Документация по проекту Fashionee.

Содержание

[Введение 2](#_Toc195645892)

[1. Общие компоненты (common components) 3](#_Toc195645893)

[1.1 Header (шапка страницы) 3](#_Toc195645894)

[1.2 Content-Block (блок с контентом) 3](#_Toc195645895)

[1.3 Footer («подвал» страницы) 3](#_Toc195645896)

[2. UI компоненты (ui) 3](#_Toc195645897)

[2.1 NavLink (навиагционная ссылка) 3](#_Toc195645898)

[2.2 Socials (блок с ссылками на социальные сети) 4](#_Toc195645899)

[3. Showcase (основное содержимое страницы Shop) 4](#_Toc195645900)

[3.1 SideBar (боковое меню) 6](#_Toc195645901)

[3.1.1 Search 6](#_Toc195645902)

[3.1.2 Filters 6](#_Toc195645903)

[3.1.3 LastViewed 7](#_Toc195645904)

[3.2 Products (основной блок страницы) 8](#_Toc195645905)

[3.2.1 Функции 8](#_Toc195645906)

[- *Добавление в корзину* 9](#_Toc195645907)

[3.2.2 Sort 10](#_Toc195645908)

[3.2.3 Product 11](#_Toc195645909)

[3.2.4 Pagination 11](#_Toc195645910)

[3 Helpers (вспомогательные функции) 11](#_Toc195645911)

[3.1 getProducts 11](#_Toc195645912)

[3.2 getFilters 14](#_Toc195645913)

[3.3 randomProducts 15](#_Toc195645914)

# 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.

Имеет набор состояний, отвечающих за разные значения:

* Состояние, в котором записывается значение всего массива продуктов для отображения на странице:

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

* Состояние, в котором для компонента Sort хранится начальное значение выпадающего списка сортировки. В дальнейшем изменяется, откликаясь на действия пользователя:

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

* Состояние, устанавливающее начальное значение страницы для пагинации. В дальнейшем изменяется, откликаясь на действия пользователя:

const [page, setPage] = useState(0);

* Состояние, в котором хранится общая длина массива товаров. Необходимо для корректного отображения надписи:

const [countOfProducts, setCountOfProducts] = useState(0);

* Состояние, в котором хранится введённое пользователем значение для поиска товара по названию:

const [searchValue, setSearchValue] = useState('');

* Состояние, в котором хранятся выбранные пользователем категории фильтрации:

const [selectedCategory, setSelectedCategory] = useState('');

* Состояние, в котором хранится выбранный пользователем диапазон цены для фильтрации товаров:

const [prices, setPrices] = useState({});

* Состояние, хранящее выбранные пользователем цвета для фильтрации:

const [selectedColors, setSelectedColors] = useState([]);

Так же имеет несколько однострочных функций изменения состояний:

* со значением сортировки:

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

* со значением страницы, для переключения пользователя между страницами:

  const changePage = (page) => setPage(page);

* со значением для поиска среди товаров:

const changeSearch = (value) => setSearchValue(value);

* со значением категорий для фильтрации:

const changeCategory = (category) => setSelectedCategory(category);

* со значением диапазона цены для фильтрации:

const changePrices = (min, max) => setPrices({ min, max });

* со значением цветов для фильтрации:

const changeColors = (colors) => setSelectedColors(colors);

В этом компоненте так же происходит вызов вспомогательной функции [getProducts](#_getProducts), куда передаются:

* объект со значением текущей страницы, количества товаров на одной странице;
* значение сортировки;
* объект со значениями для фильтрации: поиска, выбранной категории товара, диапазона цены, выбранных цветов.

В результате функция возвращает объект, который записывается в переменную info. Далее происходит обновление состояния, в котором хранится массив продуктов для отображения на странице, а также обновления состояния, следящего за количеством товаров для отображения.

Вызов этой функции происходит каждый раз при изменении пользователем страницы, типа сортировки или одного из значений для фильтрации:

useEffect(() => {

    const pagination = {

      page: page,

      itemsPerPage: itemsPerPage

    }

    const filters = {

      search: searchValue,

      category: selectedCategory,

      prices,

      colors: selectedColors

    }

    const info = getProducts(pagination, sortType, filters);

    setProducts(info.products);

    setCountOfProducts(info.total);

  }, [page, sortType, searchValue, selectedCategory, prices, selectedColors])

## SideBar (боковое меню)

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

* Search – поиск товара;
* [Filters](#_Filters) – различные фильтры для поиска товара;
* [LastViewed](#_LastViewed) – последние просмотренные товары;
  + 1. Search

Компонент Search представляет собой глупый компонент, отображающий input для поиска и записывающий введённое пользователем значение в этот input при помощи функции changeSearch, получаемой им из компонента [Showcase](#_Showcase_(основное_содержимое):

export function Search ({ changeSearch }) {

  return(

    <div className="sidebar-search">

      <input className="search-input" type="text" placeholder="Search" onChange={(e) => changeSearch(e.target.value)}/>

      <button className="search-btn">

        <img src={searchImg} alt="search" />

      </button>

    </div>

  )

}

### 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.

В данном компоненте есть несколько состояний:

* Состояния, хранящее введенные пользователем значения минимальной и максимальной цены. Начальное значение установлено из infoForFilters:

const [selectedMinPrice, setSelectedMinPrice] = useState(infoForFilters.prices.min);

  const [selectedMaxPrice, setSelectedMaxPrice] = useState(infoForFilters.prices.max);

* Состояние, хранящее выбранные пользователем цвета:

  const [selectedColors, setSelectedColors] = useState([]);

Так же есть функции изменения состояний со значениями минимальной и максимальной цены. Передаются в компонент PriceFilter, где срабатывают на изменение input onChange :

  const changeMinPrice = (e) => {

    const price = +e.target.value;

    setSelectedMinPrice(price);

  }

  const changeMaxPrice = (e) => {

    const price = +e.target.value;

    setSelectedMaxPrice(price);

  }

Функция, изменяющая состояние со значением выбранных пользователем цветов. Передается в компонент ColorFilter, срабатывает так же на onChange:

  const onColorSelect = (e) => {

    const color = e.target.value;

      if(selectedColors.includes(color)) {

        const filteredColors = selectedColors.filter((selectedColor) => selectedColor !== color);

        setSelectedColors(filteredColors)

        return

      }

    setSelectedColors([...selectedColors, color])

  };

Функция, использующееся как обработчик клика на кнопку Apply filter. В ней изменяются состояния со значением цены и цветов для фильтрации, переданные из родительского [Showcase](#_Showcase_(основное_содержимое):

  const applyFilter = () => {

    changePrices(selectedMinPrice, selectedMaxPrice)

    changeColors(selectedColors)

  }

### 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, countOfProducts, changePage,

  totalPages, currentPage

 })

### Функции

#### Избранное

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);

    }

  }, [])

#### Добавление в корзину

1. Логика добавления товара в корзину срабатывает на нажатие кнопки «Купить» в карточке товара в компоненте [Product](#_Product). При нажатии в функцию передается весь товар целиком:

 const buyProduct = (product) => {

1. По аналогии с функцией добавления товара в избранное, сначала получается товар, уже хранящийся в корзине:

    const fromCart = localStorage.getItem(cartKey);

1. Если в корзине ничего нет, то создается массив, куда записывается сам объект товара, и к этому объекту добавляется ключ count, отвечающий за количество конкретного товара в корзине(например, 2 одинаковых футболки):

    if (!fromCart) {

      console.log('товар добавлен в корзину')

      localStorage.setItem(cartKey, JSON.stringify([{...product, count: 1}]));

      return;

    }

1. Если же в корзине на момент нажатия кнопки уже были какие-то товары, то мы преобразуем ранее полученные данные в массив. Далее мы проходимся по этому массиву и проверяем, есть ли уже в корзине товар с таким же id, по которому произошло нажатие:

    const products = JSON.parse(fromCart);

    const inCart = products.find((productInLs) => productInLs.id === product.id);

1. Если id одинаковы, то происходит увеличение ключа count у объекта товара. Обновленные данные так же добавляются в LS:

    if (inCart) {

      const newProducts = products.map((productsInLS) => {

        if (productsInLS.id === product.id) {

          return {

            ...productsInLS,

            count: productsInLS.count + 1

          }

        }

        return productsInLS

      })

        localStorage.setItem(cartKey, JSON.stringify(newProducts));

        return

      }

1. Если же id товара не равны, то тогда тот товар, по которому произошло нажатие, просто добавляется в LS:

      products.push({...product, count: 1})

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

### Sort

Компонент, отображающий выпадающий список <select> с вариантами сортировки товаров на странице. Получает от родительского [Products](#_Products_(основной_блок) такие props, как:

export const Sort = ({ onSort, value }) => (

  <select className="sort" id=" " onChange={onSort} value={value}>

И сразу же их использует как функцию, срабатывающую при изменении пользователем параметра сортировки, и значение, передающееся в эту функцию.

### Product

### Pagination

Компонент, в котором происходит отображение общего количества страниц с товарами, переключение между этими страницами.

Получает из родительского компонента [Products](#_Products_(основной_блок) такие props как:

export const Pagination = ({currentPage, setPage, totalPages})

Далее в компоненте создается массив, заполняемый определенным количеством нулей. Количество нулей в этом массиве равно totalPages.

const pages = new Array(totalPages).fill(0);

Это необходимо для того, чтобы была возможность пройтись по элементам этого массива для отображения нужного количества страниц в строке пагинации на сайте:

return(

    <SC.Pagination>

      {

        pages.map((page, index) => (

        <SC.Page

          key={index}

          isCurrent={currentPage === index}

          onClick={() => setPage(index)}

        >

            {index + 1}

        </SC.Page>

        ))

      }

    </SC.Pagination>

Так же, при клике на номер страницы, происходит переключение, путем обновления состояния setPage.

Свойство isCurrent необходимо для стилизации: текущая страница выделяется красным.

# Helpers (вспомогательные функции)

## getProducts

getProducts – функция, имитирующая взаимодействие с бэкэндом. Она владеет всем списком товаров, фильтрует, сортирует их и обрезает для пагинации.

Функция принимает такие параметры, как:

export const getProducts = (pagination, sortType, filters) => {

  const { products } = data;

Логика фильтрации и сортировки товаров вынесена в отдельные функции, чтобы не перегружать getProducts, вызывающуюся в компоненте [Showcase](#_Showcase_(основное_содержимое):

В первую очередь реализуется фильтрация товаров.

  const productsForModify = onFilter(filters, products);

Далее происходит сортировка товаров:

  onSort(productsForModify, sortType);

После сортировки товаров идет обрезание для пагинации:

const startIndex = pagination.page \* pagination.itemsPerPage;

const slicedProducts = productsForModify.slice(startIndex, startIndex + pagination.itemsPerPage);

Функция возвращает объект со значениями товаров для отображения на странице и общую длину товаров:

  return {

    products: slicedProducts,

    total: productsForModify.length

  }

}

Выше было сказано, что сортировка и фильтрация вынесены в отдельные функции.

Логика фильтрации представлена в отдельной функции onFilter.

Функция получает объект filters, в котором хранятся выбранные пользователем значения для фильтрации, и массив продуктов, который в дальнейшем модифицируются в зависимости от переданных значений фильтрации:

const onFilter = (filters, products) => {

  let productsForModify = [...products];

Сначала происходит поиск товара по названию:

  if(filters.search) {

    productsForModify = productsForModify.filter((product) => product.name.indexOf(filters.search) !== -1);

  }

Далее товары фильтруются по выбранной пользователем категории:

 if(filters.category) {

    productsForModify = productsForModify.filter((product) => product.categories.includes(filters.category));

  }

Затем происходит фильтрация по указанному диапазону цены:

  if(filters.prices.min && filters.prices.max) {

    productsForModify = productsForModify.filter((product) => {

      return product.price >= filters.prices.min && product.price <= filters.prices.max

    })

  } else if(filters.prices.min) {

    productsForModify = productsForModify.filter((product) => product.price >= filters.prices.min)

  } else if(filters.prices.max) {

    productsForModify = productsForModify.filter((product) => product.price <= filters.prices.max)

  }

В конце происходит фильтрация по цветам и возвращение из функции модифицированного массива:

  if(filters.colors.length > 0) {

    productsForModify = productsForModify.filter((product) => filters.colors.includes(product.color))

  }

  return productsForModify

}

Логика сортировки представлена в функции onSort, срабатывающей на выбор пользователем значения из выпадающего списка, который представлен компонентом [Sort](#_Sort).

Через конструкцию switch case обрабатываются все возможные варианты сортировки, а именно: по имени от A-Z; по имени от Z-A; сначала дешевле; сначала дороже. На входе конструкция получает событие, произошедшее на сайте(выбор варианта сортировки). Обработка всех кейсов происходит с помощью метода .sort:

const onSort = (products, type) => {

  switch (type) {

    case 'nameAsc':

      products.sort((a, b) => a.name > b.name ? 1 : -1);

      break;

    case 'nameDesc':

      products.sort((a, b) => a.name < b.name ? 1 : -1);

      break;

    case 'priceAsk':

      products.sort((a, b) => a.price > b.price ? 1 : -1);

      break;

    case 'priceDesk':

      products.sort((a, b) => a.price < b.price ? 1 : -1);

      break;

    default:

  }

}

## 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.