МфСсылка на телегу

https://t.me/archakovblog

Дизайн: https://www.figma.com/file/fw0toTyXMwM1y4WIe0YFrJ/React-Projects?node-id=0%3A1

Исходник проекта: https://github.com/Archakov06/react-sneakers

Библиотека с готовыми классами macro-css: <https://github.com/Archakov06/macro-css>

Как создать react проект:

Загружаем node.js на комп. С сайта node.js

Открываем терминал (командную строку) на компе или в

Visual Studio Code (VS Code - это редактор кода)

пишем команду npm i -g create-react-app в терминале VS Code или в коммандной строке

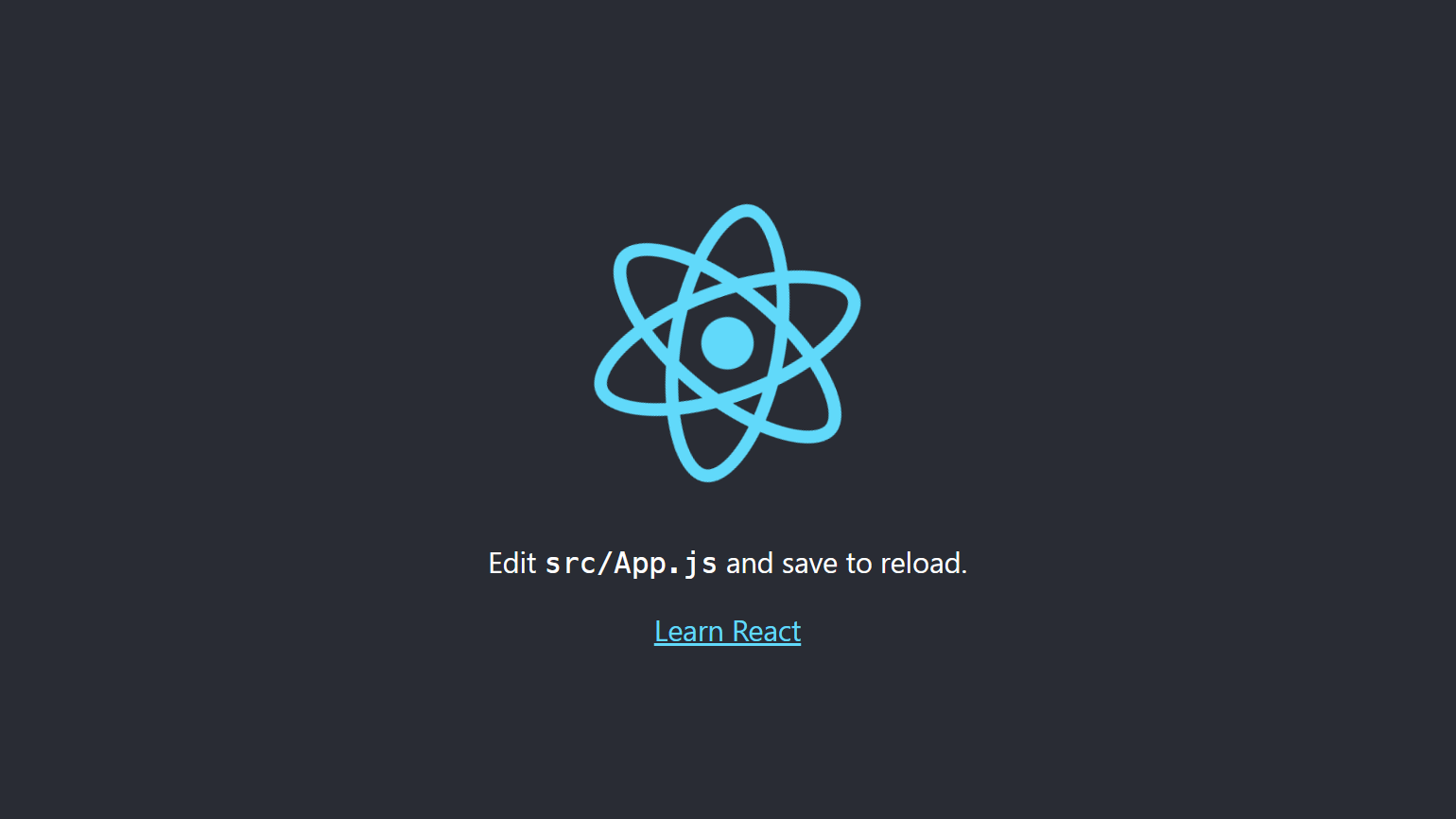
и скачиваем пакет create-react-app с помощью npm:

Пишем create-react-app my-first-app и создаем react проект под названием my-first-app в командной строке можно выбрать где он будет размещен( какой диск, какая папка) у меня он размещен здесь: C:\Users\Nik> т. е. Полностью команда выглядела так C:\Users\Nik> create-react-app my-first-app

Если будет ошибка то пишем команду npm i web-vitals --save-dev перед запуском проекта с помощью npm start.

В C:\Users\Nik> пишем cd my-first-app и переходим в наш проект

дальше запускаем его командой npm start ( C:\Users\Nik\my-first-app> npm start)

(с помощью команды cd мы перешли в папку проекта, а командой start запустили локальный сервер разработки http://localhost:3000/. В результате вы должны увидеть в браузере такую красоту:

Хороший гайд по этому здесь:

<https://skillbox.ru/media/code/reactjs-dlya-novichkov-v-programmirovanii-chto-eto-kak-ustroen-i-zachem-nuzhen/>

В реальности в проекте мы работаем с тремя файлами

App.js в нем мы пишем наш код на JSX,

index.js в нем то что написали в App.js переводим в понятный браузеру HTML

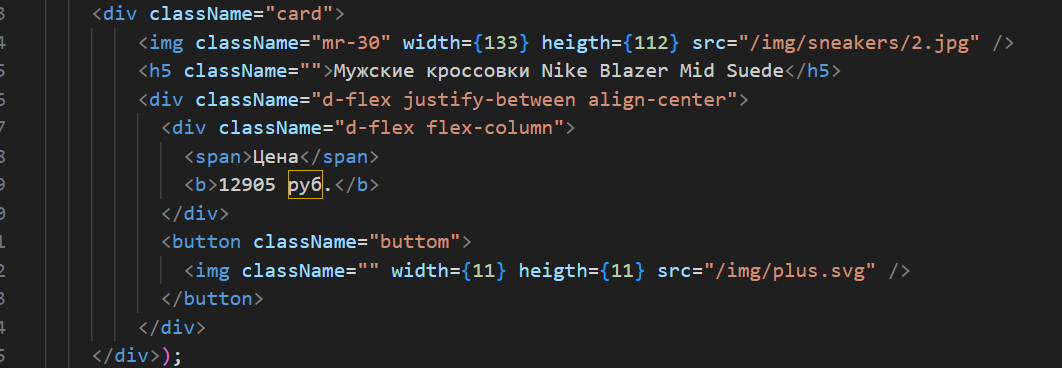
index.css подключаем для добавления стилей

Как создать и что такое компонент:

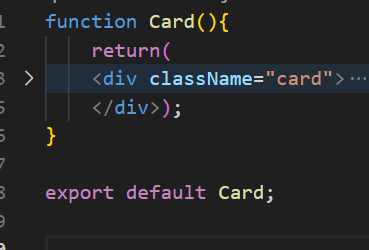
У нас несколько карточек код которых идентичен.

Берем код одной из карточек переносим в отдельный файл это и будет компонент (называем его например Card).

Вот код карточки:



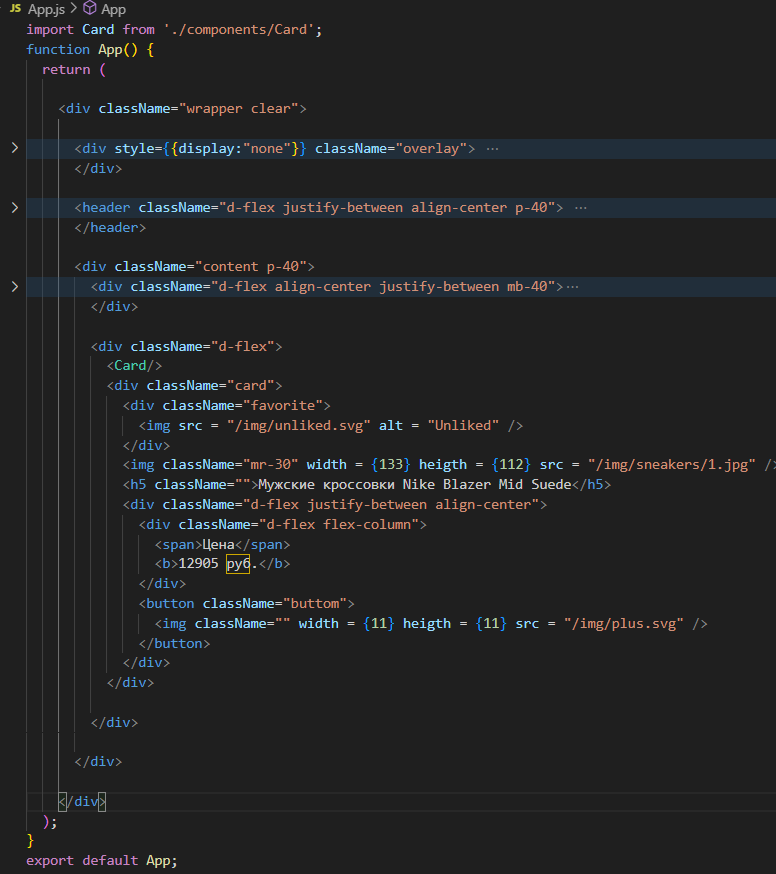
Этот код заворачиваем в функцию и очень важно разрешаем его экспорт по умолчанию (код естественно свернут).

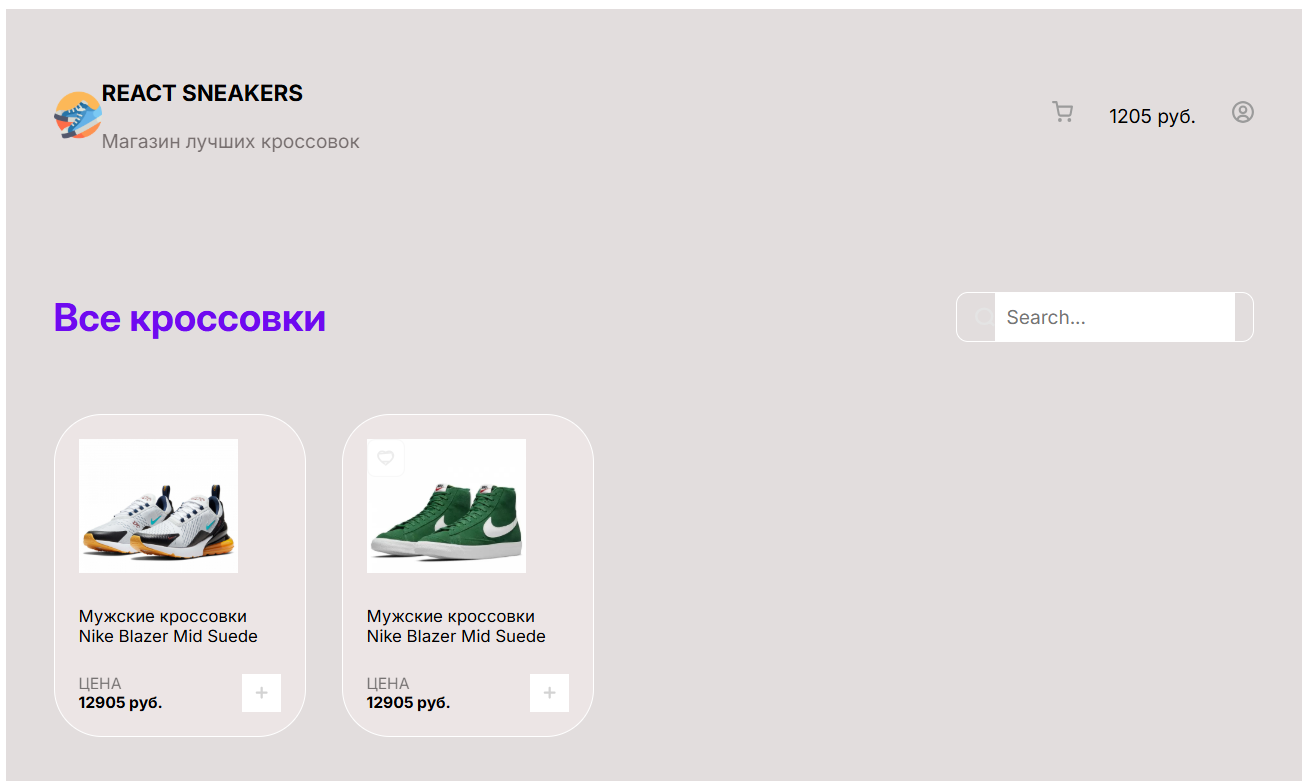


И с помощью импорта import Card from './components/Card';

подключаем  <Card/> файл в наш App.js где у нас изначально и были эти карточки.

Для примера в этом коде одна карточка просто написана кодом а другая подключена с помощью компонента

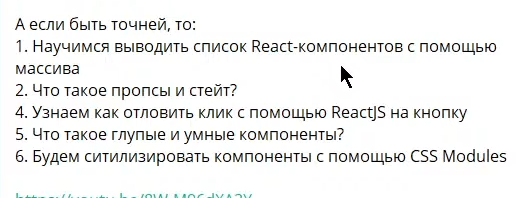


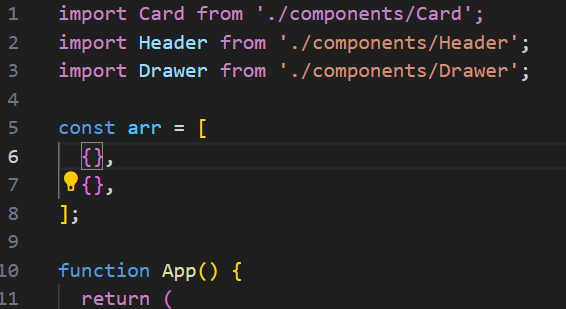


Получаем такую картинку но первая карточка подключена с помощью компонента а вторая просто кодом.

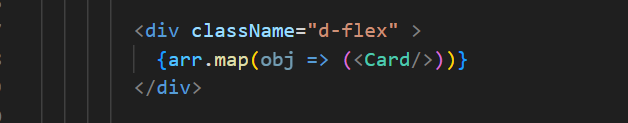
Так же делаем компонент из Header.

Третий урок что сделаем:



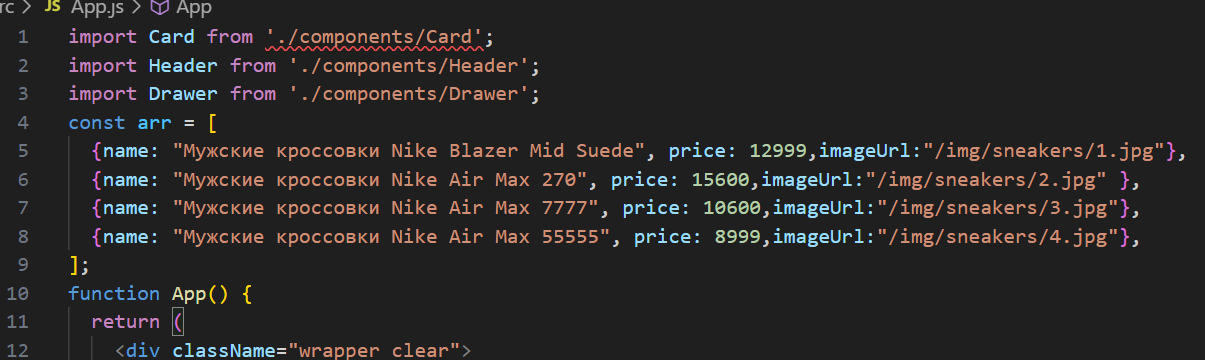
Создаем массив

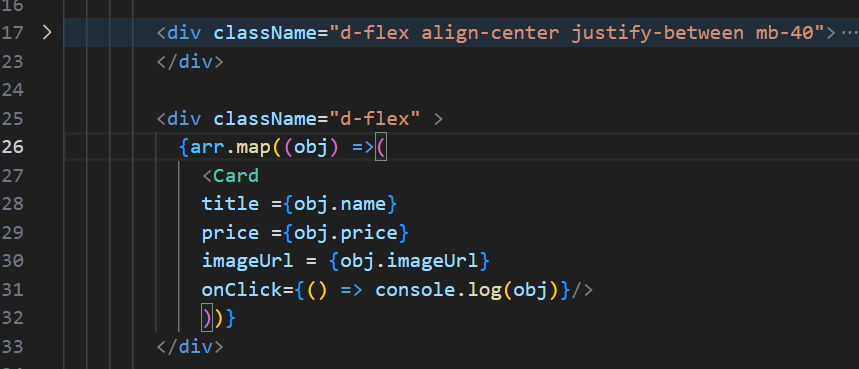
в нем два пустых обьекта {} и {}.

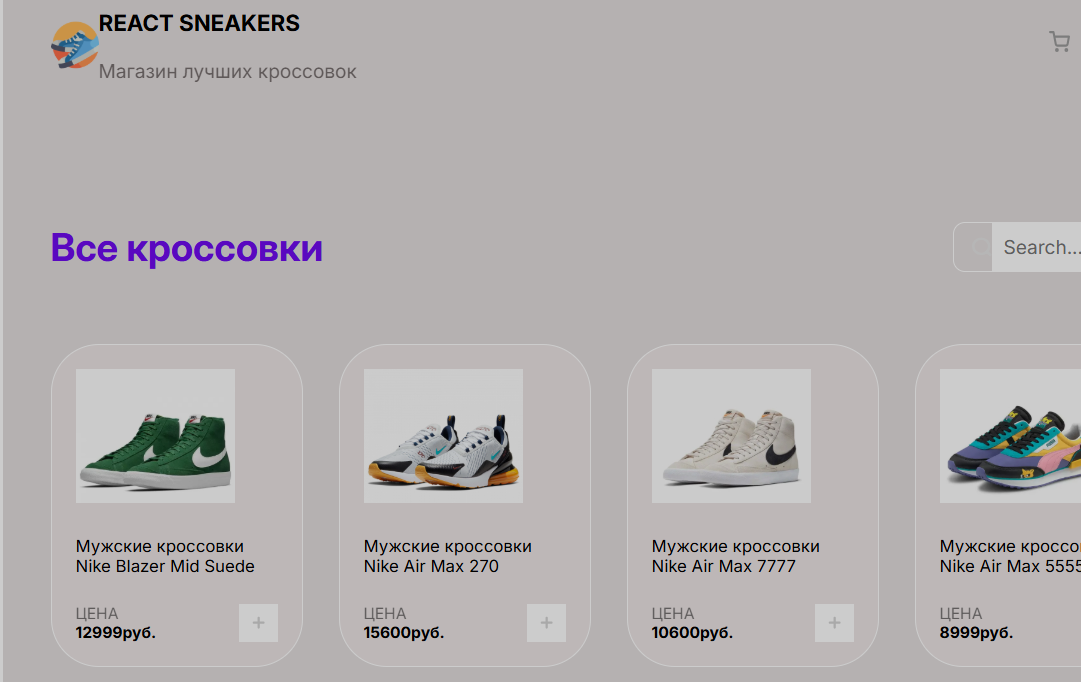
Там где подключали карточки запускаем ф.map с созданным массивом в которой при прохождении каждого обьекта возвращается компонент Card в итоге получаем страницу с двумя карточками (т. к. в arr. сейчас 2 обьекта.).

Мы превратили массив arr в список react компонентов.

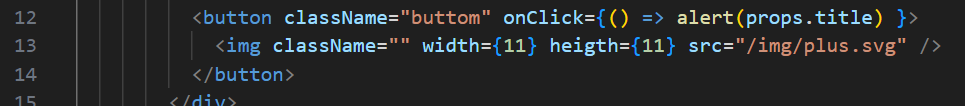
Добавляем в нашу ф. Саrd(props) props- это объект в который помещаем наши значения, цепочка такая в App.js у нас массив arr со значениями,

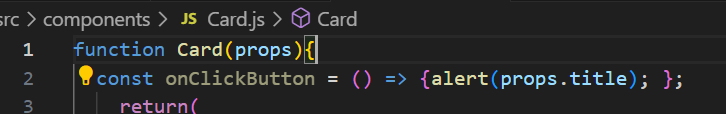


ниже здесь же в App.js c помощью map передаем значения arr в компонент Card

На выходе получаем 4ре карточки т.к.в arr 4ре обьекта.

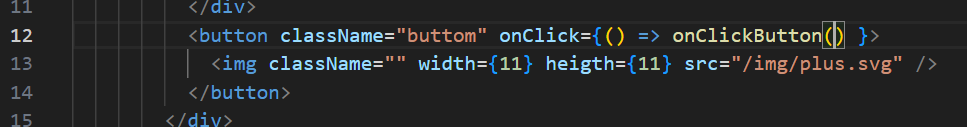
**Отлавливаем Click**

 в Card на button пишем событие onClick при котором в алерте выводим на экран для примера то что в props.title.Теперь при нажатии кнопки появляется то что в props.title



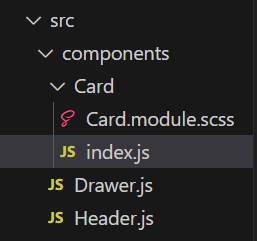
Здесь облагораживаем все создаем const onClickButton

туда засовываем ф.которая выводит в alert props.title и уже ее вызываем при событии onClick

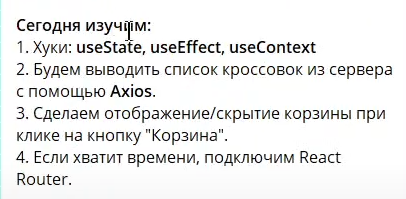


и при клике на button

выводиться props.title

**Стилизация с помощью CSS Modules**

Создаем папку Card создадим там отдельный файл Card.module.scss туда переносим стили карточки с index.scss потом этот файл подключаем к index.js (это переименованный Card.js, его переименовали в index.js чтобы комп находясь в этой папке по умолчанию обращался к нему) Получается что в папке Card мы собрали все что относится к Card,в том числе и стили карточки.



**дополнение с обьяснением useState** **4урок 1.08.00 — 1.16.00**

**Задача кнопка плюс в Card /index.js при нажатии стала зеленой.**

**Сначало import React from "react";**

**В Card /index.js задаем хук useState**

**const [isAded, setIsAded] = React.useState();** по умолчанию он false

Создаем const onIsAded = () => {setIsAded(true);}

Там где у нас плюс вешаем onClick ={onIsAded} при клике вызываем onIsAded которая задает setIsAded(true)

src = { setIsAded ? "/img/btn-checked.svg" : "/img/btn-plus.svg"}

при клике вызываем onIsAded которое меняет значение setIsAded на true и в src уже выбирается первая картинка.Вот реальный код на картинке:

**<img** className={styles.plus} onClick={onClickPlus}

src={isAded ? "/img/btn-checked.svg" : "/img/btn-plus.svg" } alt="cheked"

если в **const onIsAded = () => {setIsAded(true);}** заменим **{setIsAded(true);} на {setIsAded(!isAded);**

**1.19.00 useEffect**

С помощью use state можем открывать и закрывать корзину **4урок 1.08.00**

function App() {

   const [cartOpened, setCartOpened] = React.useState(false); //таким способом добавили хук useState

  return (

    <div className="wrapper clear">

      {cartOpened ? <Drawer /> : null} если React.useState(false) то срабатывает null и корзина Drawer закрыта если вставим true то сработает <Drawer /> и все что в этом компоненте отобразиться ( корзина откроется).

Все это делаем вручную т. е. Пишем в программе true or false теперь cделаем что бы при нажатии на корзинку в React.useState(false) false менялась на true т. е. Надо добавить событие onClick

проблема в том что функция находиться в App.js а li на который вешаем onClick в Header.js

<ul className="d-flex">

  <li className="mr-30 cu-p" >

    <img className="mr-30" width = {18} heigth = {18} src = "/img/cart.svg" />

    <span>1205 руб.</span>

  </li>

К счастью в любой реактовский компонент (например Header) мы можем передать props

Мы в Headеr в App.js зададим событие onClickCart которое будет происходить при клике, делаем это так

<Header onClickCart ={() => setCartOpened(true)} /> при этом событии запускается анонимная ф при выполнении которой setCartOpened(true) получает значение true но сейчас кроме как здесь этого события onClickCart нигде нет.

для того что бы событие onClickCart появилось в компоненте Header нужно ему (компоненту) задать props вот так function Header(props){

теперь мы спокойно при клике на li в Header.js можем достать событие onClickCart т.к.оно есть у нас в Header(props)

Навешиваем на li событие onClick( li onClick) и при наступлении события вытаскиваем из props событие onClickCart которое запускает анонимную ф

onClickCart ={() => setCartOpened(true) и это в App.js меняет в setCartOpened false на true и сооьветственно в {cartOpened ? <Drawer /> : null} срабатывает <Drawer /> а не null и открывается корзина. Внизу показал как навешивается клик на li

<ul className="d-flex">

  <li onClick={props.onClickCart} className="mr-30 cu-p" >

    <img className="mr-30" width = {18} heigth = {18} src = "/img/cart.svg" />

    <span>1205 руб.</span>

  </li>

Теперь сделаем так что бы корзина закрывалась при нажатии на крестик на корзине он у нас в Driver.js представлен картинкой

<h2 className="mb-30 d-flex justify-between">

          Корзина{" "}

          <img className="removeBtn cu-p"

            src="img/btn-remove.svg alt="Close" />

    </h2>

Идем в App.js задаем в Drawer аргумент onClose и вставляем туда анонимную ф которая передает в setCartOpened значение (false)

пишется это так <Drawer onClose ={() => setCartOpened(false)} /

выглядит так

function App() {

   const [cartOpened, setCartOpened] = React.useState(false); //таким способом добавили хук useState

  return (

    <div className="wrapper clear">

      {cartOpened ? <Drawer onClose ={() => setCartOpened(false)} /> : null} получается если у

Drawer вызван onClose срабатывает setCartOpened(false) и Drawer будет закрыт т.е. null поэтому эту часть можно сократить вот так

{cartOpened && <Drawer onClose ={() => setCartOpened(false)} />}

теперь onClose надо передать с помощью props в Drawer переходим в Drawer.js и там пишем function Drawer(props) {и там же вешаем onClick на img и задаем что бы при onClick из props вытаскивался onClose который запускает setCartOpened(false) это мы в App.js задавали (onClose ={() => setCartOpened(false) вот внизу повесили onClick на img-+

<h2 className="mb-30 d-flex justify-between">

          Корзина{" "} <img onClick = {props.onClose}

            className="removeBtn cu-p" src="img/btn-remove.svg" alt="Close"/> </h2>

Теперь сделаем так что бы данные которые в карточках. хранились на бэк энде . Для симуляции сервера на котором хранятся данные есть прога mockAPI там создаем json c нашими данными и получаем ссылку на него. Логинимся на mockAPI через GitHab niklebed71 Вот ссылка https://67a5d781510789ef0df96fa9.mockapi.io/items

Дальше в нашем App.js делаем так что бы при работе данные брались с бэк энда т. е.

был переход на ссылку и оттуда приходили данные.

Для этого первое, убираем наш arr со всеми данными карточек, а вместо него добавим хук useState, в нем задаем items и соответственно setItems вот так:

function App() {

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

данные по нашим карточкам теперь в бэк энде и когда они поступят после запроса в React.useState([]) мы будем эти данные перебирать и выводить на экран вот здесь:

  <div className="d-flex flex-wrap">

          {items.map((obj) => (

 // т.к. берем данные с backend  то вместо arr.map((obj) пишем items.map((obj) т.к.

//в const [items, setItems] = React.useState([]) у нас каждая карточка обозначена как items

            <Card

              title={obj.name}

              price={obj.price}

              imageUrl={obj.imageUrl}

              onFavorite={() => console.log("Добавили в закладки")}

              onPlus={() => console.log("Нажали плюс")} />

Для получения данных с бэк энда используем fetch(это штука которая помогает отправлять запросы на серверную часть) В нашем случае это выглядит так:

const [cartOpened, setCartOpened] = React.useState(false); //так добавили хук useState

   fetch('https://67a5d781510789ef0df96fa9.mockapi.io/items')

  return ( <div className="wrapper clear">

В консоле в network мы можем увидеть что наш запрос вернул массив с нашими кросами

Tо что возвращает Fetch (принести) называется promis (обещание) но теперь надо его вытащить из fetch, в json формате

Делаем это так

 fetch('https://67a5d781510789ef0df96fa9.mockapi.io/items').then((res)=> {

    return res.json(); })

Дословно посылаем fetch и затем его ответ res вовращаем в json формате

Проверим все ли ОК и отобразим полученный respons(ответ) в консоле

   then((json) =>{console.log(json);});

Все ок получили с бэк энда инфу в json формате но задача не отображать в consol, а отобразить на сайте. Для этого вместо consol, передадим json в setItems. Именно в setItems ( он задаст значение items и сделает рендинг экрана и у нас все отобразиться) а не в items .

function App() {

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

   const [cartOpened, setCartOpened] = React.useState(false); // добавили хук useState

   fetch('https://67a5d781510789ef0df96fa9.mockapi.io/items').then((res)=> {

    return res.json();

   })

   .then((json)=>{

    setItems(json);

   });

Почти работает но у нас очень большое к-во запросов на mockApi и сервер может заблокировать отправку данных.

Каждый раз как что то меняем в любом компоненте где есть React.useState( ) ( в нашем случае это App( ), Card,Drawe,) функция App( ) перевызывается и у нас отправляется запрос на бэкэнд.

fetch('https://67a5d781510789ef0df96fa9.mockapi.io/items').then((res)=> {

    return res.json();

   })

   .then((json)=>{

    setItems(json);

   });

Проблема в том что нам запрос на бэкэнд надо выполнять только один раз когда компонент App.js отобразиться первый раз, только в этом случае надо отправить запрос на бэкэнд а не каждый раз когда происходит рендер App,js. Как решить эту проблему:

Для этого используем хук  React.useEffect

С помощью этого хука можем отслеживать состояние ( первый рендер,обновление,удаление) и в зависимости от состояния код внутри хука выполняется либо только один раз при первом рендере и удалении или каждый раз при обновлении(изменении) любого компонента.

В нашем случае мы с помощью этого хука получаем запрос с бэк энда только один раз при первом рендере делаем это так:

React.useEffect(()=>{

    fetch('https://67a5d781510789ef0df96fa9.mockapi.io/items').then((res)=> {

return res.json();}) .then((json)=>{setItems(json);}); },[]);

  Помещаем наш запрос с ответом в хук. Чтобы заставить useEffect срабатывать только при первом рендеринге компонента, необходимо передать пустой массив ([]) в качестве второго аргумента функции. Первый аргумент наш запрос с ответом в json а второй пустой массив ([]).

Делаем что бы при нажатии на плюс товар добавлялся в корзину. 2.14.00

Добавляем хук const [cartItems, setCartItems]

 const [items, setItems] = React.useState([]);// карточки товаров с бэкэнда

const [cartItems, setCartItems] = React.useState([]); // хук useState добавление товара в корзину

const [cartOpened, setCartOpened] = React.useState(false); // открытие и закрытие корзины

в cartItems мы будем храним массив с теми товарами которые в корзине и его надо перенести в Drawer.js(корзина).В Drawer.js в function Drawer(props) у нас есть props в нем есть onClose и items их мы задали в App.js  вот здесь:

return (

    <div className="wrapper clear">

      {cartOpened && <Drawer items = {cartItems} onClose ={() => setCartOpened(false)} />}

      <Header onClickCart ={() => setCartOpened(true)} />

вот еще раз эта строка:

{cartOpened && <Drawer items = {cartItems} onClose ={() => setCartOpened(false)} />}

Проблема в том что если надо из props в function Drawer(props) вытащить массив items, а он пустой (например когда в корзину ничего не добавлено) выйдет ошибка, поэтому надо сдедать так что бы при вызове props.items если там пусто то по умолчанию передавался пустой массив. Это делается с помощью деструкторизации и выглядит так

function Drawer({onClose,items = []}) {

Вместо props подставляем то что внутри props т. е. onClose и items и items задаем [ ] по умолчанию, идея в том что в props можно хранить какие-то данные по умолчанию

**Главное на этом этапе создали массив с товарами которые в корзине (или пустой).**

2.19.00Делаем что бы при клик плюс товар добавлялся в корзину.

Идем дальше в App.js есть метод onPlus но сейчас он не вызывается

  onFavorite={() => console.log("Добавили в закладки")}

              onPlus={() => console.log("Нажали плюс")}

            /> Надо перейти в Card в index.js и там с помощью деструкторизации вытащим из props то что там есть т. е. вместо function Card(props) { пишем

function Card({onFavorite,imageUrl,title,price,onPlus}) { и заменяем в Card в index.js props.onFavorite на onFavorite. props.imageUrl на imageUrl и т.д.

В Card у нас есть onPlus в App.js ему задано onPlus={() => console.log("Нажали плюс")}

Сделаем так что бы onPlus срабатывал при нажатии на кнопку + для этого добавим  onPlus();

в const onClickPlus = () => {

    onPlus();

    setIsAded(!isAded); }; внизу у нас есть такой код

<img

          className={styles.plus}

          onClick={onClickPlus}

          src={isAded ? "/img/btn-checked.svg" : "/img/btn-plus.svg"}

        /> получается как сработал onClick запустился onClickPlus а там отработал

сonst onClickPlus = () => {onPlus(); setIsAded(!isAded); };

зто дополнение

в App.js const onAddToCart = () => {alert(123)}; там же  onPlus={onAddToCart}

в index.js onClick={onClickPlus}; жмем на плюс запускаем onClickPlus там же const onClickPlus = () => { onPlus();… запускает ф. OnPlus() в App.js onPlus={onAddToCart}

там же const onAddToCart = () => {alert(123)};

Итого нажиманм на плюс index.js onClick={onClickPlus} --->const onClickPlus = () => { onPlus();…---> App.js onPlus={onAddToCart}--->const onAddToCart = () => {alert(123)};

Зададим в App.js const onAddToCart = () => {alert(123)}; 2.22.25

  return (<div className="wrapper clear"> и получим при нажатии на + alert(123)

Все работает жмем на плюс и у нас срабатывает alert(123)

Но нам надо понять какой из этих товаров добавить в корзину (т. е. какой обьект)

Сделаем так что бы при нажатии на плюс отображались imageUrl,title,price обьекта

Сначало в App.js сделаем так что бы при нажатии на + отображалось слово plus вот так

 onFavorite={() => console.log("Добавили в закладки")}

              onPlus={()=> console.log("plus")} Но нам надо не plus а, imageUrl,title,price

В index.js в onPlus передадим imageUrl,title,price вот так:

const onClickPlus = () => {

    onPlus({imageUrl,title,price});

и вместо plus передадим этот {imageUrl,title,price} обьект в onPlus вот так

   onFavorite={() => console.log("Добавили в закладки")}

              onPlus={(obj)=> console.log(obj)} теперь при нажатии на + получим в консоле imageUrl,title,price обьекта

Этот obj мы получаем из index.js

const onClickPlus = () => {

    onPlus({imageUrl,title,price});Закончил 2.26.30

Дополнение в нашем компоненте например Card(props)

вместо props пишем аргументы которые входит в props т. е.

Card({onFavorite,imageUrl,title,price,onPlus}) и дальше можем эти аргументы использовать

например так const onClickPlus = () => {onPlus({imageUrl,title,price}); здесь мы эти арг. передали в ф. OnPlus нажав на плюс в карточке мы получаем в onPlus{imageUrl,title,price} именно этой карточки.

Теперь obj который в onPlus а именно {imageUrl,title,price}передаем в ф.onAddToCart(obj)

Вот в index.js эта ф.const onAddToCart = (obj) => {console.log(obj)}; она просто выводит obj в консоль 2.28.15

Теперь 2.29.30 нам надо этот обьект передать в cartItems который в хуке

 const [cartItems, setCartItems] = React.useState([]); Делается это так: Вместо

const onAddToCart = (obj) => {console.log(obj)}; пишем  const onAddToCart = (obj) => {

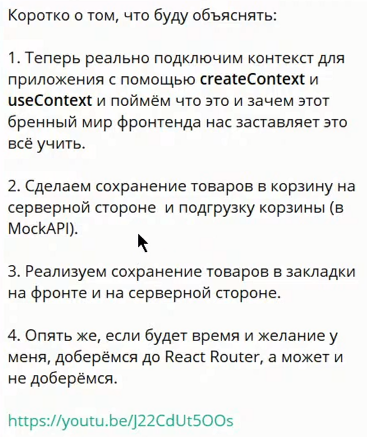
 setCartItems([...cartItems,obj]); }; т. е. передаем в setCartItems что было в ...cartItems и +

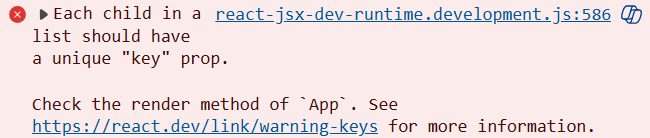
obj т. е. данные еще с одной карточки.

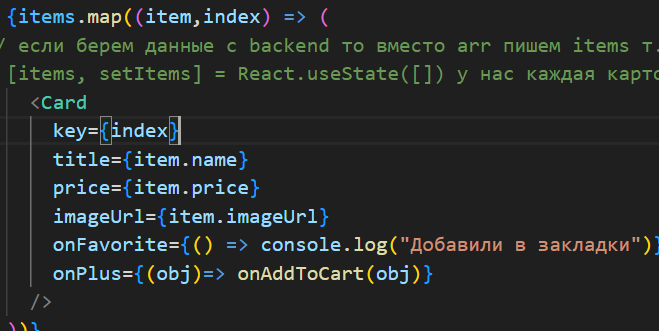
Обьяснение каким способом передаем данные в в корзину (setCartItems) нам надо выводить не только данные объекта на который нажали но и данные обьектов которые были нажаты до этого, в js это делается с помощью команды push примерно так arr ={}-→arr.push(1) = {1} -→arr.push(555) получаем массив {1,555} но в реакте другой способ arr = {555} -→заменись на новый массив куда добавь старый и 777 пишется вот так arr = {...arr, 777} получаем {555,777} см2.32.00 т.е.const onAddToCart = (obj) => {setCartItems([...cartItems,obj]); }; Но лучше написать что предыдущие данные беруться не из ...cartItems а из prev  где prev — это предыдущее состояние, т. е. пишем так const onAddToCart = (obj) => {

    setCartItems(prev => [...prev,obj]);};2.34.30

Переходим к 5-му уроку

 14.00 ,Избавляемся от ошибок в консоле.

 Надо задать каждому элементу который в map рендериться уникальны ключ.Хоть это не совсем правильно в качестве ключа используем index.



28.20 **создаем поиск** создаем хук useState("") но вместо массива ставим строку(" ")

const [searchValue, setSearchValue] = React.useState(""); будем помещаем в searchValue что вводим в input. Input находиться в App.js  <input placeholder="Search.."/>

Вводимые в input данные надо сохранить для этого создадим метод onChangeSearchInput

 const onChangeSearchInput = (event) => { console.log(event.target.value);  };

пока с его помощью просто будем выводить в консоль значение из input (event.target.value).Что бы этот метод срабатывал пишем в input

 <input onChange={onChangeSearchInput} placeholder="Search..." />

когда что то в input изменилось (onChange) срабатывает (onChange) = {onChangeSearchInput}

и выводит в консоле то что в поле input {console.log(event.target.value);}

Но наша задача другая выводить то что есть в inpute, для этого то что вводим в input мы будем вставлять в setSearchValue вот так  const onChangeSearchInput = (event) => {

    setSearchValue(event.target.value);};

Теперь попробуем отобразить данные из input там где нам надо. В нашем случае мы хотим что если в input введем что нибудь, то в заголовке вместо 'Все кроссовки' должно отобразиться `Поиск по запросу: и то что введено в input (event.target.value).

Делаем это так: <h1>{searchValue ? `Поиск по запросу: "${searchValue}"` : 'Все кроссовки'}</h1>

**Обьяснение:** напомнюв searchValue храним то что у нас в поле input. Когда пишем что то в input срабатывает onChange вот здесь <input id="search" name="search"  value={searchValue} onChange={onChangeSearchInput} placeholder="Search..." />

и запускается onChangeSearchInput она передает содержимое в setSearchValue вот так

const onChangeSearchInput = (event) => {setSearchValue(event.target.value);};

и оттуда все это передается в searchValue эти переменные в одном хуке useState. В переменной searchValue появляется что то значит ее значение true и срабатывает первое условие в <h1>{searchValue ? `Поиск по запросу: "${searchValue}"` : 'Все кроссовки'}</h1> т.е вместо 'Все кроссовки' выводиться `Поиск по запросу: и значение того что введено в input "${searchValue}"` так же надо сделать input контролируемый , это значит что когда получаем данные из input эти же данные надо сохранить в этом же input,для этого в input вставляем поле value={searchValue}

**33.10 Будем делать х нажимая на который поле поиска становиться пустым:**

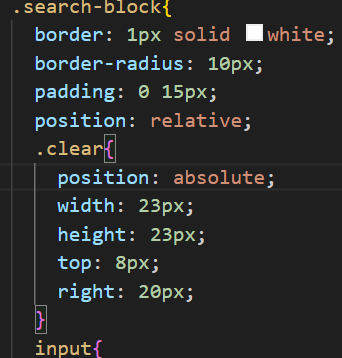
Сначала сделаем Х (крестик). Берем из Drawer.js картинку с крестиком : Корзина{""}

<img onClick={onClose} className="removeBtn cu-p"src="img/btn-remove.svg"  alt="Close"/> Убираем не нужное и вставляем эту картинку в App.js

<div className="search-block d-flex" >

  <img  src="/img/search.svg" alt="Search" />

<img className="clear cu-p" src="img/btn-remove.svg" alt="Clear"/>

даем этому классу имя clear и в index.scss задаем ему разные стили примерно так:

Все это делаем внутри класса .search-block{ задаем размер , ставим в

.search-block relative в clear : absolute и т. д. Получаем справа от input крестик

Теперь сделаем что бы он Х появлялся только когда что то есть в input.

#### Дополнение. Условный рендеринг {searchValue && (...)}Этот синтаксис означает: если searchValue имеет значение (не null, не undefined, не false, не пустая строка "", не 0), то выполняем код внутри скобок. Если searchValue пустая строка ("") или false, то элемент <img> не будет отображаться.

Смысл такой проверяем если в searchValue что то есть т. е. значение true тогда изображается правая часть т. е. появляется X. Делаем этот так:

{searchValue &&(<img className="clear cu-p"src="img/btn-remove.svg" alt="Clear"/>)}

Теперь надо сделать так что бы при нажатии на Х input становился пустым для этого добавим:onClick={()=> setSearchValue("") т. е. что бы при клике на Х setSearchValue становилось пустым ("") вот так: {searchValue &&(<img onClick={()=> setSearchValue("")} className="clear cu-p" src="img/btn-remove.svg" alt="Clear"/>)}

То что мы сделали является конролируемый input это значит что то что находиться в searchValue всегда будет находиться в value input. Например если возьмем и уберем из <input id="search" name="search" value={searchValue} onChange={onChangeSearchInput} placeholder="Search..." />вот эту часть value={searchValue} input станет не контролируемым и при нажатии на X придется вручную очищать то что находиться в input т.к . значение ( value) в input будет прежним, а не примет значение searchValue которое при нажатии устанавливает новое значение переменной searchValue — "" (пустая строка) 39.45

Теперь перед тем как рендерить карточки

 {items.map((item,index) => (

            <Card

              key={index}

              title={item.name}...

С помощью **filter** найдем в **item name** любое содержимое которое есть в **searchValue**

{items

**.filter((item) => item.name?.includes(searchValue))**

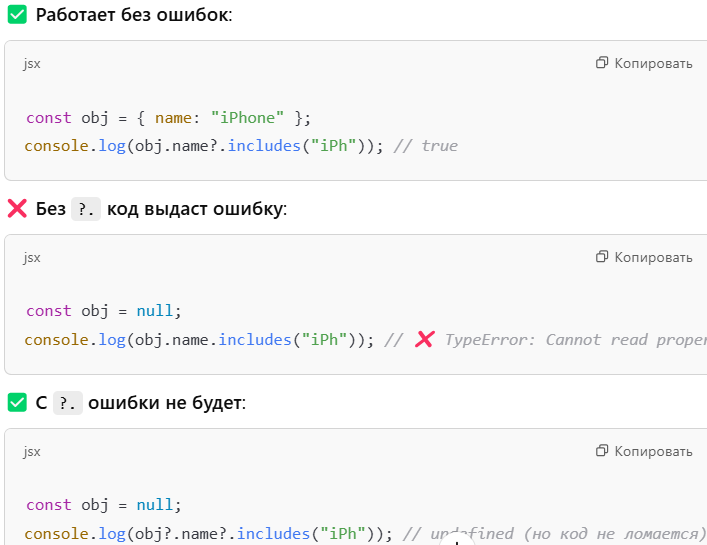
.map((item,index) =>(

<Card

key={index}

title={item.name}

этот код фильтрует массив объектов(item), оставляя только те, у которых свойство name содержит строку searchValue (то что введено в поле input).



**Обьяснение** **Оператор ?**. называется Optional Chaining (опциональная цепочка). Он позволяет безопасно обращаться к свойствам объекта, даже если некоторые из них могут быть undefined или null. **item.name?.includes(searchValue);**

Если item.name существует, выполнится includes(searchValue).

Если item.name не существует (undefined/null), то код не сломается, просто вернёт undefined.

Все ок но надо сделать так что бы поиск не зависел от регистра букв. Для этого при filter надо перевести в нижний регистр item.name(т.е. свойство name обьектов item в которых ищем совпадения) и searchValue (то что мы ввели в input) т. е. приводим в нижний регистр то что мы ищем searchValue и то где мы ищем item.name

Делается это так:{items.filter((item) => item.name?.**toLowerCase()**.includes(searchValue**.toLowerCase()**)) .map((item,index) =>(...

44.00

Теперь сделаем чтобы когда мы нажимали на + товар сохранялся не только в корзине но и на бэкэнде(mockApi).

Товар в корзину мы отправляем с помощью **onAddToCart** const onAddToCart = (obj) => {

setCartItems(prev => [...prev,obj]);}; когда нажимаем + на карточкес в index.js срабатывает onClick он вызывает {onClickPlus} onClick={onClickPlus} от onClickPlus запускается  const onClickPlus = () => { onPlus({imageUrl,title,price});

  //в onClickPlus задаем setIsAded значение true    setIsAded(!isAded); }; //isAded по умолчанию(by default)-false, как кликнули вызвали onClickPlus сработала ф.

    // setIsAded (!isAded) и isAded получила противоположное значение т.е. true опять кликнули isAded стал -false

**от onPlus в App.js в Card** запускается onAddToCart(obj) <Card onPlus={(obj)=> onAddToCart(obj)}/> и тут мы доходим до const onAddToCart = (obj) => {setCartItems(prev => [...prev,obj]);}; и наконец кладем нашу карточку в корзину прибавляя ее к тем которые там уже есть

**А теперь нам все это надо еще отправить на сервак**

Делаем в mockApi новый роут назовем его cart и теперь по этому адресу мы будем хранить товары добавленные в корзину'https://67a5d781510789ef0df96fa9.mockapi.io/cart'

то что на сайте здесь 'https://67a5d781510789ef0df96fa9.mockapi.io/items'

46.40 Получается при нажатии на + кроме добавления товара в корзину визуально через вызов метода onAddToCart(obj) в App.js надо еще паралельно отправить запрос на сервак

Делать это будем с помощью **Axios**

**Дополнение:** библиотека **Axios в React** используется для выполнения HTTP-запросов. Это означает, что с его помощью можно отправлять запросы к серверу, получать данные или отправлять их на сервер, и работать с результатами этих запросов в приложении. Эта библиотека позволяет отправлять такие же запросы как и с помощью fetch если fetch уже встроен в браузур то axios надо отдельно скачивать

Входим в **mockApi** через авторизацию на **GitHab по лог niklebed71 ssKolataval!**

Для начала надо axios прикрутить к нашему проекту пишем в терминале npm install axios дальше делаем import import axios from "axios"; и переделываем fetch запрос

48.30 под axios  React.useEffect(()=>{

    // fetch('https://67a5d781510789ef0df96fa9.mockapi.io/items').then((res)=> {

    //   return res.json();}).then((json)=>{setItems(json);});

 axios.get('https://67a5d781510789ef0df96fa9.mockapi.io/items').then((res)=> {

      setItems(res.data);});

axios отправь **get запрос (запрос на получение данных, список кроссовок**) вот по этому адресу 'https://67a5d781510789ef0df96fa…...items' когда этот запрос отправиться затем .then( возьми ответ сервера (res=> {и передай в setItems(res.data);});)

**res** — это объект, который содержит информацию о результате запроса, **res.data** будет содержать данные, полученные от сервера (в данном случае, **список кроссовок**).  
setItems(res.data) - это функция, которая, используется для обновления состояния в React. Т.е., когда данные получены, они передаются в setItems, чтобы обновить состояние компонента.

51.20 Сделаем что бы при нажатии на + карточка сохранялась не только в корзине визуально но и на бэкзнде в массиве для этого надо наоборот данные передать в наш массив на бэкэнде используем **post запрос (запрос на отправку данных**) делаем это все в методе onAddToCart который вызывается при нажатии на +  const onAddToCart = (obj) =>{

    axios.post('https://67a5d781510789ef0df96fa9.mockapi.io/cart', obj); отправляем на сервер setCartItems(prev => [...prev,obj]); }; отправляем пользователю

По этому адресу 'https://67a5d781510789ef0df96fa9.mockapi.io/cart'(здесь cart а не items) передать obj который возвращает onAddToCart (то что в корзине)  56.00

Вместе с запросом на получение списка кроссовок отправим запрос на получения того что в корзине вот так:React.useEffect(()=>{

     axios.get('https://67a5d781510789ef0df96fa9.mockapi.io/items').then((res)=>{

      setItems(res.data);});

    axios.get('https://67a5d781510789ef0df96fa9.mockapi.io/cart').then((res)=> {

      setCartItems(res.data);}); },[]);

Только заменим в ссылке item на cart и передадим эти данные в setCartItems (т. е. в корзину а не в setItems где все карточки) Теперь даже после обновления страницы все что было в корзине не пропадает **56.50** Все работает

Теперь сделаем удаление **59.10** создаем const onRemoveItem

 const onRemoveItem = (id) => {передаем сюда id карточки которую удаляем

    axios.delete('https://67a5d781510789ef0df96fa9.mockapi.io/cart/${id}');

    setCartItems(prev => [...prev,obj]);//};

здесь используем в запросе метод delete и передаем номер карточки ${id}

**1.00.30** этот метод надо использовать в корзине идем туда {cartOpened && <Drawer items = {cartItems} onClose ={() => setCartOpened(false)} onRemove ={onRemoveItem}/>}

и пишем onRemove = {onRemoveItem}. Передаем onRemove в пропс в Drawer.js

function Drawer({onClose,onRemove,items = []}) { return ( в Drawer.js

на + который на карточке в корзине вешаем onClick  <img onClick={() => onRemove(obj.id)} className="removeBtn" src="img/btn-remove.svg" alt="Remove" />

**Дополнение:** если просто написать в onClick={onRemove}то в onRemove в параметры передастся event который есть у onClick, а нам надо передать obj.id т.е id той карточки которую будем удалять из корзины поэтому передаем нужные параметры с помощью анонимной функции вот так **onClick={() => onRemove(obj.id)} 1.02.00**

Если вставим console.log(id)  const onRemoveItem = (id) => {//удаляет товар из корзины

    console.log(id);

    // axios.delete('https://67a5d781510789ef0df96fa9.mockapi.io/cart/${id}');

    // setCartItems(prev => [...prev,obj]);//};

то в консоле будет отображаться id того элемента на который нажимаем. Теперь без отправки запроса попробуем скрыть тот элемент на + которого кликали.Делаем так:

const onRemoveItem = (id) => {//удаляет товар из корзины

    // console.log(id);

    // axios.delete('https://67a5d781510789ef0df96fa9.mockapi.io/cart/${id}');

     setCartItems((prev) => prev.filter((item) => item.id !== id)); };

когда выполниться onRemoveItem тогда setCartItems возьми что сейчас есть в prev (там массив с кросами которые в корзине) отфильтруй самого себя prev.filter((item) => и проверь что item.id товаров в твоем массиве не равен переменной (id) в onRemoveItem = (id)**. 1.02.50** Метод filter работает так что после фильтрации карточек которые в корзине он вернет только те карточки чей item.id !== id который мы передали при нажатии на + товара в onRemoveItem = (id) т. е. вернет массив со всеми карточками кроме той чей id получил при клике на + карточки, получается карточка с этим id удалилась.**1.04.30**

Теперь украсим корзину вставив в нее текст и картинку идем в Drawer.js и пишем там такой код Корзина{""} <img onClick={onClose} className="removeBtn cu-p"

            src="img/btn-remove.svg"  alt="Close"/> </h2>

        <div class="cartEmpty d-flex align-center justify-center flex-column flex">

          <img class = "mb-20" width = "120px" height = "120px"  src="/img/empty-cart.jpg" alt="CartEmpty"/>

          <h2>Корзина пустая</h2>

          <p className="opacity-6">Добавьте хотя бы одну пару кроссовок, что бы сделать заказ</p>

          <button clas="greenButton">

            <img src="/img/arrow.svg" alt="Arrow" />Вернуться назад

          </button>

        </div>  здесь ничего необычного добавили картинку надпись и кнопку

        <div className="items">

          {items.map((obj) => (<div className="cartItem d-flex align-center mb-20">

**Попробовал стилизовать корзину а именно class="cartEmpty**

**добавил в index.scss такие стили**

**.cartEmpty {**text-align: center;

      p {width: 280px;line-height: 24px;}

      .greenButton { width: 245px;margin-top: 20px;

        &:hover {img {transform: rotate(180deg) translateX(3px);}}

        img {position: relative;top: 1px; transform: rotate(180deg);margin-right: 15px;

          transition: transform 0.15s ease-in-out; } }} **1.06.30 почемуто не все работает**

1.7.10 теперь сделаем так, если товаров в корзине нет то отображалась пустая коробка, а если есть то корзина с товарами. Для этого в Driver.js пишем

 {

        items.length > 0 ?// если корз пустая выводи cartEmpty если нет items и cartTotalBlock

        <div>  важно если у нас выводиться два блока то их надо обьединить например в один

        <div className="items">….</div> div как сделано здесь

        <div className="cartTotalBlock">…. </div>

        </div>

        :

        <div class="cartEmpty d-flex align-center justify-center flex-column flex">…..</div>

        } 1.10.20

Сделаем что бы при нажатии на кнопку вернуться назад корзина закрывалась, находим в Driver.js кнопку и вешаем onClick при котором срабатывает = {onClose}

 <button onClick = {onClose} class="greenButton">

            <img src="/img/arrow.svg" alt="Arrow" />Вернуться назад </button>

{onClose} мы для Driver.js в App.js задали и передали пропсом в Driver.js

1.11.20Теперь будем делать так что бы при отчистке товар очищался не только на сайте но и с на бэкэнде для этого при удалении товара с корзины надо одновременно посылать запрос на удаление товара с сервака все делается в App.js вот здесь const onRemoveItem = (id) => {

    axios.delete('https://67a5d781510789ef0df96fa9.mockapi.io/cart/${id}');

    setCartItems((prev) => prev.filter((item) => item.id !== id )); };

Сохранение в корзине сделали делаем сохранение закладок **1.14 Переходим в Card / index.js**

**создадим хук const [isFavorite, setIsFavorite] = React.useState(false); по умолчанию isFavorit-false**

**расписываем что бы при изменении значения isFavorite показывалась одна из 2-х img**

**<div className={styles.favorite} onClick={onClickFavorite}>**

        <img src= {isFavorite ? "/img/liked.svg" : "/img/unliked.svg"} alt="Unliked" />

      </div>

**создаем  const onClickFavorite = () => {**setIsFavorite(!isFavorite);};с ее помощью можем менять текущее состояние isFavorite для отображения одной **из 2-х img**

**return (при onClick на картинку сердечка вызываем onClickFavorite**

    <div className={styles.card}>

      <div className={styles.favorite} onClick={onClickFavorite}>

        <img src= {isFavorite ? "/img/liked.svg" : "/img/unliked.svg"} alt="Unliked" />

**1.17.20 Идем** в App.js **и с**оздаем массив закладок для Favorites:

const [Favorites,setFavorites] = React.useState([]);

потом создаем новый метод в App.js: const onAddToFavorite = (obj) =>{

  axios.post('https://67f584ec913986b16fa4db84.mockapi.io/favorites', obj); передаем инфу на сервер  setFavorites(prev => [...prev,obj]); }; отображаем инфу на сайте Этот метод надо вызывать при каждом нажатии на кнопку для этого пишем <Card...

onFavorite={(obj) => onAddToFavorite(obj)}...

Загвоздка в том что отображать Favorites надо на отдельной странице

Еще проблема в том что в бесплатной версии на mockApi можем создать только 2 resourse поэтому заходим под другим аккаунтом kolakola1219@gmail.com в mockApi и создаем новый проект его имя Favorites там делаем новый resourse под названием favorites и получаем на него ссылку. https://681880cb5a4b07b9d1cf6ac7.mockapi.io/favorites

до 1.35.10 делалось сокрытие адреса в mockApiпродолжить с 1 36 10

Далее нажимая на сердечко в index.js работает onClick= дальше{onClickFavorites

const onClickFavorites = () => {

    onFavorites({imageUrl,title,price}); это передается в function Card({onFavorites,imageUrl,title,price,onPlus} отсюда в App.js в

 <Card...

onFavorites={(obj) => onAddToFavorite(obj)}...

и обьект передаем на сервер и на сайт const onAddToFavorites = (obj) =>{

  axios.post(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/favorites`, obj);

  setFavorites(prev => [...prev,obj]); }; продолжить с 1 41 10

Понимаем что что onFavorites создается в App.js в Card

<Card…. Вот оно  onFavorites={(obj) => onAddToFavorites(obj)}…. /> ))}

для того что бы onFavorites в App.js был связан с index.js его надо засунуть туда с помощью пропсов function Card({onFavorites,….onPlus}) {

В Card в index.js кликаем onClick={onClickFavorites}> дальше

const onClickFavorites = () => {

 вызывается   onFavorites({imageUrl,title,price});…….. и она передается в function Card({onFavorites………..})Теперь в Card в index.js у нас onFavorites( с {imageUrl,title,price} той карточки на которую кликнули

и дальше уже в App.js когда вызываем Card в onFavorites у нас obj в котором {imageUrl,title,price} из той карточки на которую кликнули в Card в index.js ну а дальше от onFavorites={(obj) срабатывает => onAddToFavorites с данными той же карточки (obj)} и дальше все передается

const onAddToFavorites = (obj) => в

{axios.post(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/favorites`, obj);

  setFavorites(prev => [...prev,obj]); };

В Card.js функции onClickFavorites и onClickPlus вызывают onFavorites(...) и onPlus(...).  
Эти функции передаются из App.js как пропсы и обёрнуты вокруг onAddToFavorites(...) и onAddToCart(...).  
То есть клики в Card вызовут функции onAddToFavorites(...) и onAddToCart(...). в App.js, куда ты уже можешь писать свою основную логику — добавление в массив, localStorage, отправка на сервер и т.д.

делаем  onPlus={(obj)=> onAddToCart(obj)} в Арр при его вызове будет осуществляться какая нить логика {(obj)=> onAddToCart(obj)} она прописана здесь

  const onAddToCart = (obj) =>{

    axios.post('https://67a5d781510789ef0df96fa9.mockapi.io/cart', obj);

    setCartItems(prev => [...prev,obj]); };

Что бы запускать onPlus из компонента (index.js) надо его передать в компонент это делают с помощью пропсов function Card({onFavorites,imageUrl,..onPlus}) {

Теперь в index.js пишем логику что и когда передавать в пропс например в нашем случае все запускается с нажатия на плюс (цель хотим добавить карточку в корзину)

пишес событие onClick= и создаем {onClickPlus} в которой мы и пишем то что сделать при клике const onClickPlus = () =>

{onPlus({imageUrl,title,price}); setIsAded(!isAded); };

здесь передаем в onPlus данные карточки ({imageUrl,title,price});

onPlus с {imageUrl,title,price} передается в пропс который в Card вот здесь

function Card({onFavorites,imageUrl,..onPlus}) и он c пропса передается в App.js

в App.js

<Card

key={index}

title={item.name}

price={item.price}

imageUrl={item.imageUrl}

onFavorites={(obj) => onAddToFavorites(obj)}

onPlus={(obj) => onAddToCart(obj)}

/>

Значит, при вызове onFavorites внутри Card, фактически вызывается:

onAddToFavorites(obj)

А при onPlus — вызывается - onAddToCart(obj)

**1.44.00 React Router начинаем развлекаться**.Она нужна что бы делать Быстрый переход по страницам без перезагрузки приложения Устанавливаем в терминале эту библиотеку

npm i react-router-dom. Идем в index.js и подключаем import { BrowserRouter as Router } from 'react-router-dom';

Там же оборачиваю App.js в Router <React.StrictMode>

    <Router>

    <App />

    </Router>

  </React.StrictMode>

Сделаем так что бы App.js рендерил данные в зависимости от того что указано в адресной строке сначала импортируем в App.js – import Route from 'react-router-dom'; Продолжить 1.50

Как использовать React Router:

Подключаем в App.js import {Routes,Route} from 'react-router-dom';

После Header пишем  <Header onClickCart ={() => setCartOpened(true)} />

<Routes> (оболочка для Route без нее Route не работает )

<Route path="/favorites" element={<p>This is test</p>}/> (в path адрес при котором

</Routes> (будет открываться то что написано в element)

121112222 ( это просто рандомный текст)

      <div className="content p-40">

Если в адресной строке <http://localhost:3000/> то на экране  

This is test нет

Если в адресной строке http://localhost:3000/favorites то на экране  This is test появляется.

Таким способом будем использовать компонент Route.

Писать в адресной строке имя path(favorites) не удобно поэтому будем использовать компонент Link что бы то что в Route появлялось когда нажмем на фигню к которой подключен Link (в нашем случае подключим Link к сердечку которое в Header) 1.52.00

Идем в Header.js подключаем Link import {Link} from 'react-router-dom'

Берем картинку сердечка и оборачиваем в Link

 <Link to="/favorites"> (пишем куда ведет link здесь на /favorites

<img width = {18} heigth = {18} src = "/img/heartz.svg" alt="Закладки" /> </Link>

Теперь нажав на сердечко попадаем на страничку favorites(это страничка где запись This is test) Для закрепления сделаем переход с http://localhost:3000/favorites при нажатии на логотип на главную страницу [http://localhost:3000](http://localhost:3000/). Оборачиваем в Header.js в Link логотип

<header className="d-flex justify-between align-center p-40">

<Link to="/">

<div className="d-flex align-center">

  <img width = {40} heigth = {40} src = "/img/logo.png" />

  <div>

    <h3  className="text-uppercase">React Sneakers</h3>

    <p className="opacity-5">Магазин лучших кроссовок</p>

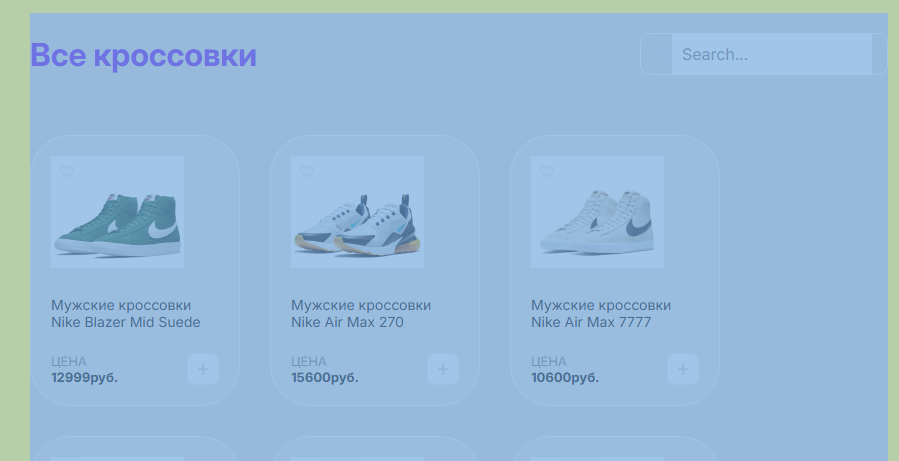
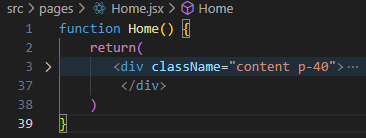
  </div>

</div>

</Link>

<ul className="d-flex">

Теперь если находимся на http://localhost:3000/favorites будем переходить на [http://localhost:3000](http://localhost:3000/) важно то что это происходит быстро без перезагрузки страницы 2.00.00

Теперь используя это мы можем в зависимости от события (например клик по иконке) показывать или скрывать разные блоки сайта. Для этого надо выделить эти блоки, сделаем их отдельными компонентами. По сути это будут как отдельные страницы, кликаем по сердечку скрывается блок с карточками кроссовок заголовком и поиском и открывается другой заголовок и набор карточек который в favorites на сайте выглядит как открытие новой страницы. Создадим отдельную папку pages и там файл Home.jsx (как понимаю создаем jsx т.к там проще писать) туда перенесем блок с карточками кроссовок заголовком и поиск  Т

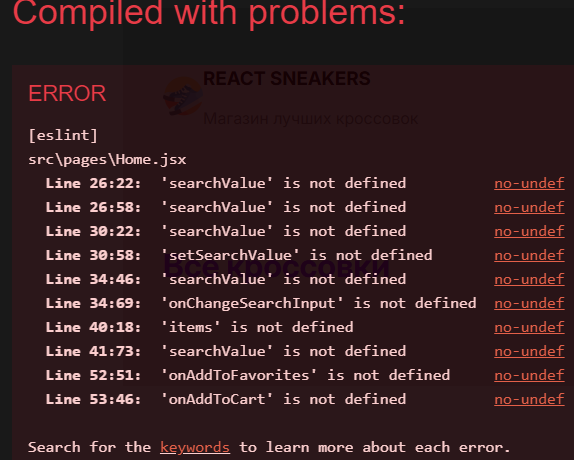
Теперь эта часть у нас не отображается.Т.к. card после переноса в home.jsx теперь должна рендериться в home.jsx туда же из App.js перенесем import Card from "./components/Card";

Но изменим путь import Card from "../components/Card"; поставим еще точку т.к. нам надо не на одну а уже на две ступени из файла home.jsx подняться что бы выйти на Card. Так же в home.jsx надо подставить export default Home;

App.js пишем import Home from "./pages/Home"; и рендерим наш компонент Home когда в адресной строке / (код наисан немного по другому чем в уроке,там старая версия)

<Routes> <Route path="/" exact element={<Home/>}/> </Routes>

Если запустим код то выдаст ошибки т.к. в Home нет всех этих методов но они есть в App.js.

Можно их тупо перенести в Home но обычно используют технологию

**Contex** 2.07.50 для этого в App.js в Home вставляем недостающие методы <Routes>

        <Route path="/" exact element={<Home

         items={items}

         onChangeSearchInput={onChangeSearchInput}

         searchValue={searchValue}

         setSearchValue={setSearchValue}

         onAddToCart={onAddToCart}

         onAddToFavorites={onAddToFavorites}/>}/>

</Routes>

Для того что бы все это передалось в Home.jsx надо там написать это же пропсами

functionHome( {items,searchValue,setSearchValue,onChangeSearchInput,

onAddToFavorites,onAddToCart, }){

2.09.50 Теперь в компонент Home.jsx который по сути является страницей, с помощью пропсов передаем из App.js все что ему надо <Route path="/" exact element={<Home.. Home.jsx из пропсов вытаскивает все что ему передали и рендерит(открывает) страницу Home.jsx с теми данными которые он получил в пропсы из App.js.

По адресу http://localhost:3000/ у нас теперь все что написанно в App.js плюс через Route подключенная страница Home.jsx

Если мы перейдем <http://localhost:3000/favorites> нажав на сердечко карточек с кросами не будет и ничего кроме header не будет. Но нам надо что бы вместо карточек с кроссами отображалась страница с теми кроссами что добавлены в [favorites](http://localhost:3000/favorites). Для этого копируем Home.jsx меняем название на Favorites.jsx делаем такие изменения function Favorites(){ return (

<div className="content p-40">

            <div className="d-flex align-center justify-between mb-40">

                <h1>Мои закладки</h1>

            </div>

            <div className="d-flex flex-wrap">Тут будут мои закладки </div>

</div>);}

export default Favorites;

Хотя создали новую страницу но надо сделать что бы она подключалась при нажатии на сердечко. Для этого идем в App.js и пишем в <Routes>

        <Route path="/" exact element={<Home

         items={items}….../>}/>

         <Route path="/favorites" exact element={<Favorites/>}/>

</Routes>

Что бы кроме рендеринга(отображения) страницы Home если адрес страницы Route path="/" при адресе стр. path="/favorites" отображалась стр. Favorites

Но для этого надо эту страницу импортировать в App.js т. к. она у нас в той же папке что и Home.jsx адрес такой import Favorites from "./pages/Favorites";

2.14.00 Теперь нам надо сделать что бы на нашей странице Favorites отображались

те закладки которые кликнув на сердечко мы туда положили.

идем в React.useEffect( и добавляем ссылку на сервер(mockapi) где храним favorites  axios.get(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart`).then((res)=> {

      setFavorites(res.data);});

при первом рендереннге useEffect с mockapi получаем карточки в setFavorites

Затем в App.js в Route со ссылкой на стр Favorites пишем

<Route path="/favorites" exact element={<Favorites items = {favorites}/>}/>

и передаем элементы которые у нас в favorites на нашу стр. Favorites

На стр. Favorites там где надпись Тут будут мои закладки подставляем рендеринг items и Card из Home т. к. карточки в favorites по дизайну точно такие как в Home. Вот так:

 <div className="d-flex flex-wrap">

            {items уберем фильтрацию onFavorites и onPlus

 // .filter((item) => item.name?.toLowerCase().includes(searchValue.toLowerCase()))

.map((item, index) => ( <Card key={index} title={item.name} price={item.price}

imageUrl={item.imageUrl} //onFavorites={(obj) => onAddToFavorites(obj)} // onPlus={(obj) => onAddToCart(obj)}/> ))} Еще надо в пропс в Favorites добавить items вот так function Favorites({items}){ и импортировать в Favorites компонент Card вот так :

import Card from "../components/Card"; 2.16.00

Все ок но у карточек в Favorites нет состояния что они добавленны в корзину т. е. надо что бы сердечки были подсвечены (true) в Card создаем новое с-во favorited

function Card({onFavorites,imageUrl,title,price,onPlus,favorited}) {

передадим его на стр. Favorites.jsx в Card со значением true

<Card key={index} title={item.name} price={item.price} imageUrl={item.imageUrl}

   favorited={true}...если favorited={true} в карточках в Favorites сердечки будут подсвечены. Берем эту переменную favorited значение которой true и в index.js пишем ее в хук вместо значения false const [isFavorites, setIsFavorites] = React.useState(было false стало favorited); В карточки (Card) на стр. Favorites.jsx передали с-во favorited со значением {true} и сердечки подсвечены (true)

В index.js в Card для того что бы в const [isFavorites, setIsFavorites] = React.useState(favorited);было всегда булевое значение передадим для favorited = false functionCard({onFavorites,imageUrl,title,price,onPlus,favorited = false}) если этого не сделать то при первом рендеренге если в стр.Favorites не будет передана ни одна Card то значение const [isFavorites,setIsFavorites]=React.useState(favorited);

будет undefined хотя надо что бы оно было булевое 2.19.00

Если сейчас нажмем в закладках( Favorites) на карточке на сердечко получим ошибку что у нас нет такого метода здесь, т. к. он храниться в App.js передадим его в Favorites.jsx для этого в App.js в <Route path="/favorites" exact element={<Favorites items = {favorites} добавим onAddToFavorites={onAddToFavorites} и передадим его в Favorites.jsx/>}/>

</Routes>

В Favorites.jsx function Favorites({items, добавим этот метод в пропс onAddToFavorites}){ а уже из пропса передадим его в Card (которая в Favorites.jsx)

  <Card key={index} title={item.name}...onFavorites={onAddToFavorites} 02.20.00

Теперь когда мы нажмем на сердечко в избранном (Favorites) у нас появиться еще одна карточка но без подсвеченного сердечка было стало

в данный момент при нажатии на сердечко <Route path="/favorites" exact element={<Favorites items = {favorites} onAddToFavorites={onAddToFavorites}/>}/>

</Routes> срабатывает {onAddToFavorites}и

отправляет запрос на добавление карточки const onAddToFavorites = (obj) =>{

  axios.post(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart`, obj);

  setFavorites(prev => [...prev,obj]); }; и у нас добавляется еще одна такая же карточка.

Это происходит т. к. мы не сделали проверку есть ли эта карточкка в избранном если она есть то не надо добавлять если нет то добавляй. Можно сделать проверку по Id но у нас нет Id поэтому в mockApi сделаем Id для карточек для этого пишем

в начале карточки"id": 1, и дальше всем карточкам будет присваиваться "id" 2.22.00

Теперь нам надо делать проверку если в favorites найду обьект с таким же id то тогда делать delete на mockApi так же его надо удалить из setFavorites массив закладок для Favorites

if(favorites.find(obj=>obj.id === id)) { ищем одинаковый id

  axios.delete(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart/${obj.id}`);}

setFavorites((prev) => prev.filter((item) => item.id !== obj.id ));

если такого id не будет тогда создай карточку и пошли на mockApi и на сайт else{

  axios.post(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart`, obj);

  setFavorites((prev) => [...prev,obj]); } 2.23.50

Прикол в том что при нажатии на сердечко id у нас не передается потому что если зайти в

Card в  const onClickFavorites = () => {onFavorites({imageUrl,title,price});

    setIsFavorites(!isFavorites);};мы не передаем здесь id, но надо передать и id

Получается нажимая запускаем ф.onClickFavorites она передает imageUrl,title,price

но не id, добавим id onFavorites({id,imageUrl,title,price}); надо передать все также в Favorites.jsx Card можно просто передать id = {item.id} но можно сразу передать все с-ва хранящиеся в obj написав {...item} соответственно можно убрать оставшиеся которые уже есть в items например такие (невыделенные)

<Card key={index} title={item.name} price={item.price} imageUrl={item.imageUrl}favorited={true} onFavorites={onAddToFavorites} {...item}/>))} все с-ва которые есть в этом item будут передаваться в Card. Еще добавим в index.js function Card({id,onFavorites,imageUrl...и теперь из Card можно вытаскивать id и его передать сюда const onClickFavorites =()=>{onFavorites({id,imageUrl,title,price});2.25.30

Теперь перейдем в Home.jsx и сделаем тоже самое передадим все с-ва item {...item}

<Card key={index} title={item.name} price={item.price} imageUrl={item.imageUrl}  onFavorites={(obj) => onAddToFavorites(obj)} onPlus={(obj) => onAddToCart(obj)}

{...item}/>))} 2.27.00Теперь мы получаем item id и можем делать фильтрацию или проверку по id все это мы написали раньше в App.js в const onAddToFavorites=(obj)=>{…

Фильтрацию делаем так:2.27.50

**const onAddToFavorites = (obj) => {**

    if (favorites.find((favObj) => favObj.id === obj.id)) {Если в favorites найдется такой id как и в том обьекте что я нажал при клике favObj.id === obj.id

  await axios.delete(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart/${obj.id}`);

отправить запрос на удаление в бэкенд,потом в setFavorites профильтруй все item

      setFavorites((prev) => prev.filter((item) => item.id !== obj.id));у которых item.id !== obj.id это уберет все obj. у которых item id не= obj.id

} else { const res = await если в favorites obj с таким id не нашелся

axios.post('https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart', obj);отправь запрос

setFavorites((prev)=>[..prev, res.data]);}};на создание и в useState сохрани этот обьект

2.28.35

Давай сделаем так что бы при удалении обьекта из Favorites он удалялся только с бэкэнда но оставался на сайте, ПО КРАЙНЕЙ МЕРЕ ДО ПЕРЕЗАГРУЗКИ это удобно т. к. человек может по ошибке удалить карточку, а так до перезагрузки только сердечко будет не активным.

Для этого уберем  setFavorites((prev) => prev.filter((item) => item.id !== obj.id));

Возникла проблема при добавлении obj в favorites его id будет равен порядковому номеру добавления его в favorites а, в useState у него id как в useState т. е. не совпадают. Что бы решить эту проблем надо при добавлении obj в favorites дожидаться ответа от бэкэнда и только потом его обновлять(массив favorites). Для этого используем **конструкцию async await** это тоже что конструкция them например здесь App.js

axios.getполучи res('https://67a5d781510789ef0df96fa9.mockapi.io/items'). C этого адреса затем then((res)=> { вытащи res.data и верни их в setItems(res.data);

В нашем случае делаем так 2.32.55

Обьяснение

**конструкция try catch 2.38.30**

желательно обернуть все в конструкцию **try catch** это нужно для того что бы когда используется конструкция async await можно было бы отловить ошибку при вызове запроса else {const res = await axios.post('https://68… если делать async await без **try catch** не узнаем когда ошибка произойдет await возвращает в консоль неотлавлеваемую ошибку.

const onAddToFavorites = async (obj) => {

  try {

    if (favorites.find((favObj) => favObj.id === obj.id)) {

await axios.delete(`https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart/${obj.id}`);

      setFavorites((prev) => prev.filter((item) => item.id !== obj.id));}

else { const res = await axios.post('https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart', obj);

      setFavorites((prev) => [...prev, res.data]); id  }

  } catch (error) {

    console.error("Ошибка при работе с избранным:", error); } };

Говорим try (попробуй выполнить этот код if (favorites.fi…...) если что то в этом коде сломается отлови эту ошибку catch (error) { и предупреди об этом выведя в консоль

    console.error("Ошибка при работе с избранным:", error)

**Дополнение по mockApi** боролся с mockApi можно создать бесплатно только два ресурса на проекте у меня уже были cart https://67a5d781510789ef0df96fa9.mockapi.io/cart

и items <https://67a5d781510789ef0df96fa9.mockapi.io/items>пришлось на другом аккаунте [kolakola1219@gmail.com](mailto:kolakola1219@gmail.com) sstaval зайти в mockApi и сделать новый проект и создать ресурс heart https://681880cb5a4b07b9d1cf6ac7.mockapi.io/heart здесь храним **закладки.**

**Начинаем урок 6 до 0.27.30 ничего с этого момента начинаем исправлять баги.**

**Создал бекап на 20.05 пока не стал исправлять баги в 6 уроке.**

Нажимаем на карточке на плюс товар в корзине плюс зеленый, еще раз нажимаем на плюс (по идее товар должен пропасть с корзины) карточка еще раз добавляется в корзину.

Закончил 0.57.00 в cart нет id надо на mockApi сделать.

До 1.20 юзаем React skeleton прога <https://skeletonreact.com/> работает плохо поэтому код для скелетонов и сами скелетоны убогие(потом доработать в этой или другой программе) с 1.20 встраиваем скелетоны в код

Закончил на 1.23

1.28.30 делаем так что бы сначало скелетон затем картинка

**Закончил на 2.45.00**

1.57.00 const appContex = React.createContext({});//ф.реакта если изменения в обьекте то оповестит console.log(appContex);//все другие компоненты которые зависят от этого обьекта function App() { Закончил на 2.15

До 2.45 очень мудрено и запутано, с 2.45 начинаем делать компоненты info.jsx заказ оформлен спеасибо за заказ и т. д. Закончил на 3.15 Дальше ничего интересного

Урок 7 До 0.17.30 можно не смотреть.

0.18.30 делаем счетчик суммы в header до 0.24.30

c 0.24.30 делаем счетчик в корзине до 0.28.44

Создаем кастомные хуки 1.31.30

Список заказов 1.43.20

2.21.30 Заливаем на gitHabPages (есть видос деплой реакт приложение)

2.46.00 Обьясняет почему картинки не показыаются на gitHabPages и что сделать

3.20.10 Где исходники найти.

**Как скачать с GitHab на VS Code** и вернуть обратно проект с изменениями:

Открываем терминал и переходим в папку cd путь\_до\_папки (cd C:\Users\Kola\Desktop)

Потом git clone <https://github.com/username/repository.git> (Замените https://github.com/username/repository.git ссылкой на ваш репозиторий) После завершения клонирования откройте проект в VS Code одним из следующих способов:

**Из интерфейса**: Нажмите File > Open Folder, выберите директорию вашего проекта.

**Через командную строку**: Запустив терминал в нужной директории, введите: code .  
Затем запускаем проект npm start наверняка получим ошибку что npm не установлен и надо будет npm install ввести, а дальше npm start.

Теперь внеся изменения в проект передадим его обратно в gitHab, открываем терминал и пишем git add .( PS C:\Users\Kola\Desktop\git\sneakers0425> git add .)

Добавляем коммит git commit -m "Ваше описание изменений"(PS C:\Users\Kola\Desktop\git\sneakers0425> git commit -m "Создал файл gitGit"

Отправляем изменения на удалённый репозиторий: git push -u origin main

**Если надо переключиться между ветками**

git fetch origin # Обновляем список ветвей с сервера

git checkout название\_ветки # Переключаемся на выбранную ветвь

Например, если нужно перейти на ветку develop, выполняем:

git fetch origin

git checkout develop

**Создание новой ветки**

Чтобы создать новую ветку проекта в VS Code и перенести её на GitHub, выполните следующие шаг

Создание новой ветки локально

Откройте терминал в VS Code :git checkout -b <название\_ветки>

Например: git checkout -b feature-new-component

Это создаст новую ветку и переключит вас на неё.

Сделайте изменения в файлах и подтвердите их в терминале: git add .

git commit -m "<сообщение>" Пример:git add . git commit -m "Добавлен новый компонент"Пуш новой ветки на удалённый репозиторий

Отправьте вашу новую ветку на GitHub: git push origin <название\_ветки>

Например: git push origin feature-new-component