.name} — ₹{it.price} x {it.qty}</span>
              <button onClick={() => onUpdateQty(it.id, it.qty - 1)}>-
</button>
              <button onClick={() => onUpdateQty(it.id, it.qty +
1)}>+</button>
              <button onClick={() => onRemove(it.id)}>Remove</button>
            </li>
          ))}
        </ul>
      )}
      <h3>Total: ₹{total}</h3>
    </div>
  );
}
```

What to check: Changing qty updates the displayed total.

Step 5 — Styling (optional)

Add a few simple styles in `src/index.css` or `App.css`:

```
.layout { display: flex; gap: 24px; }
.menu-list, .order-summary { flex: 1; padding: 12px; border: 1px solid
#ddd; }
.menu-item { margin-bottom: 12px; }
```

Manual tests to perform (smoke tests)

1. Click **Add** on `Espresso` — order should show `Espresso x 1`.
2. Click **Add** on `Espresso` again — quantity should become 2.
3. Click **+** on the order summary — qty increments.
4. Click **-** down to 0 — item should be removed.
5. Add multiple different items — total must sum correctly.

Assessment questions (for grading)

1. What is the difference between props and state? (Answer: props are read-only data passed from parent to child; state is internal and can be updated via hooks like `useState`.)
2. Where is the order state stored in this app and why? (Answer: in `App.jsx` so multiple child components can access/update it — single source of truth.)
3. Explain how `addToOrder` updates an existing item's quantity instead of duplicating the item. (Answer: it searches `prev` state for the item id and either maps to increment qty or appends a new object with `qty:1`.)
4. Why is `key` needed when rendering lists? (Answer: keys let React track items efficiently during re-renders.)

Stretch goals / extensions

- Persist the order to `localStorage` so a page refresh keeps the cart.
- Replace `useState` with `useReducer` and implement actions: ADD, REMOVE, UPDATE_QTY.
- Add a simple filter or category tabs for the menu.
- Fetch menu items from a mock JSON API instead of `menu.js`.
- Add unit tests for critical functions like `addToOrder` using Jest.
