Документация по проекту Fashionee.

Содержание

[Введение 2](#_Toc194442379)

[1. Общие компоненты (common components) 3](#_Toc194442380)

[1.1 Header (шапка страницы) 3](#_Toc194442381)

[1.2 Content-Block (блок с контентом) 3](#_Toc194442382)

[1.3 Footer («подвал» страницы) 3](#_Toc194442383)

[2. UI компоненты (ui) 3](#_Toc194442384)

[2.1 NavLink (навиагционная ссылка) 3](#_Toc194442385)

[2.2 Socials (блок с ссылками на социальные сети) 4](#_Toc194442386)

[3. Showcase (основное содержимое страницы Shop) 4](#_Toc194442387)

[3.1 SideBar (боковое меню) 4](#_Toc194442388)

[3.1.1 Filters 4](#_Toc194442389)

[3.1.2 LastViewed 5](#_Toc194442390)

[3.2 Products (основной блок страницы) 5](#_Toc194442391)

[4. Helpers (вспомогательные функции) 5](#_Toc194442392)

[4.1 getFilters 5](#_Toc194442393)

[4.2 randomProducts 6](#_Toc194442394)

# Введение

# Общие компоненты (common components)

## Header (шапка страницы)

Компонент Header является шапкой страницы. В нем содержатся такие элементы как:

* Бургер – меню;
* Логотип сайта;
* Навигация между страницами сайта (текстовое меню). Для него используется UI компонент NavLink;
* Навигация между страницами профиля (меню с иконками). Для него создан компонент внутри Header с названием HeaderIconMenu;

## Content-Block (блок с контентом)

Компонент Content-Block представляет собой блок с контентом, содержащим в себе:

* название страницы, на которой находится пользователь;
* две навигационные ссылки для переключения между страницами. Используется UI компонент NavLink;
* баннер с изображением.

## Footer («подвал» страницы)

Компонент Footer представляет собой блок с информацией необходимой пользователю, который размещается внизу страницы. В нем содержатся такие элементы как:

* Логотип и описание компании;
* Ссылки на социальные сети компании. Используется компонент Socials;
* Блок About, содержащий ссылки на информацию о компании и продукции. Используется компонент NavLink;
* Блок Useful Links, содержащий ссылки на информацию о поддержке, правилах использования и т.д. продукции. Используется компонент NavLink;
* Уведомление об авторском праве;
* Возможные способы оплаты;

# UI компоненты (ui)

## NavLink (навиагционная ссылка)

NavLink – «глупый», статичный компонент, который получает необходимые данные и отображает их на странице. Данный компонент получает такую информацию как:

* className – имя класса для стилизации;
* href – ссылка на страницу;
* text – текст для отображения ссылки на странице;
* img – иконка для навигационной ссылки (если таковая имеется)
  1. Socials (блок с ссылками на социальные сети)

Socials – так же статичный компонент, в который вынесено оформление блока с ссылками на социальные сети для того, что бы не перегружать визуально версткой компонент Footer.

# Showcase (основное содержимое страницы Shop)

Showcase – компонент, в котором находится содержимое страницы Shop, разделенное в свою очередь еще на 2 компонента:

* SideBar – боковое меню с поиском, фильтрами и т.д.;
* Products – основной блок с карточками товара, сортировкой и пагинацией.

Передает в [Products](#_Products_(основной_блок) следующие props:

<Products

       products={products}

       changeSortType={changeSortType}

       sortType={sortType}

      />

В качестве products={products} используется состояние, в котором записывается значение всего массива продуктов:

const [products, setProducts] = useState(data.products);

В качестве sortType={sortType} используется состояние, в котором для компонента Sort хранится начальное значение выпадающего списка сортировки:

const [sortType, setSortType] = useState('priceAsk');

В качестве changeSortType={changeSortType} используется функция, меняющая значение сортировки:

const changeSortType = (type) => setSortType(type);

## SideBar (боковое меню)

SideBar – один из компонентов страницы Shop. Содержит в себе следующие компоненты:

* Search – поиск товара;
* Filters – различные фильтры для поиска товара;
* LastViewed – последние просмотренные товары;

### Filters

Компонент Filters представляет собой общий сборный компонент для трех разных компонентов фильтров:

* CategoryFilter (женские, мужские и тд; список)
* PriceFilter (от минимального к максимальному; два инпута)
* ColorFilter (по цвету; чекбоксы)

В Filters вызывается функция **getFilters,** собирающая значения для фильтров, результат ее работы передается в каждый отдельный фильтр.

Компоненты фильтров с помощью infoForFilters переданным им, отрисовывают списки и значения фильтров.

### LastViewed

Компонент LastViewed представляет собой блок, где по логике должны выводиться три последних просмотренных товара, но в данном проекте для упрощения используется вывод трех рандомных товаров. Рандомные товара для вывода получаются с помощью вспомогательной функции randomProducts

## Products (основной блок страницы)

Products – основной блок страницы Shop. Содержит в себе следующие компоненты:

* Sort – компонент с сортировкой товара;
* Product – компонент карточки товара;
* Pagination – переключение между страницами;

Компонент Products получает следующие props из родительского [Showcase](#_Showcase_(основное_содержимое):

export function Products ({products, changeSortType, sortType}) {

Так же в этом компоненте реализована логика добавления и удаления товара из избранного.

1. Создается состояние, необходимое для хранения массива товаров находящихся в LocalStorage (далее LS). Это нужно, что бы при добавлении товара в избранное в компоненте Product моментально корректно отображался цвет «сердца» (индикатор нахождения товара в избранном):

  const [productsInLS, setProductsInLS] = useState([]);

1. Логика добавления и удаления товара срабатывает на нажатие кнопки «сердце» на карточке товара. Логика находится в функции, которая входным параметром получает id товара, на котором произошло нажатие:

const favoriteActions = (productId) => {

1. Сначала в функции получается товар, уже находящийся в избранном в LS:

const fromLS = localStorage.getItem(favoritesKey);

1. Далее идет проверка. Если товара, на который нажали, еще нет в избранном в LS, то тогда в LS передается пустой массив (который далее и будет наполняться) и первый для этого массива id товара. Также сразу обновляется состояние useState setProductsInLS:

if (!fromLS) {

      setProductsInLS([productId]);

      localStorage.setItem(favoritesKey, JSON.stringify([productId]));

      return;

    }

1. Для удаления или добавления второго и более товара преобразуем полученные в виде строки товары fromLS в массив для работы и проверяем, есть ли товар в LS:

  const products = JSON.parse(fromLS);

  const inLS = products.find((id) => id === productId);

1. Если товар в избранном уже есть, то повторное нажатие на «сердце» означает намерение пользователя удалить товар из избранного. Для удаления фильтруется массив товаров – все товары, id которых не равен id товара, по которому произошло повторное нажатие, записываются в новый массив. Так же происходит обновление актуального списка избранного путем передачи нового отфильтрованного массива в setProductsInLS и в сам LS:

    if(inLS) {

      const filteredProducts = products.filter((id) => id !== productId);

      setProductsInLS(filteredProducts);

      localStorage.setItem(favoritesKey, JSON.stringify(filteredProducts));

      return;

    }

1. Если же товара в избранном нет, то происходит его добавление в массив избранного. И соответственно обновляется setProductsInLS и LS:

    products.push(productId);

    setProductsInLS(products);

    localStorage.setItem(favoritesKey, JSON.stringify(products));

Для того, чтобы при перезагрузке страницы у товаров, уже находящихся в избранном, корректно отображался цвет кнопки «сердце», используется useEffect, срабатывающий при монтировании компонента. В нем происходит получение массива товара из LS и записывается это в состояние setProductsInLS:

useEffect(() => {

    const fromLS = localStorage.getItem(favoritesKey);

    if(fromLS) {

      const products = JSON.parse(fromLS);

      setProductsInLS(products);

    }

  }, [])

# Helpers (вспомогательные функции)

## getFilters

getFilters – функция, которая выбирает из продукта необходимые значения для фильтров.

Что в этой функции происходит?

Для начала создается объект с начальными значениями для фильтров:

const filtersInfo = {

    categories: new Set(),

    prices: {

      min: 0,

      max: 0

    },

    colors: new Set()

  };

Для ключей categories и colors создается множество уникальных значений Set. Для ключа prices создается вложенный объект с min и max значениями цены. Изначально они равны нулю.

Далее мы приступаем к заполнению filtersInfo необходимыми значениями:

 products.forEach((product) => {

Для этого мы проходим по каждому продукту в массиве. И с каждым продуктом делаем следующее:

product.categories.forEach((category) => {

      filtersInfo.categories.add(category);

    });

Здесь мы в каждом товаре перебираем значение categories (т. к. их может быть несколько) и каждое значение записываем в Set.

Далее определяем максимальную цену:

if (product.price > filtersInfo.prices.max) {

      //если значение цены у товара больше, чем стоит в объекте, то записываем его как max

      filtersInfo.prices.max = product.price

    };

Здесь на каждой итерации происходит сравнение цены текущего продукта с уже записанным в filtersInfo. На первой итерации происходит сравнение начального значения 0 со значением цены первого продукта. Так как оно в любом случае больше 0, то в prices.max записывается цена первого продукта. На второй итерации уже сравнивается prices.max = цена первого продукта и цена второго. Таким образом, находится максимальная цена.

Далее мы заполянем Set у colors:

 filtersInfo.colors.add(product.color);

  });

Здесь все просто, так как у продукта один цвет. На этом цикл foreach заканчивается.

Далее мы преобразуем множества Set в массивы для того, чтобы в компонентах фильтров можно было работать с их значениями:

  filtersInfo.categories = [...filtersInfo.categories];

  filtersInfo.colors = [...filtersInfo.colors];

  return filtersInfo

}

## randomProducts

randomProducts – функция, которая получает 3 рандомных продукта для вывода в блоке LastViewed.

Для начала мы получаем 1 рандомный продукт:

export const getRandomProduct = ({ products }) => {

  const getRandomId = Math.floor(Math.random() \* 24) + 1;

  return products.find((product) => product.id === getRandomId);

}

Далее мы пишем функцию, которая три раза вызывает getRandomProduct, записывает каждый продукт в массив для вывода на странице, а также проверяет, что бы в массив не попали одинаковые продукты:

export const threeRandomProducts = (data) => {

  const randomProducts = [];

  const usedId = new Set();

Здесь мы создаем пустой массив, куда будем записывать продукты; создаем пустое множество уникальных значений, куда будем записывать те продукты, что уже добавлены в массив.

Далее запускаем цикл, который срабатывает до тех пор, пока массив randomProducts не заполнится до трех:

while (randomProducts.length < 3) {

    const randomProduct = getRandomProduct(data); //Находим рандом продукт

    //проверяем,нет ли этого продукта в уже использованных

    if (!usedId.has(randomProduct.id)) {

      randomProducts.push(randomProduct); //если этот продукт еще не использован то добавляем его в массив продуктов для вывода на страницу

      usedId.add(randomProduct.id); //добавляем в использованные продукты

    }

  }

  return randomProducts;

В этом цикле мы так же проверяем, добавлен этот продукт уже в массив или нет. Если id продукта не записано в множестве usedId, то мы данный продукт добавляем в массив для вывода, затем в множество использованных id.