Lesson 8 Exercises - Data Mutation, Types of Requests

8a. On the orders page, make the “Add to Cart” button for each product work. Use a quantity of 1 and remember to reload the cart after.

8b. In CheckoutPage.jsx, separate the code that reloads the Payment Summary into another useEffect (this should use the dependency array `[]`. Test that everything still works.

8c. My backend has an api called `POST /api/reset`. This resets the data to some default values. To use it, add `window.axios = axios;` in your code (this makes `axios` available in the Console). Save, and in the Console, try running `axios.post('/api/reset')`. Refresh the page and check.

Solutions in description

8d. On the home page, after adding a product to the cart, a message should appear saying "✓ Added"

- The HTML element for this message already exists. It’s `<div classname="added-to-cart">`
- However, this element has the style `opacity: 0;` (meaning it’s completely see-through and invisible).
- Create a state, which tracks whether to show this “Added” message (set the initial value to `false`).
- After adding to cart, set this state to `true`, set the opacity of the element to 1 (not see-through) using the prop: `style={{ opacity: yourState ? 1 : 0 }}`
- Using `setTimeout`, after 2 seconds, set the state back to `false` to make the message disappear.

Challenge Exercises

In the checkout page, we'll make the update quantity feature work.

8e. First, go to the code for each cart item (inside OrderSummary.jsx or CartItemDetails.jsx) and find: `<span className="quantity-label">`

- Before this element, add a textbox: `<input type="text" />`
- Add a className to this textbox. Style it with CSS and set `width: 50px;`

8f. Add a state to track if the quantity is being updated (initial value `false`). When clicking "Update" switch this state between `true` and `false`. If the state is `true` show the textbox, otherwise show the quantity label. Note: you'll need to move the code into a CartItemDetails component (if you haven't done so) in order for each cart item to have its own state.

8g. We'll create a controlled input for the quantity textbox.

- Create a state for the quantity in the textbox: `quantity, setQuantity` and set the initial value to `cartItem.quantity`
- Add a prop `value={quantity}` to the input element.
- Add a prop `onChange={}` to the input element and give it a function.
- The function gets a parameter called `event`. Get the text in the textbox using `event.target.value` and save it in the state using `setQuantity`

8h. If the isUpdating state is `true` and we click the "Update" link, send a request `PUT /api/cart-items/:productId` and send the backend an object with the property `quantity` (the value is the `quantity` saved in the previous exercise - remember to convert it to a number!). `await` the request, reload the cart, and set the isUpdating state to `false`.

8i. We'll add keyboard events to the quantity input:

- Add an `onKeyDown` prop to the input and give it a function. The function gets a parameter `event` and `event.key` has the key that was pressed.
- If the key pressed is `'Enter'`, update the quantity in the backend (run the same function as when the "Update" link is pressed).
- If the key pressed is `'Escape'`, cancel the update (set the `quantity` back to `cartItem.quantity`, and set the isUpdating state to `false`).

We'll make the search bar at the top of the home page work.

8j. Open the home page, and in the Header component, find the search bar `<input className="search-bar">`

- Create a controlled input for the search bar. Add an `onClick` prop to the search button. When clicked, console.log the search text (for now).
- Check the search bar also works on the orders page and tracking page.

8k. Now when we click the search button, we'll navigate to the home page in order to show the search results. Using the `useNavigate` hook from react-router, navigate to the URL `/?search=${search}`

- Navigating to `/` will navigate to the home page. `?search=${search}` saves the search text in the URL so we can share it between pages.

8l. In the home page, `import { useSearchParams } from 'react-router';`

- Get the search text using `const [searchParams] = useSearchParams();` and `const search = searchParams.get('search');`
- When loading the products, if `search` exists, use this URL Path instead: `/api/products?search=${search}` (also add `search` to the dependency array. Values from outside of useEffect should be in dependency array).
- In the Header, also get the search text out of the url. If it exists, set it as the initial value of the search state. Try it out to see if search bar works!