Name	Name	Last commit date
README.md	Create README.md	4 months ago

README.md

Lesson 7 Exercises - Backend, Data Fetching, Async Await

7a. In OrdersPage.jsx, use async await to load the data instead of promise.

7b. In OrderSummary.jsx, separate the cart item details into a component:

- Create a new component named CartItemDetails.
- Move `<img className="product-image">` and `<div className="cart-item-details">` into this new component.
- Hint: you'll need to use a fragment `<></>` since you'll be returning multiple elements from the component.

7c. In OrderSummary.jsx, separate the `delivery-date` into a component:

- Create a new component named DeliveryDate. Move the code that finds the selectedDeliveryOption into the component as well.

Solutions in description

7d. We saw useEffect runs twice due to `<StrictMode>`. In main.jsx, temporarily remove `<StrictMode>` and check that useEffect runs once.

- Running twice helps us catch bugs. (Running useEffect twice should result in the same HTML being rendered. This is called idempotency).
- This only happens in development. In production (the website is on the Internet), `<StrictMode>` doesn't do anything. Add `<StrictMode>` back.

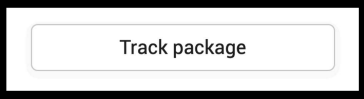
7e. In OrdersPage.jsx, separate the `<div className="orders-grid">` into its own component, named OrdersGrid.

7f. In OrdersGrid, separate the `<div className="order-header">` and `<div className="order-details-grid">` into their own components.

7g. In CheckoutPage, separate the header into a component (did this in lesson 6 exercises). Pass in the cart as a prop, display the total quantity.

Challenge Exercises

We'll display the tracking page using data from the backend. First, make sure you created the TrackingPage component and added it to the Router (see exercise 6h solution).



7h. Open the orders page in the browser and click a "Track package" button. The tracking page will not work because there's a <Header /> in the page and it needs the cart. Pass the cart into <Header /> using a prop. Check the page works.

7i. Find the "Track package" button in the code. It links to `"/tracking"`. Update this so it links to `` `/tracking/${orderId}/${productId}` `` (insert the order and product ids into the string). Click "Track package", and notice the order id and product id are now in the URL.

7j. In order for the URL `/tracking/${orderId}/${productId}` to work, we also need to update the Router. In App.jsx, update the Router so the tracking page uses `path="tracking/:orderId/:productId"`

- `:orderId` and `:productId` are called URL params (parameters). We can replace them with any text, and this allows us to save an order id and a product id directly in the URL.
- To get these values out of the URL, in the TrackingPage, use the hook: `import { useParams } from 'react-router';` then at the top of the TrackingPage, run `const params = useParams();`
- Try `console.log(params);` and open a tracking page. Notice `params` is an object that contains the orderId and productId from the URL.
- Destructure `const params` into `const { orderId, productId }`

Solutions in description

7k. Now, we'll load the data for the Tracking Page from the backend. Using `axios`, `useEffect`, and the `orderId` from the URL:

- Make a request to `` `/api/orders/${orderId}?expand=products` `` this will load the order from the backend (with product details attached).
- Instead of the dependency array `[]`, use `[orderId]`. This will re-run useEffect if `orderId` changes (reload the order if `orderId` changes).
- Save the order in useState with initial value of null. Before returning the HTML, check if the order is loaded using `if (!order) { return null; }`
- After the order is loaded, replace the text in the HTML using the values in the order (skip the progress bar for now). Hint: get product details and delivery time using the `productId` and `order.products.find()`
- Go to the orders page, click "Track package" for some other products / orders, and check that the details are correct.

7l. We'll calculate the progress (how close the product is to delivery).

- First: `orderProduct.estimatedDeliveryTimeMs - order.orderTimeMs` gets the total time required for delivery (`totalDeliveryTimeMs`).
- Then `const timePassedMs = dayjs().valueOf() - order.orderTimeMs;` calculates the amount of time that has passed since creating the order.
- Calculate `(timePassedMs / totalDeliveryTimeMs) * 100` this gives us the delivery progress as a percent (50% = halfway, 100% = delivered).
- If the progress is > 100%, set it to 100% (limit it to at most 100%).
- Scroll down to the `<div className="progress-bar">` and give it a prop `style={{ width: `${deliveryPercent}%` }}`. Refresh, and check that the progress bar now reflects the delivery percent.
- Temporarily update `timePassedMs` to `totalDeliveryTimeMs * 0.3` and refresh. Check that the progress bar is updated. Change it back after.

7m. If the percent is >= 100, instead of “Arriving on” display “Delivered on”.

7n. We’ll highlight the correct label based on the delivery percent:

- Check if the delivery percent is < 33. Save it in a variable `isPreparing`
- Check if the percent is >= 33 and < 100. Save it in a variable `isShipped`
- Check if the percent is === 100. Save it in a variable `isDelivered`
- Scroll down to the `progress-label` elements. If an element has the class `current-status` that means it will be highlighted in green.
- For each `progress-label`, change `className` to a template string, and only add the class `current-status` if it is the current status. Example: `` `progress-label ${isPreparing && 'current-status'}` ``

7o. In 6k, we added a 404 page. Pass the cart into the Header in this page.