useReducer

The Cart component is using useState to manage the data. But we're missing some opportunities for enhanced user experience. There's no validation for credit card number, CVV number, or expiration dates. We could make all that work with useState, but let's see what it would be like to do all of that using a reducer.

Setting up for useReducer

1. Edit Cart.tsx. Create the reducer blank:

```
// We *could* add tip, area, location, etc. if desired
type State = {
  cvv: number | undefined;
  expiryMonth: number | undefined;
  expiryYear: number | undefined;
  pan: string;
  panIsValid: boolean;
}
type Action = {
  type: string;
  payload?: any;
}
// Reducer function merely returns a *copy* of the state object so far
const reducer = (state: State, action: Action = { type: "" }): State => {
  console.log({ state, action }) // Debugging only - Remove eventually
  switch (action.type) {
    default:
      return { ...state }
  }
}
```

2. Set the initial state. Note that since it depends on the user existing, it will have to be inside the component, after the user is read from context.

```
const initState = {
  cvv: undefined,
  expiryMonth: user?.creditCard?.expiryMonth,
  expiryYear: user?.creditCard?.expiryYear,
  pan: user?.creditCard?.pan ?? "",
  panIsValid: false,
};
```

3. Import useReducer from React.

4. Inside the Cart function, call useReducer:

```
const [state, dispatch] = useReducer<Reducer<State, Action>>(reducer, initState);
```

This gives us a state object that we can use for display purposes and a dispatch function that we'll use to dispatch actions to the reducer itself.

5. Run and test. You should see no difference yet, but you should also see no errors.

Implementing the CVV reducer case

6. Find the <input> where we're setting the CVV. Change it to this:

```
<input value={state.cvv}
       onChange={e => dispatch({type:"SET_CVV", payload:{cvv:+e.target.value}})}
       type="number"
       id="cvv"
       required />
```

7. In the reducer, add an action for "SET_CVV":

```
case "SET_CVV":
  return { ...state, cvv: action.payload.cvv };
```

8. Run and test. You should still be able to change the CVV in the form.

9. Note that placeOrder is still using the variable cvv instead of state.cvv. You'll need to change that. After doing so, you should be able to place an order and it still works.

10. Now that you've refactored all instances of cvv to state.cvv, you can delete the useState for cvv and setCvv.

One down, three to go!

Setting the expiry date

11. Create an action for "SET_EXPIRY_MONTH" and another for "SET_EXPIRY_YEAR".

12. Change both <input>s to handle dispatching an action instead of calling the setter.

13. In those same inputs remember to use state.expiryYear instead of expiryYear and state.expiryMonth instead of expiryMonth.

14. Change the placeOrder function to use the new values also.

15. Delete the useState lines for both of them.

16. Make sure they work as expected before going on.

Let's pretend that our UX people said we're supposed to provide a default value for month or year if one is entered and the other is blank.

17. Change the SET_MONTH case to say that if state.year is undefined, set it to the current year.

18. Run and test. Enter a month, leaving the year blank. If the current year is filled in, you can move on.

19. Change the SET_YEAR case to say that if state.month is undefined, set it to the current month.

20. Run and test the year just like you did with month.

Checking the credit card number

Your credit card number is called the "primary account number" or "PAN".

21. Write a reducer action for "SET_PAN" and change the <input> to dispatch instead of setPan.

22. Change all instances of "pan" to "state.pan". Delete the useState for it.

23. Run and test to make sure you can still put any number in that <input> box.

Did you know that all credit cards have a checksum built in to the primary account number? Luhn's algorithm can warn you if the card number has been typed in wrong. Here's a TypeScript function that implements the Luhn's algorithm.

```
function validateLuhn(pan: string): boolean {
  // Remove spaces and dashes
  pan = pan.replace(/\s+/g, '').replace(/-/g, '');
  if (pan.length < 13 || pan.length > 19) {
    return false;
  }
  let sum = 0;
  let isDoubled = false;
  for (let i = pan.length - 1; i >= 0; i--) {
    let digit = +pan[i];
    if (isDoubled) {
      digit *= 2;
      if (digit > 9)
        digit -= 9;
    }
    sum += digit;
    isDoubled = !isDoubled;
  }
  console.log(sum, sum % 10 === 0)
  return sum % 10 === 0;
}
```

Let's use it to give feedback to the user that they mistyped their number.

24. First, create a ref.

```
const panRef = useRef<HTMLInputElement>(null);
```

25. Next add the ref to the <input>.

```
<input ref={panRef} value={state.pan} ... etc. etc.  />
```

Now you're ready to implement Luhn's algorithm check in the SET_PAN action.

26. In the SET_PAN action, run the function. If it says that the PAN is invalid, set state.panIsValid to true. Otherwise set it to false.

27. Bonus!! If you temporarily interpolate state.panIsValid on the component somewhere you can watch it turn from false to true. Do something like this:

```
<h2>{state.panIsValid ? 'valid' : 'invalid'}</h2>
```

Let's make the credit card number box turn red for an invalid card and green for a valid one.

28. Add this to your component:

```
panRef.current && panRef.current.setCustomValidity(state.panIsValid ? '' : 'Bad pan');
```

29. Run and test. A bad number should turn it red and a good number should turn it green.

Cool, right?

Bonus! Do the other fields

You can consider yourself finished but if you just have extra time and want a challenge, here are a couple more assignments.

30. Make the CVV turn red if it is anything other than a three digit number.

31. Make the Expiry month/year turn red if the month/year combination is in the past. In other words, if the card is expired.