REACT. Корзина с товарами

Установка

Создание react приложения

npx create-react-app start-app

Установка node-sass

npm i node-sass

Перенос верстки в случае с sass и отдельными папками под блоки (корзина с товарами)

Удаляем из **src** все лишнее. Правим index.js и App.js.

Создадим src/components/App. Перенесем и переименуем расширение для App.js - src/components/App/App.jsx. Перенесем src/components/App/_base.scss, _reset.scss, _vars.scss, _section-cart.scss Подключим их в **App.jsx**

```
import "./_vars.scss"
import "./_reset.scss"
import "./_base.scss"
import "./_section-cart.scss";
```

Переносим в App.jsx всю верстку из index.html. Меняем **class** на **className.** Закроем все одиночные теги img и input

Создадим папку для заголовка - src/components/Tiitle. Создадим в ней Title.jsx. В нем scf-tab:

```
const Title = () => {
  return <h1 className="title-1">Корзина товаров</h1>;
};
export default Title;
```

Подключим его к App.jsx. В Title.jsx подключим для него стиль _titles.scss и переименуем его _style.scss. Можем также переименовать Title.jsx в index.jsx Тогда в импорте в App.jsx можно указать только папку:

```
import Title from "../Title";
```

Создадим src/components/Cart/Cart.jsx. Все подключим.

Создадим src/components/CartHeader/index.jsx и style.scss. Все подключим.

Создадим src/components/Ptoduct/index.jsx и style.scss. Все подключим.

Создадим src/components/CartFooter/index.jsx и style.scss. Все подключим.

Подключим файл с переменными внутри стилей для некоторых компонентов, добавим в стилях style.scss сверху строку:

```
@import "../App/vars";
```

Создадим для счетчика src/components/Count/index.jsx и style.scss. Все подключим.

Создадим для кнопки удаления src/components/ButtonDelete/index.jsx. Все подключим.

json-server

npm i json-server

Создадим для него json-файл в корне приложения src/data.json и передадим туда данные по продуктам в таком виде:

```
{
  "products": [
    {
      "id": 1,
      "img": "macbook.jpg",
      "title": "Apple MacBook Air 13",
      "count": 1,
      "price": 110000,
      "priceTotal": 110000
    },
    {
      "id": 2,
      "img": "apple-watch.jpg",
```

```
    "title": "Apple watch",
    "count": 1,
    "price": 29000,
    "priceTotal": 29000
   },
   {
    "id": 3,
    "img": "mac-pro.jpg",
    "title": "Mac Pro",
    "count": 1,
    "price": 190000,
    "priceTotal": 190000
   }
  ]
}
```

Запускаем сервер следующей командой:
npx json-server --watch src/data.json --port 8000

Компонент с состоянием - cart

```
const Cart = () => {
 const [cart, setCart] = useState(null); // корзина с товарами
 const [total, setTotal] = useState(null); //общее количество и сумма в футере
 const [fetchData, setFetchData] = useState(true);  // флаг. При его изменении произойдет обновление состояния, выполнится fetch запрос, и страница перерендерится

 useEffect(() => {
  fetch("http://localhost:8000/products")
   .then((res) => res.json())
   .then((data) => {
    setCart(data);
   });
 }, [fetchData]);
...
}
```

Удаление товара

Используем флаг для того, чтобы состояние изменилось уже после удаления записи с сервера. В Cart.js

```
 const deleteProduct = (id) => {
  fetch("http://localhost:8000/products/" + id, {
   method: "DELETE",
  }).then((res) => {
   res.ok && setFetchData((value) => !value);
  });
 };
```

fetch для increase

Используем для fetch метод PUT. И также используем флаг, чтобы изменить состояние после сделанного fetch. В Cart.js

```
 const increase = (id) => {
  const product = cart.find((product) => id === product.id);

  const data = {
   ...product,
   count: product.count + 1,
   priceTotal: (product.count + 1) * product.price,
  };

  fetch("http://localhost:8000/products/" + id, {
   method: "PUT",
   headers: { "Content-Type": "application/json" },
   body: JSON.stringify(data),
  }).then((res) => {
   res.ok && setFetchData((value) => !value);
  });
 };
```

fetch для decrease

Все аналогично

```
const decrease = (id) => {
  const product = cart.find((product) => id === product.id);
  const newCount = product.count - 1 > 1 ? product.count - 1 : 1;

  const data = {
    ...product,
    count: newCount,
    priceTotal: newCount * product.price,
  };

  fetch("http://localhost:8000/products/" + id, {
    method: "PUT",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  }).then((res) => {
    res.ok && setFetchData((value) => !value);
  });
};
```

fetch для changeValue

Все аналогично

```
const changeValue = (id, value) => {
  const product = cart.find((product) => id === product.id);

  const data = {
    ...product,
    count: value,
    priceTotal: value * product.price,
  };

  fetch("http://localhost:8000/products/" + id, {
    method: "PUT",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  }).then((res) => {
    res.ok && setFetchData((value) => !value);
  });
};
```

Добавление товара

При клике по кнопке будет добавляться случайный товар

```
const addProduct = () => {
  const titles = ["Apple MacBook Air 13", "Apple watch", "Mac Pro"];
  const images = ["macbook.jpg", "apple-watch.jpg", "mac-pro.jpg"];
  const prices = [110000, 29000, 190000];

  const randomValue = (array) => {
    return array[Math.floor(Math.random() * array.length)];
  };

  const price = randomValue(prices);

  const data = {
    img: randomValue(images),
    title: randomValue(titles),
    count: 1,
    price: price,
    priceTotal: price,
  };

  fetch("http://localhost:8000/products/", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
```

```
    body: JSON.stringify(data),
   }).then((res) => {
    res.ok && setFetchData((value) => !value);
   });
  };
```

AppContext. Проброска состояния. Props drilling

Часто бывает нужно "пробросить" какие-либо свойства через несколько компонентов (они могут быть как в одном файле, так и в разных). 1-ый вариант - через **пропсы**. Другой вариант - использовать **контекст.** Подключим его в Cart.js:

```
import { useState, useEffect, createContext } from "react";
```

Перед описанием компонента Cart вставляемЖ

```
export const AppContext = createContext(null);
```

В других компонентах, куда мы хотим пробросить свойства, нам необходимо их обернуть в AppContext.Provider и указать значения, которые хотим передать. Рассмотрим пример для передачи функции **deleteProduct**. В Cart.js:

```
 return (
  <AppContext.Provider value={{deleteProduct}}>
   <section className="cart">
    <CartHeader />
    {cart &&
     cart.map((product) => {
      return (
       <Product
        key={product.id}
        product={product}
        increase={increase}
        decrease={decrease}
        changeValue={changeValue}
       />
      );
     })}
    {total && <CartFooter total={total} />}
   </section>
   <section className="button-wrapper">
    <Button title="Add product" onClick={addProduct} />
   </section>
  </AppContext.Provider>
 );
};
```

Через пропсы дальше уже не указываем deletePruduct. Удаляем передачу deleteProduct в Product.js. В ButtonDelete.js сделаем импорты и достанем из контекста:

```
import { useContext } from "react";
import { AppContext } from "../cart";

const ButtonDelete = ({ id }) => {
 const { deleteProduct } = useContext(AppContext);

 return (
  <button
   type="button"
   onClick={() => {
    deleteProduct(id);
   }}>
   <img src="./img/icons/cross.svg" alt="Delete" />
  </button>
 );
};

export default ButtonDelete;
```

Доработка AppContext

В Cart.js немного переделаем вывод корзины:

```
const products = () => {
  return cart.map((product) => {
    return (
      <Product
        key={product.id}
        product={product}
        increase={increase}
        decrease={decrease}
        changeValue={changeValue}
      />
    );
  });
};

return (
  <AppContext.Provider value={{ deleteProduct }}>
    <section className="cart">
      <CartHeader />
      {cart && products()}
      {total && <CartFooter total={total} />}
    </section>
    <section className="button-wrapper">
      <Button title="Add product" onClick={addProduct} />
    </section>
  </AppContext.Provider>
);
```

Далее разбираемся с остальным контекстом:

```
const products = () => {
  return cart.map((product) => {
    return <Product key={product.id} product={product} />;
  });
};

return (
  <AppContext.Provider value={{ deleteProduct, increase, decrease, changeValue, addProduct }}>
    <section className="cart">
      <CartHeader />
      {cart && products()}
      {total && <CartFooter total={total} />}
    </section>
    <section className="button-wrapper">
      <Button title="Add product" />
    </section>
  </AppContext.Provider>
);
```