Лабораторная работа: Добавление состояния в компоненты корзины

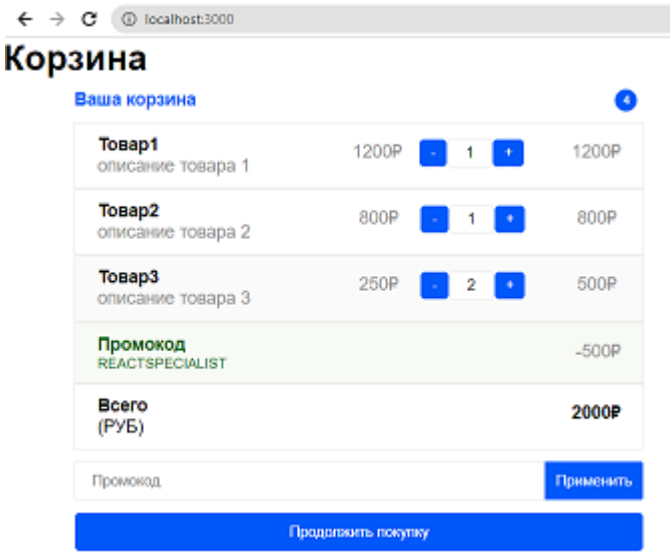
В этой лабораторной мы обновим часть компонентов корзины - добавим работу с состоянием. Воспользуйтесь раздаточным материалом, чтобы не создавать все компоненты заново

Упражнение 1: Задание состояния компоненту счётчику `Counter.js`

1.0. Верните в прежний вид компонент **src/components/App.js**

```
...
return (
    <div className="App">
      <header className="App-header">
        <h1>Корзина</h1>

        <Basket items={items} />
      </header>
    </div>
  );
...
```

Страница в начале работы выглядит так



1.1. Добавьте компоненту **src\components\Counter.js** состояние - количество добавленных в корзину товаров.

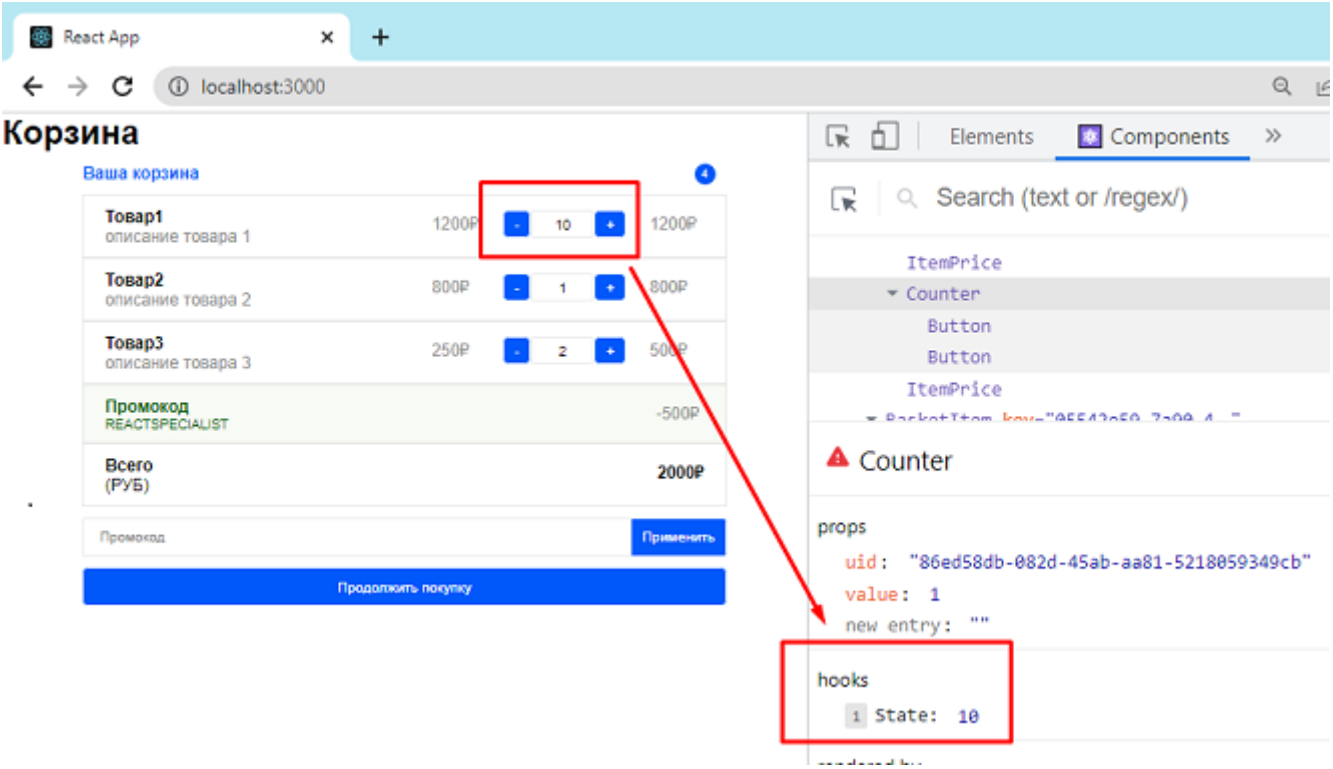
```
const [qty, setQty] = React.useState(value);
```

1.2. Измените код **src\components\Counter.js** так, чтобы при нажатии на кнопки `-` и `+` происходил вызов функции `setQty` для изменения текущего состояния. Выведите текущее состояние `qty` в свойство `value` элемента `input`.

```
return (
    <div className="Counter">
      <div className="Counter_into">
        <Button
          value="-"
          onClickHandler={() => {
            setQty(qty-1)
          }}
        />
        <input
          className="Counter_input"
          value={qty}
          onChange={(ev) => {
            setQty(ev.target.value);
          }}
        />
        <Button
          value="+"
          onClickHandler={() => {
            setQty(qty+1)
          }}
        />
      </div>
    </div>
  );
```

1.3. Убедитесь, что

- нет ошибок в консоли (если есть, то исправьте)
- при нажатии на кнопки `-` и `+` изменяется значение в `input`
- проверьте, изменяется ли состояние компонента в *Devtools*



1.4. Давайте оптимизируем код: Нужно задать для изменения значений создать отдельные функции `qtyButtonAdd` - увеличивает `qty`на единицу - `setQty(qty + 1)` `qtyButtonSub` - уменьшает `setQty(qty - 1 > 0 ? qty - 1 : 0)` `qtyInputChange` - проверяет ввод в `<input />`

```
const qtyInputChange = (ev) => {
  let qty = parseInt(ev.target.value);

  if (qty < 0 || isNaN(qty)) {
    qty = 0;
  }

  setQty(qty);
};
```

Итоговый вид возвращаемого из значения `Counter` должно быть такое

```
<div className="Counter">
     <div className="Counter_into">
       <Button value="-" onClickHandler={qtyButtonSub} />
       <input
         className="Counter_input"
         value={qty}
         onChange={qtyInputChange}
       />
       <Button value="+" onClickHandler={qtyButtonAdd} />
     </div>
   </div>
```

Упражнение 2: Подъём состояния

В этом упражнении мы изменим структуру работу с состоянием так, чтобы при изменении количества товаров в позиции пересчитывалась цена товара за нужно количество товаров и общая цена. Чтобы это сделать, нам будет нужно будет перенести (поднять) состояние в вышележащий компонент, то есть в `Basket`. **Примечание:** ещё лучше было бы разместить в `App`

2.1. Измените компонент `Basket` так, чтобы свойство `items` попадало в `useState`:

```
const Basket = ( props ) => {

  const [items, setItems] = React.useState(props.items);
```

2.2. Передайте `setItems` и `items` в качестве свойств в `BasketItem`:

```
{items.map((item) => (
  <BasketItem {...item} key={item.uid} items={items} setItems={setItems} />
))}
```

2.3. Получите `setItems` и `items` в компоненте `BasketItem` и передайте их далее в `Counter`

```
const BasketItem = ({
  uid,
  title,
  description,
  price,
  qty,
  setItems,
  items
}) => {

  return (
    <div className='BasketItem'>
      <ItemInfo title={title} description={description} />
      <ItemPrice value={price} currency={'₽'} />
      <Counter value={qty} uid={uid} items={items} setItems={setItems} />
      <ItemPrice value={qty * price} currency={'₽'}  />
    </div>
  )
}
```

Примечание: теперь внутри `Counter` мы можем вызывать `setItems`, а состояние будет меняться в `Basket`, вызывая перерисовку нужного содержимого. Это и есть *подъём состояния*.

2.4. Откройте компонент `Counter` и примите свойства `items` и `setItems`

```
const Counter = ({ value, uid, items, setItems }) => {
```

2.5. Закомментируйте весь код до `return` внутри компонента `Counter` и опишите константу `qty`.

```
const Counter = ({ value, uid, items, setItems }) => {
  // const [qty, setQty] = React.useState(value);

  // const qtyButtonAdd = () => setQty(qty + 1);

  // const qtyButtonSub = () => setQty(qty - 1 >  0 ? qty - 1 : 0);

  // const qtyInputChange = (ev) => {
  //   let qty = parseInt(ev.target.value);

  //   if (qty < 0 || isNaN(qty)) {
  //     qty = 0;
  //   }

  //   setQty(qty);
  // };

  const qty = value;
```

2.6. Внутри компонента `Counter` перепишите функцию `qtyButtonAdd` так, чтобы она **увеличивала** количество нужного товара при нажатии на кнопку

```
  const qtyButtonAdd = () => {
    // получаем новый массив на основе данных вышележащего состояния
    let newItems = items.slice();

    // находим индекс элемента, в котором будем менять количество
    let index = newItems.findIndex( item => item.uid == uid )

    // меняем количество у найденного элемента
    newItems[index].qty++;

    // устаналиваем новое состояние
    setItems(newItems);
  };
```

2.7. Внутри компонента `Counter` перепишите функцию `qtyButtonSub` так, чтобы она **уменьшала** количество нужного товара при нажатии на кнопку

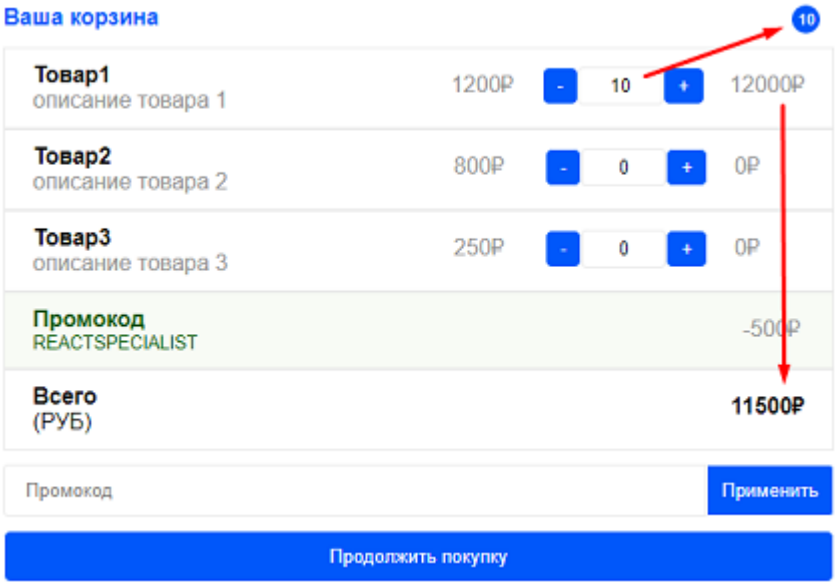
```
  const qtyButtonSub = () =>{
    let newItems = items.slice();
```

```
    let index = newItems.findIndex( item => item.uid == uid )

    newItems[index].qty--;

    if (newItems[index].qty < 0 || isNaN(newItems[index].qty)) {
      newItems[index].qty = 0;
    }

    setItems(newItems);
  }
```

2.8. Внутри компонента `Counter` перепишите функцию `qtyInputChange` так, чтобы она **изменяла** количество нужного товара при нажатии на кнопку

```
  const qtyInputChange = (ev) => {
    let newItems = items.slice();

    let index = newItems.findIndex( item => item.uid == uid )

    newItems[index].qty = parseInt(ev.target.value);

    if (newItems[index].qty < 0 || isNaN(newItems[index].qty)) {
      newItems[index].qty = 0;
    }

    setItems(newItems);
   }
```

2.9. Убедитесь, что при нажатии на кнопки увеличения и уменьшения товара, а также при введении данных в `<input />` корректно выводится кол-во товаров в правом нижнем углу корзины и общая сумма.



Упражнение 3: AJAX и обращение к API

В этом упражнении мы изменим `App` так, чтобы перечень товаров `items` приходил через `fetch` запрос к серверу. Самое главное в этом примере не бизнес-логика, а реализация AJAX-запроса и получение ответа.

3.1. Создайте файл **public\items.json** и перенесите туда содержимое `items`. **Примечание:** JSON требует названия свойств заключать в двойные кавычки, потому нажмите правую кнопку мыши на коде внутри **public\items.json** и выберите форматирование (я использовал Prettier)

```
[
  {
    "uid": "86ed58db-082d-45ab-aa81-5218059349cb",
    "title": "Товар1",
    "description": "описание товара 1",
    "price": 1200,
    "qty": 1
  },
  {
    "uid": "05542e59-7a90-4e80-bf9d-78967f272049",
    "title": "Товар2",
    "description": "описание товара 2",
    "price": 800,
    "qty": 1
  },
  {
    "uid": "7793e4f0-fe86-47cc-98f6-e01b6beeb3af",
    "title": "Товар3",
    "description": "описание товара 3",
    "price": 250,
    "qty": 2
  }
]
```

3.2. Импортируйте в `App` объект `React` , хуки `useState` и `useEffect` и укажите состояния для хранения ошибки, индикации загрузки и самих товаров в корзине

```
  const [startItems, setStartItems] = React.useState([]);
  const [error, setError] = useState(null);
  const [isLoaded, setIsLoaded] = useState(false);
```

3.3. Опишите вызов `useEffect`

```
  useEffect(() => {
    fetch("http://localhost:3000/items.json")
      .then(res => res.json())
      .then(
        (result) => {
          setIsLoaded(true);
          setStartItems(result);
```

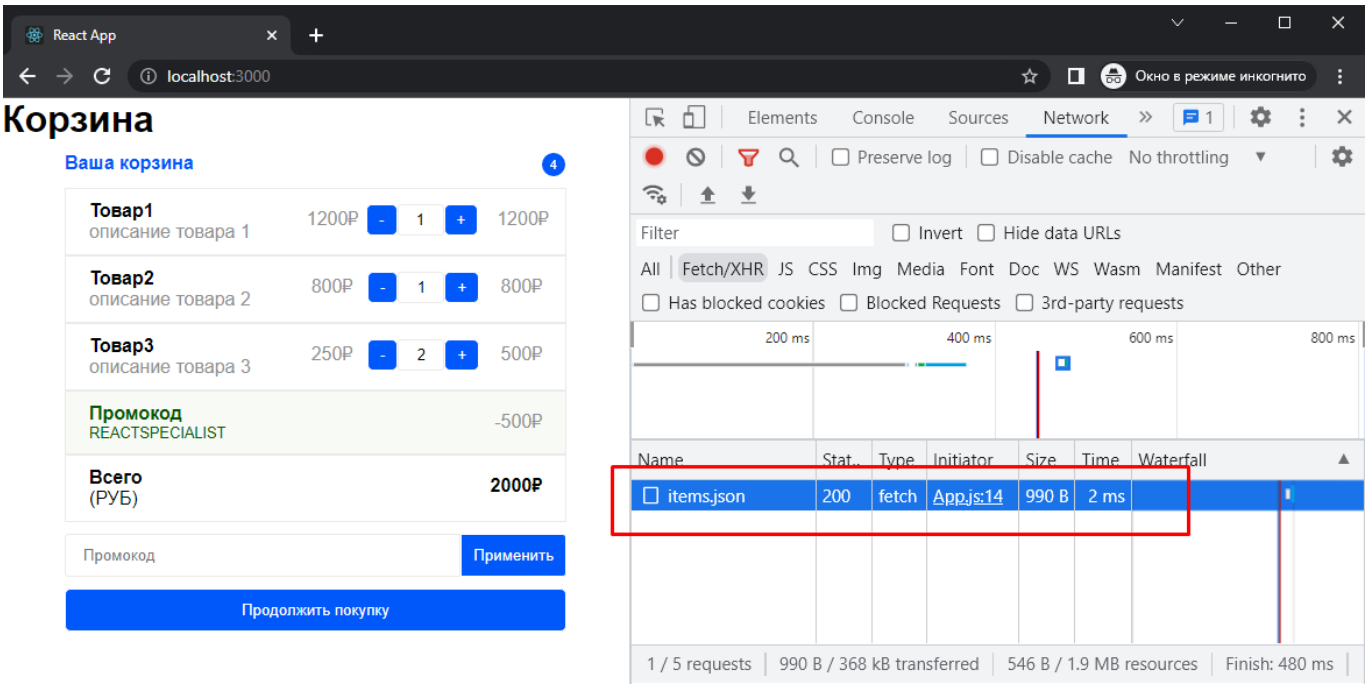
```
        },
        (error) => {
          setIsLoaded(true);
          setError(error);
        }
      )
  }, [])
```

3.4. Перепишете оставшуюся часть компонента с учетом отображения индикатора загрузки и ошибки

```
  let basketPlace = null;

  if (error) {
    basketPlace = <div>Ошибка: {error.message}</div>;
  } else if (!isLoaded) {
    basketPlace = <div>Загрузка...</div>;
  } else {
    basketPlace = <Basket items={startItems} />
  }

  return (
    <div className="App">
      <header className="App-header">
        <h1>Корзина</h1>

        {basketPlace}
      </header>
    </div>
  );
```

3.5. Убедитесь, что показываются товары из `items.json`

3.6. Откройте панель разработчика в Google Chrome через `F12` и далее вкладку `Network` - после обновления страницы там виден благополучный запрос со статусом `200` или `304`



Упражнение 4: Удаление товара из корзины (для самостоятельной домашней работы)

Добавьте в `BasketItem` ссылку на удаление отдельной позиции корзины. Навесьте обработчик, который будет удалять товар из корзины на основе `items` и `setItems`

Упражнение 5: Применение промокода (для самостоятельной домашней работы)

Добавьте в `App` состояние отвечающее за хранение промокода. Компонент `<BasketPromoInfo />` должен показываться только когда промокод введён в `<BasketPromoCode />`

##Выводы

- мы добавили состояние в реальный компонент корзины - `Counter`
- реализовали подъём состояния - теперь состояние хранится в `Basket`
- текущая реализация далеко не идеальна:
  - в `Counter` передаётся список всех товаров корзины
  - мы вынуждены прокидывать свойства `items` и `setItems` через несколько компонентов
  - прокидывание свойств усложняет работу и влечёт к ошибкам
  - чтобы исправить ситуацию, можно воспользоваться хуком useContext
- создание отдельных состояний у разных компонентов усложняет работу и отладку. **Лучше** хранить в одном месте
- мы написали пример использования состояния при выполнении AJAX-запроса
- узнали о хуке `useEffect` - он отвечает за сторонние эффекты