Redux 1.4_CW Exercises

javascript functions for purity

ex01: create a cart reducer

challenge

Create a `cartReducer` that handles adding, removing, updating quantity, and calculating the total price of cart items.

1. Create a `cartReducer` function that takes two parameters: `state` (initial state) and `action`.
2. Implement the switch case for each action type (`ADD_TO_CART`, `REMOVE_FROM_CART`, `UPDATE_QUANTITY`, `CALCULATE_TOTAL`).
    1. `ADD_TO_CART` - cart/added

        Make sure you handle updating of the quantity of an item if it already exists in the cart.

    2. `REMOVE_FROM_CART` - cart/removed
    3. `UPDATE_QUANTITY` - cart/updatedQuantity
    4. `CALCULATE_TOTAL` - cart/calculateTotal
3. For `ADD_TO_CART`, add the product to the `cartItems` array with the help of spread operator.
4. For `REMOVE_FROM_CART`, remove the product from the `cartItems` array based on its ID using array.filter.
5. For `UPDATE_QUANTITY`, update the quantity of the product based on its ID using Object.assign.
6. For `CALCULATE_TOTAL`, calculate the total price of all items in the cart and update the `total` field using array.reduce.

solution:

```
const initialState = {
  cartItems: [],
  total: 0,
}

const cartReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'cart/added':
      const existingCartItem = state.cartItems.find(
        (item) => item.id === action.payload.id,
      )

      if (existingCartItem) {
        const updatedCartItems = state.cartItems.map((item) => {
          if (item.id === action.payload.id) {
            return {
              ...item,
              quantity: item.quantity + 1,
            }
          }
          return item
        })

        return {
          ...state,
          cartItems: updatedCartItems,
        }
      } else {
        return {
          ...state,
          cartItems: [...state.cartItems, { ...action.payload, quantity: 1 }],
        }
      }
    case 'cart/removed':
      return {
        ...state,
        cartItems: state.cartItems.filter((item) => item.id !== action.payload),
      }
    case 'cart/updatedQuantity':
      const updatedCartItems = state.cartItems.map((item) => {
        if (item.id === action.payload.productId) {
          return Object.assign({}, item, { quantity: action.payload.quantity })
        }
        return item
      })
      return {
        ...state,
        cartItems: updatedCartItems,
      }
    case 'cart/calculateTotal':
      const totalPrice = state.cartItems.reduce(
        (total, item) => total + item.price * item.quantity,
        0,
      )
      return {
        ...state,
        total: totalPrice,
      }
    default:
      return state
```

```
  }
}

export default cartReducer
```

COPY

ex02: create cart action creators

challenge

Create action creators for each type of action in the cart reducer.

1. Create constant actions for `ADD_TO_CART`, `REMOVE_FROM_CART`, `UPDATE_QUANTITY`, `CALCULATE_TOTAL`
2. Create an action creator function named `addToCart` that takes a `product` parameter and returns an action object with the `type` of `ADD_TO_CART` and the `payload` as `product`.
3. Create an action creator function named `removeFromCart` that takes a `productId` parameter and returns an action object with the `type` of `REMOVE_FROM_CART` and the `payload` as `productId`.
4. Create an action creator function named `updateQuantity` that takes a `productId` and `quantity` parameter and returns an action object with the `type` of `UPDATE_QUANTITY` and the `payload` as an object containing `productId` and `quantity`.
5. Create an action creator function named `calculateTotal` that returns an action object with the `type` of `CALCULATE_TOTAL`.
6. Make sure you update the `cartReducer` cases with action constants.

solution

```
export const ADD_TO_CART = 'cart/added'
export const REMOVE_FROM_CART = 'cart/removed'
export const UPDATE_QUANTITY = 'cart/updatedQuantity'
export const CALCULATE_TOTAL = 'cart/calculateTotal'

export const addToCart = (product) => ({
  type: ADD_TO_CART,
  payload: product,
})

export const removeFromCart = (productId) => ({
  type: REMOVE_FROM_CART,
  payload: productId,
})

export const updateQuantity = (productId, quantity) => ({
  type: UPDATE_QUANTITY,
  payload: { productId, quantity },
})

export const calculateTotal = () => ({
  type: CALCULATE_TOTAL,
```

```
})
```

COPY

ex03: create and subscribe to the cart store

challenge

Create and subscribe to the cart store to listen when the state changes.

1. Import the necessary action creators and the cart reducer.
2. Create a Redux store using `createStore` and passing the cart reducer.
3. Subscribe to the store using the `store.subscribe` method.

solution

```
import { createStore } from 'redux'
import cartReducer from './cartReducer'

const store = createStore(cartReducer)

store.subscribe(() => {
  console.log(store.getState())
})
```

COPY

ex04: render **product list**

challenge

Implement the `renderProducts` function to display the list of products.

```
const products = [
  { id: 1, name: 'Product A', price: 10 },
  { id: 2, name: 'Product B', price: 20 },
  { id: 3, name: 'Product C', price: 15 },
]
```

COPY

1. Inside the `renderProducts` function, select the `productList` element using `document.getElementById`.
2. For each product, create a list item (`<li>`) containing the product name and price.
3. Call the `renderProducts` function in `**index.js**`.

solution

```
const products = [
  { id: 1, name: 'Product A', price: 10 },
```

```
  { id: 2, name: 'Product B', price: 20 },
  { id: 3, name: 'Product C', price: 15 },
]

const renderProducts = () => {
  const productList = document.getElementById('product-list')
  productList.innerHTML = products
    .map((product) => {
      return `
        <li>
          ${product.name} - Rs.${product.price}
        </li>`
    })
    .join('')
}

renderProducts()
```

COPY

ex05: render cart

challenge

1. Create a function named `updateCart` that will be responsible for rendering the cart items and updating the total cart price. Inside the `updateCart` function, get the current state from the Redux store using `store.getState()`.
    1. Select the `cartList` & `cartTotal` elements using `document.getElementById("cart-total")`.
    2. For each cart item, create a list item (`<li>`) containing the item's name, price, and quantity.

       Join the array of HTML strings using `.join("")` and set it as the `innerHTML` of the `cartList` element.
    3. Update the `cartTotal` element's content with the total price from the state.
2. Call the `**updateCart()**` in the `**index.js**` & in the `**store.subcribe**`.

solution

```
store.subscribe(() => {
  console.log(store.getState())
  updateCart()
})

const updateCart = () => {
  const state = store.getState()
```

```
  const cartList = document.getElementById('cart-list')
  const cartTotal = document.getElementById('cart-total')

  cartList.innerHTML = state.cartItems
    .map((item) => {
      return `
        <li>
          ${item.name} - Rs.${item.price} - Quantity: ${item.quantity}
        </li>`
    })
    .join('')

  cartTotal.innerHTML = `Total: Rs.${state.total}`
}

updateCart()
```

COPY

ex06: play with dispatch actions

challenge

Now, let's play around with dispatching actions and see how the cart state updates.

1. Dispatch the `addToCart` action to add one unit of "Product A" (id: 1) to the cart.
2. Dispatch the `calculateTotal` action after adding the first item.
3. Dispatch the `addToCart` action to add one unit of "Product B" (id: 2) to the cart.
4. Dispatch the `calculateTotal` action after adding the second item.
5. Dispatch the `addToCart` action again to add one more unit of "Product A" (id: 1) to the cart.
6. Dispatch the `calculateTotal` action after adding the third item.
7. Dispatch the `removeFromCart` action to remove "Product A" (id: 1) from the cart.
8. Dispatch the `calculateTotal` action after removing an item.
9. Dispatch the `updateQuantity` action to change the quantity of "Product B" (id: 2) to 5.
10. Dispatch the `calculateTotal` action after updating the quantity.

solution

```
store.dispatch(addToCart({ id: 1, name: 'Product A', price: 10, quantity: 1 }))
store.dispatch(calculateTotal())
store.dispatch(addToCart({ id: 2, name: 'Product B', price: 20, quantity: 1 }))
store.dispatch(calculateTotal())
store.dispatch(addToCart({ id: 1, name: 'Product A', price: 10, quantity: 1 }))
store.dispatch(calculateTotal())
store.dispatch(removeFromCart(1))
store.dispatch(calculateTotal())
store.dispatch(updateQuantity(2, Number(5)))
store.dispatch(calculateTotal())
```

COPY

ex07: play with dispatch actions using redux devtool

1. Dispatch the `addToCart` action to add one unit of "Product A" (id: 1) to the cart.

```
{
  type: 'cart/added',
  payload: { id: 1, name: "Product A", price: 10, quantity: 1 }
}
```

COPY

- **Shopping Cart App**

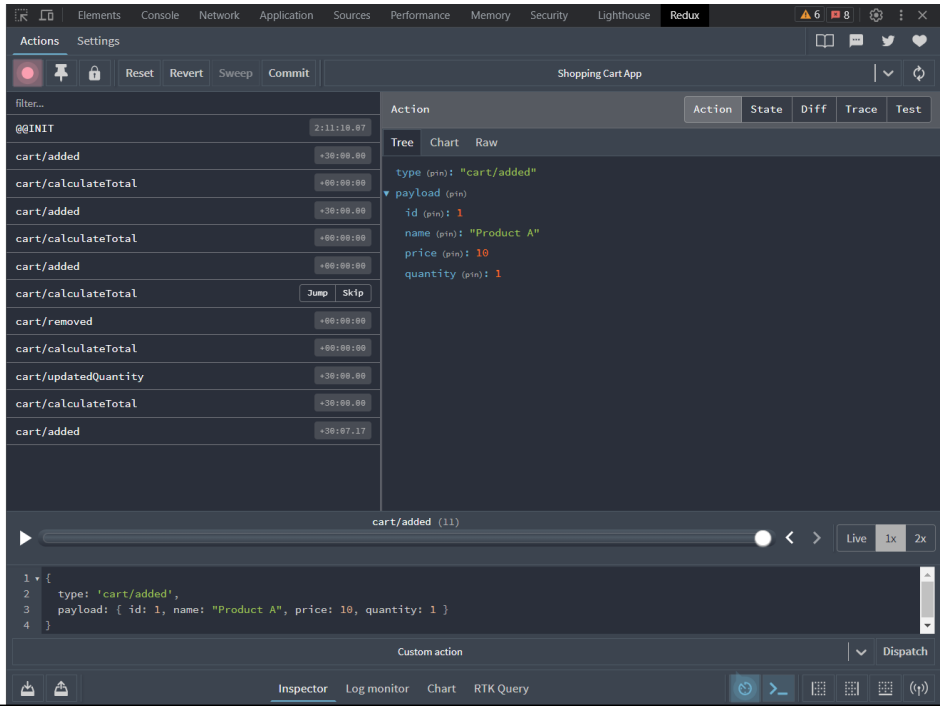
  Products

  - Product A - Rs.10
  - Product B - Rs.20
  - Product C - Rs.15

  Cart

  - Product B - Rs.20 - Quantity: 5
  - Product A - Rs.10 - Quantity: 1

  Total: Rs.100



1. Dispatch the `calculateTotal` action after adding the first item.

```
{
  type: 'cart/calculateTotal'
}
```

COPY

- **Shopping Cart App**

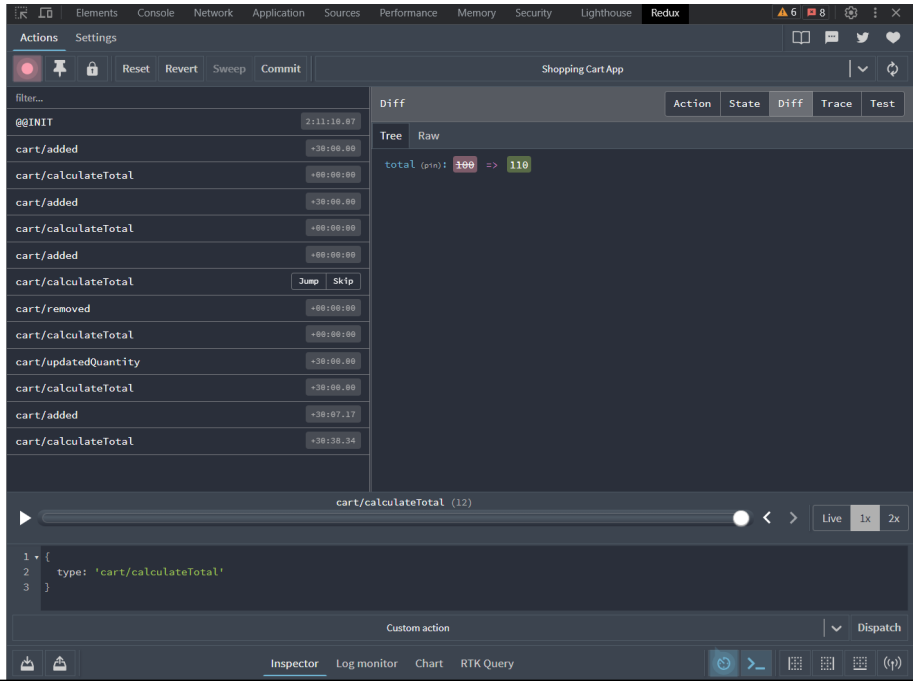
  Products

  - Product A - Rs.10
  - Product B - Rs.20
  - Product C - Rs.15

  Cart

  - Product B - Rs.20 - Quantity: 5
  - Product A - Rs.10 - Quantity: 1

  Total: Rs.110



2. Dispatch the `addToCart` action to add one unit of "Product C" (id: 3) to the cart.

```
{
  type: 'cart/added',
  payload: { id: 3, name: "Product C", price: 15, quantity: 1 }
}
```

COPY

- **Shopping Cart App**

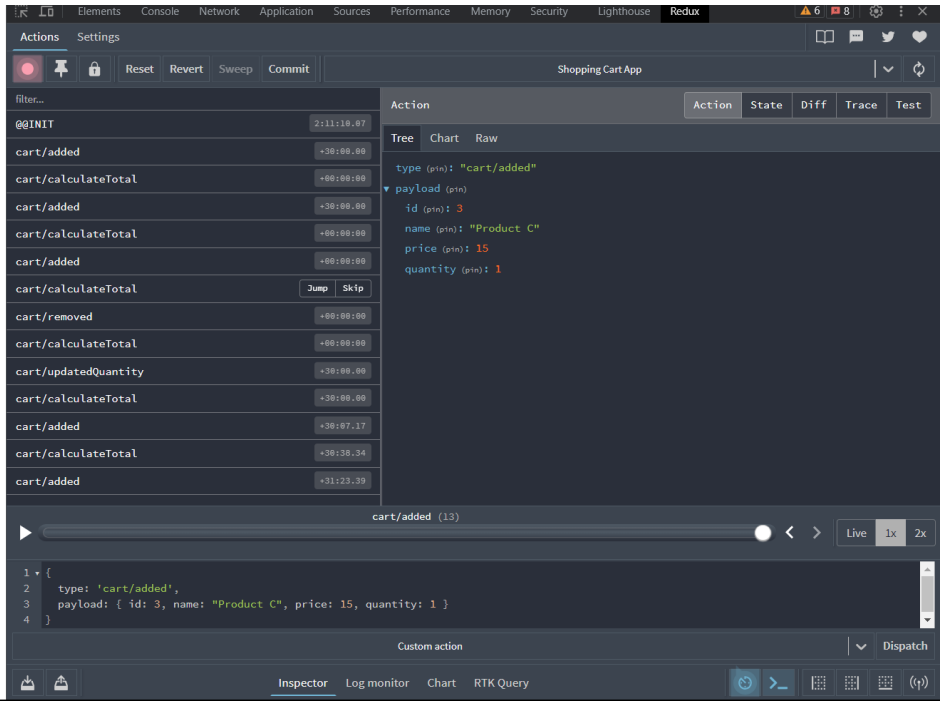
  Products

  - Product A - Rs.10
  - Product B - Rs.20
  - Product C - Rs.15

  Cart

  - Product B - Rs.20 - Quantity: 5
  - Product A - Rs.10 - Quantity: 1
  - Product C - Rs.15 - Quantity: 1

  Total: Rs.110



1. Dispatch the `calculateTotal` action after adding the third item.

```
{
  type: 'cart/calculateTotal'
}
```

COPY

- **Shopping Cart App**

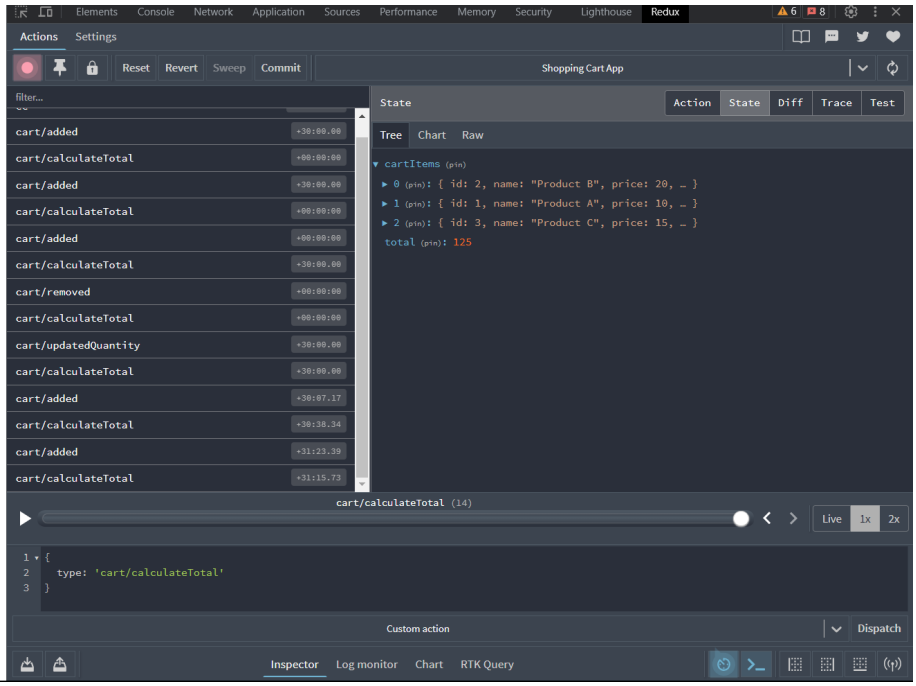
  Products

  - Product A - Rs.10
  - Product B - Rs.20
  - Product C - Rs.15

  Cart

  - Product B - Rs.20 - Quantity: 5
  - Product A - Rs.10 - Quantity: 1
  - Product C - Rs.15 - Quantity: 1

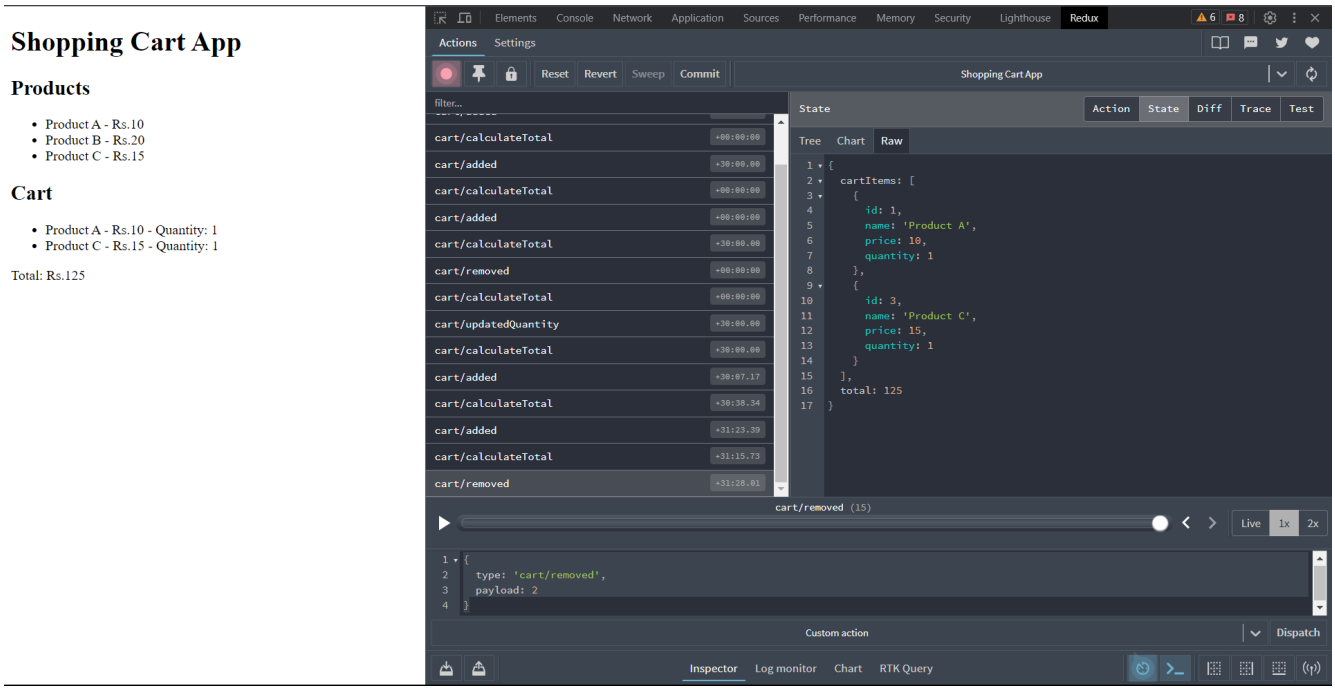
  Total: Rs.125



2. Dispatch the `removeFromCart` action to remove "Product B" (id: 2) from the cart.

```
{
  type: 'cart/removed',
  payload: 2
}
```

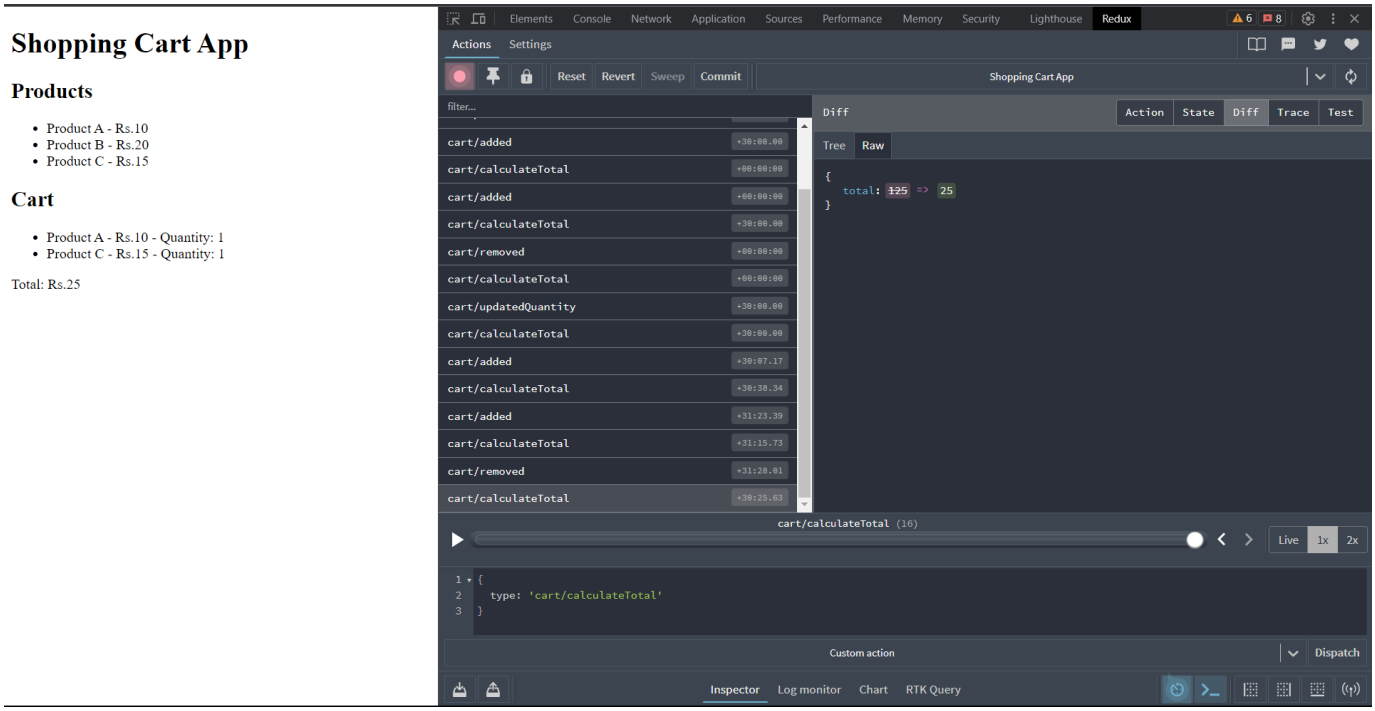
COPY



3. Dispatch the `calculateTotal` action after removing an item.

```
{
  type: 'cart/calculateTotal'
}
```

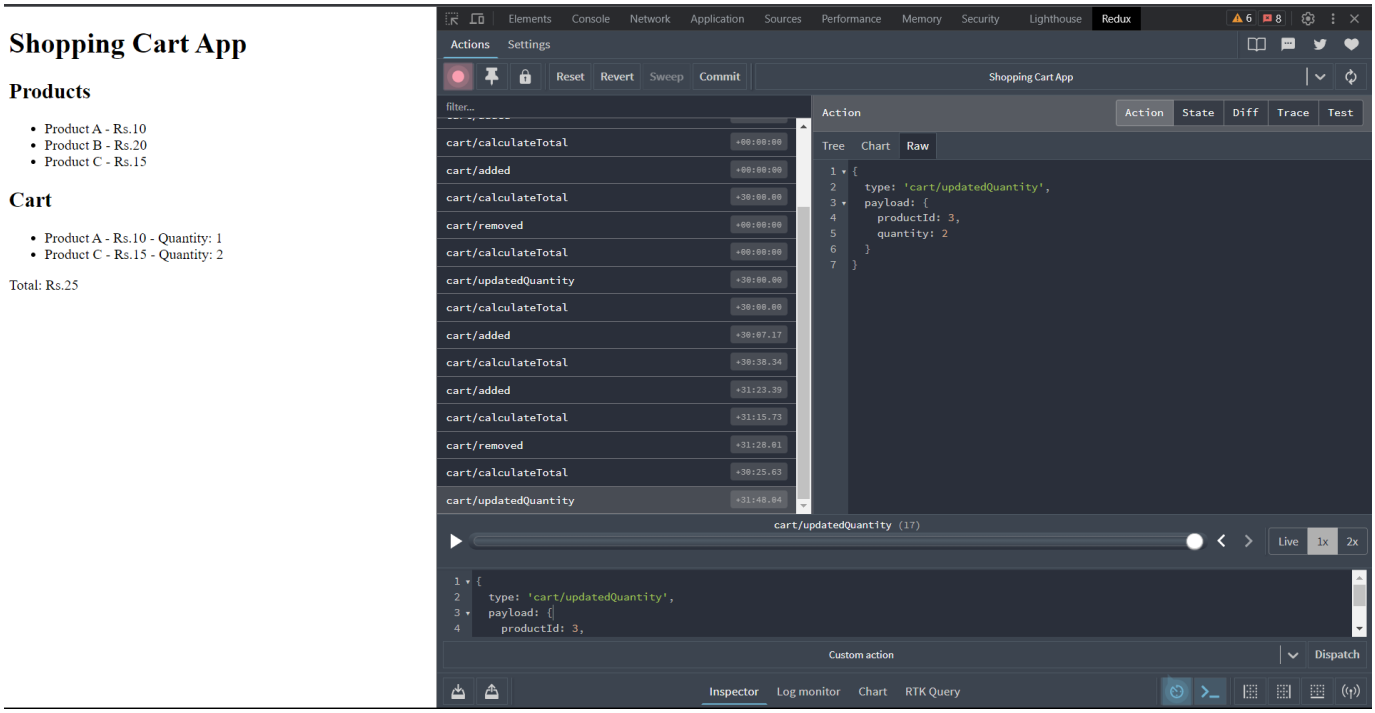
COPY



4. Dispatch the `updateQuantity` action to change the quantity of "Product C" (id: 3) to 2.

```
{
  type: 'cart/updatedQuantity',
  payload: {
    productId: 3,
    quantity: 2
  }
}
```

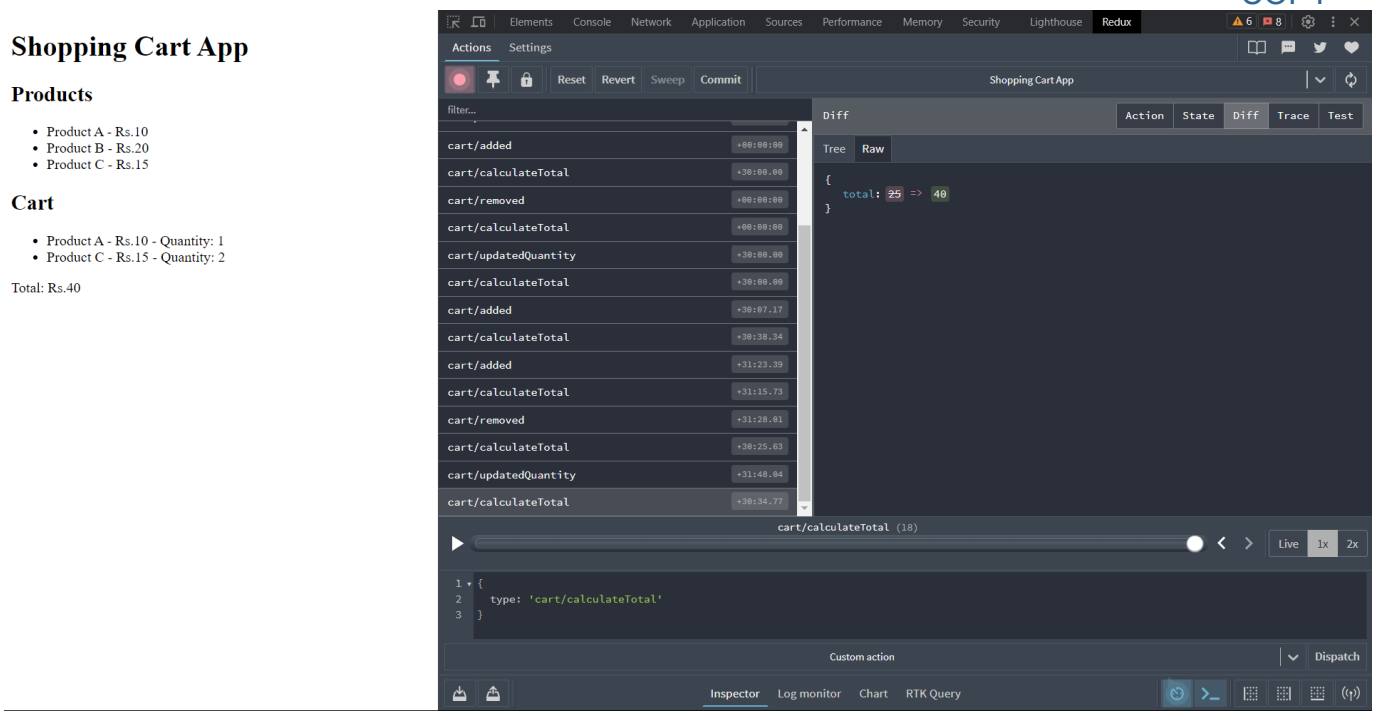
COPY



5. Dispatch the `calculateTotal` action after updating the quantity.

```
{
  type: 'cart/calculateTotal'
}
```

COPY



entire solution w/ UI interactions

https://codesandbox.io/s/rx1-3-cw-entire-solution-w-interaction-9zn3vl

entire solution w/o UI interactions

https://codesandbox.io/s/rx1-3-cw-entire-solution-w-o-ui-interactions-dm3xwc