useContext

In this lab, let's exercise React's Context API. It appears we'll need the user state variable throughout the app. Currently it is at the top in App but is needed in a few components lower in the tree. Our app is currently passing user down using props drilling. We'll clean that up in this lab.[1]

Understanding the context

1. Edit App.tsx. Look for where we're establishing the user state variable:

```
const [user, setUser] = useState<User>();
```

2. Now look in the JSX. See how we're passing that user down to <Cart /> and to <Order /> and to <Orders />? If you look in Orders.tsx, you'll see that we're further passing it down to <OrderSummary />. This is prop drilling.

If we can make user available to <OrderSummary /> directly it won't be needed in <Orders /> at all.

Writing to context

3. At the top of App.jsx, import createContext.

4. Create the context and provide a default value of undefined.

```
export const UserContext = createContext<User|undefined>(undefined);
```

5. Wrap App in <UserContext>:

```
<UserContext.Provider value={user}>
```

6. Run and test. You should see no difference but no errors either.

Removing the prop drilling

7. Remove the user prop from <Cart />. Obviously you'll need to remove it from the <Cart /> tag in App.tsx and also from the function declaration in Cart.tsx.

8. If you run and test now, you'll get an error because Cart no longer has a user. Let's add it back in.

Reading the context in Cart

9. In Cart.tsx, import useContext from react.

10. Also import UserContext from App.tsx.

---

[1] In reality, if you're passing props down a level or two prop drilling isn't awful. More than that and useContext or a state container is a better choice. We'll implement it in this situation for hands-on practice experience.

11. Near the top of the Cart component, call useContext and save its result in the user variable. As soon as you type this, all linting errors should disappear.

12. Not only so, but if you run and test, you should be 100% working again, even without passing the user in. Go ahead and add some things to the cart, then log in. When you visit Cart/checkout, your logged-in data is recognized.

Cool, right?

13. Note: you may get a warning about fast refresh not working. If you want to solve it, move the context creation into a separate JavaScript file and import it into both App.tsx and Cart.tsx. This is optional and only affects performance during development time.

Reading context in Order and NavBar

User is also being passed from App to Order and to NavBar through props. Let's change that to context just like you did in Cart above.

14. Remove the user from props, both passing down from App.tsx and also receiving props in Order.tsx and NavBar.

15. Read user from context in Order.tsx and in NavBar.

16. Run and test. Make sure your refactor hasn't broken anything.

Reading the context in OrderSummary

Let's do one more, and this is the one where we'll see the most benefit. User is prop-drilled from App to Orders and then to OrderSummary. If we useContext to read user in OrderSummary directly, the user object can completely bypass Orders.

17. Remove the user prop in <Order /> and <OrderSummary />. See how clean that makes them?

Unfortunately it also breaks until we add the user back in through context.

18. Add context to OrderSummary.tsx just like you did with Cart above.

19. Run and test. If your refactor worked, your list of past orders will be unchanged.