**Методы и средства решения стандартных задач профессиональной деятельности**

**ЛАБОРАТОРНАЯ РАБОТА №9**

**Взаимодействие с пользователем**

**Теория**

Рассмотрим пример интернет-магазина где корзина состоит из двух компонентов — кнопки для вызова окна, а также всплывающего окно самой корзины.

Кнопка корзины представляет из себя круглую кнопку с изображением корзины. В правом верхнем углу находится счетчик выбранных товаров. Кнопку фиксируем на одном месте, чтобы пользователь ее постоянно видел при прокрутке страницы.

Разметка для кнопки корзины:

<button class="cart" id="cart">

<img class="cart\_\_image" src="./images/cart.png" alt="Cart" />

<div class="cart\_\_num" id="cart\_num">0</div>

</button>

Стили для кнопки корзины:

.cart {

width: 75px;

height: 75px;

border-radius: 50%;

background-color: #364364;

transition: 0.1s;

cursor: pointer;

position: fixed; /\* Фиксированное расположение \*/

top: 50px; /\* в правом верхнем углу \*/

right: 50px;

display: flex;

justify-content: center;

align-items: center;

object-fit: contain;

padding: 15px;

box-shadow: 1px 2px 4px rgba(0, 0, 0, 0.1);

}

/\* Увеличиваем кнопку при наведении на нее \*/

.cart:hover {

transform: scale(1.1);

}

/\* Стилизуем счетчик товаров \*/

.cart\_\_num {

position: absolute;

background-color: #d62240;

width: 30px;

height: 30px;

border-radius: 50%;

display: flex;

justify-content: center;

align-items: center;

color: #fff;

font-size: 18px;

font-weight: 500;

top: -5px;

right: -5px;

}

Окно с товарами представляет из себя список выбранных товаров с ценами и кнопками удалить. Внизу находится информация по стоимости заказа и скидки.



Разметка окна оформления заказа:

<div class="popup">

<div class="popup\_\_container" id="popup\_container">

<div class="popup\_\_item">

<h1 class="popup\_\_title">Оформление заказа</h1>

</div>

<div class="popup\_\_item" id="popup\_product\_list">

<div class="popup\_\_product">

<div class="popup\_\_product-wrap">

<img

src="./images/1.jpg"

alt=""

class="popup\_\_product-image"

/>

<h2 class="popup\_\_product-title">

</h2>

</div>

<div class="popup\_\_product-wrap">

<div class="popup\_\_product-price"></div>

<button class="popup\_\_product-delete">&#10006;</button>

</div>

</div>

</div>

<div class="popup\_\_item">

<div class="popup\_\_cost">

<h2 class="popup\_\_cost-title">Итого</h2>

<output class="popup\_\_cost-value" id="popup\_cost">0</output>

</div>

<div class="popup\_\_cost">

<h2 class="popup\_\_cost-title">Скидка</h2>

<output class="popup\_\_cost-value" id="popup\_discount">0</output>

</div>

<div class="popup\_\_cost">

<h2 class="popup\_\_cost-title">Итого со скидкой</h2>

<output class="popup\_\_cost-value" id="popup\_cost\_discount"

>0</output>

</div>

</div>

<button class="popup\_\_close" id="popup\_close">&#10006;</button>

</div>

</div>

Стили окна оформления заказа:

.popup {

position: fixed;

top: 0;

left: 0;

background-color: rgba(0, 0, 0, 0.33); /\* Небольшое затемнение на фоне \*/

z-index: 1000; /\* Поверх всех окон \*/

width: 100%;

height: 100%;

display: none; /\* По умолчанию скрываем окно \*/

justify-content: center;

align-items: center;

user-select: none;

overflow-y: scroll;

}

/\* Стилизация контейнера окна \*/

.popup\_\_container {

display: flex;

flex-direction: column;

justify-content: space-between; /\* Распределяем элементы внутри равномерно \*/

width: 100%;

max-width: 800px;

min-height: 300px;

background-color: #fff;

box-shadow: 1px 2px 4px rgba(0, 0, 0, 0.1);

border-radius: 4px;

position: relative;

}

/\* Элементы окна (заголовок, список товаров, вывод стоимости) \*/

.popup\_\_item {

border-bottom: 1px solid #ddd;

padding: 20px;

}

.popup\_\_item:last-of-type {

border-bottom: none;

}

.popup\_\_title {

font-size: 20px;

}

/\* Стили для вывода товара \*/

.popup\_\_product {

display: flex;

align-items: center;

justify-content: space-between;

margin-bottom: 20px;

}

.popup\_\_product-wrap {

display: flex;

align-items: center;

}

.popup\_\_product-image {

width: 100px;

height: 100px;

object-fit: contain;

margin-right: 10px;

}

.popup\_\_product-title {

max-width: 300px;

font-weight: 500;

}

.popup\_\_product-price {

font-size: 18px;

margin-right: 15px;

}

/\* Стили для кнопки удаления товара \*/

.popup\_\_product-delete {

font-size: 12px;

padding: 5px;

cursor: pointer;

color: #d62240;

}

/\* Стили для вывода стоимости товаров \*/

.popup\_\_cost {

display: flex;

align-items: center;

justify-content: flex-end;

margin-bottom: 5px;

}

.popup\_\_cost-title {

margin-right: 15px;

font-size: 20px;

color: #364364;

text-align: right;

}

.popup\_\_cost-value {

font-size: 20px;

}

/\* Стили для кнопки закрытия окна \*/

.popup\_\_close {

position: absolute;

cursor: pointer;

top: 0;

right: 0;

padding: 20px;

color: rgba(54, 67, 100, 0.7);

font-size: 20px;

}

.popup\_\_close:hover {

color: rgb(54, 67, 100);

}

.popup--open {

display: flex;

}

Для взаимодействия с пользоватем магазина и добавления товаров в корзину напишем две вспомогательные функции: **toNum** и **toCurrency**.

function toNum(str) {

const num = Number(str.replace(/ /g, ""));

return num;

}

function toCurrency(num) {

const format = new Intl.NumberFormat("ru-RU", {

style: "currency",

currency: "RUB",

minimumFractionDigits: 0,

}).format(num);

return format;

}

Функция **toNum** принимает строковый параметр и удаляет в нем все пробелы, затем преобразует его в число с помощью встроенной функции **Number**. Эта функция  полезна для преобразования строк цен в числовые значения, которые можно использовать для расчетов.

Функция **toCurrency** форматирует число в строку валюты с использованием символа российского рубля.

Считываем все элементы корзины и popup (окна оформления заказа):

const cardAddArr = Array.from(document.querySelectorAll(".card\_\_add"));

const cartNum = document.querySelector("#cart\_num");

const cart = document.querySelector("#cart");

Считываем все элементы окна оформления заказа:

const popup = document.querySelector(".popup");

const popupClose = document.querySelector("#popup\_close");

const body = document.body;

const popupContainer = document.querySelector("#popup\_container");

const popupProductList = document.querySelector("#popup\_product\_list");

const popupCost = document.querySelector("#popup\_cost");

const popupDiscount = document.querySelector("#popup\_discount");

const popupCostDiscount = document.querySelector("#popup\_cost\_discount");

Для открытия мы нажимаем круглую кнопку корзины, а для закрытия на крестик в самом окне.

Считываем нужные элементы и навешиваем обработчики на клик:

cart.addEventListener("click", (e) => {

e.preventDefault();

popup.classList.add("popup--open");

body.classList.add("lock");

});

popupClose.addEventListener("click", (e) => {

e.preventDefault();

popup.classList.remove("popup--open");

body.classList.remove("lock");

});

Далее опишем класс для одного товара.

class Product {

imageSrc;

name;

price;

priceDiscount;

constructor(card) {

this.imageSrc = card.querySelector(".card\_\_image").children[0].src;

this.name = card.querySelector(".card\_\_title").innerText;

this.price = card.querySelector(".card\_\_price--common").innerText;

this.priceDiscount = card.querySelector(".card\_\_price--discount").innerText;

}

}

Класс **Product** представляет товар с некоторыми свойствами, такими как источник изображения, название, цена и цена со скидкой. У него есть конструктор, который принимает параметр **card**, который будет HTML-элементом, представляющим карточку продукта на веб-странице.

Конструктор инициализирует свойства объекта **Product**, запрашивая параметр **card** для определенных элементов с помощью CSS-селекторов и извлекая соответствующую информацию из этих элементов. Например, свойство **imageSrc** устанавливается путем получения исходного URL-адреса первого дочернего элемента с классом **card\_\_image** внутри параметра **card**.

Класс для описания корзины покупок и расчета стоимости и скидок.

const cardAddArr = Array.from(document.querySelectorAll(".card\_\_add"));

const cartNum = document.querySelector("#cart\_num");

class Cart {

products;

constructor() {

this.products = [];

}

get count() {

return this.products.length;

}

addProduct(product) {

this.products.push(product);

}

removeProduct(index) {

this.products.splice(index, 1);

}

get cost() {

const prices = this.products.map((product) => {

return toNum(product.price);

});

const sum = prices.reduce((acc, num) => {

return acc + num;

}, 0);

return sum;

}

get costDiscount() {

const prices = this.products.map((product) => {

return toNum(product.priceDiscount);

});

const sum = prices.reduce((acc, num) => {

return acc + num;

}, 0);

return sum;

}

get discount() {

return this.cost - this.costDiscount;

}

}

Класс **Cart**, который представляет корзину покупок. У него есть конструктор, который инициализирует пустое свойство массива **products**.

Класс имеет методы для добавления и удаления товаров из корзины, которые изменяют массив **products**, добавляя или удаляя данный товар по указанному индексу.

Также определены три метода получения для стоимости корзины, стоимости со скидкой и скидки со скидкой. Эти методы вычисляют соответствующие значения путем перебора массива **products** и суммирования цен или скидок на каждый продукт.

Вспомогательная функция **toNum** используется для преобразования строк цен в числовые значения.

## Создаем объект корзины и сохраняем его в localStorage

const myCart = new Cart();

if (localStorage.getItem("cart") == null) {

localStorage.setItem("cart", JSON.stringify(myCart));

}

const savedCart = JSON.parse(localStorage.getItem("cart"));

myCart.products = savedCart.products;

cartNum.textContent = myCart.count;

Этот код инициализирует новый экземпляр класса **Cart** и присваивает его переменной **myCart**. Затем он проверяет, есть ли какие-либо данные, хранящиеся в локальном хранилище браузера под ключом **cart**. Если таких данных нет, он сохраняет версию объекта **myCart** в формате JSON в локальном хранилище.

Далее код извлекает данные из локального хранилища под ключом «card» и преобразует их обратно в объект с помощью JSON.parse(). Затем проанализированный объект присваивается переменной **savedCart**.

Наконец, массив **products** из **savedCart** присваивается массиву **products** из **myCart**, а значение свойства **count** из **myCart** используется для обновления текстового содержимого элемента **cartNum**.

Обратите внимание, что этот код предполагает, что класс Cart имеет свойства products и count, и что переменная cartNum представляет элемент в DOM, который отображает количество товаров в корзине.

## **Добавляем товар в корзину**

myCart.products = cardAddArr.forEach((cardAdd) => {

cardAdd.addEventListener("click", (e) => {

e.preventDefault();

const card = e.target.closest(".card");

const product = new Product(card);

const savedCart = JSON.parse(localStorage.getItem("cart"));

myCart.products = savedCart.products;

myCart.addProduct(product);

localStorage.setItem("cart", JSON.stringify(myCart));

cartNum.textContent = myCart.count;

});

});

Этот код добавляет прослушиватель событий к каждому элементу в массиве **cardAddArr**, который содержит элементы DOM, представляющие кнопки **cardAdd** на веб-странице. Когда пользователь нажимает одну из этих кнопок, вызывается функция прослушивания.

Функция прослушивателя сначала предотвращает действие события click по умолчанию, используя e.preventDefault(). Затем он находит ближайший элемент **.card** к нажатой кнопке, используя метод **closest()**.

Код создает новый экземпляр продукта, используя элемент **card**, извлекая информацию из **card** для создания объекта **product**.

Далее код извлекает сохраненные данные корзины из localStorage и присваивает свой массив **products** свойству **myCart.product**s. Затем он добавляет объект **product** в экземпляр **myCart**, используя свой метод **addProduct**.

После добавления товара в корзину код сохраняет обновленный объект **myCart** в localStorage в виде строки JSON с помощью **JSON.stringify()**. В итоге, он обновляет текстовое содержимое элемента **cartNum**, чтобы отобразить новое количество товаров в корзине, используя свойство **myCart.count**.

## **Заполнение корзины**

function popupContainerFill() {

popupProductList.innerHTML = null;

const savedCart = JSON.parse(localStorage.getItem("cart"));

myCart.products = savedCart.products;

const productsHTML = myCart.products.map((product) => {

const productItem = document.createElement("div");

productItem.classList.add("popup\_\_product");

const productWrap1 = document.createElement("div");

productWrap1.classList.add("popup\_\_product-wrap");

const productWrap2 = document.createElement("div");

productWrap2.classList.add("popup\_\_product-wrap");

const productImage = document.createElement("img");

productImage.classList.add("popup\_\_product-image");

productImage.setAttribute("src", product.imageSrc);

const productTitle = document.createElement("h2");

productTitle.classList.add("popup\_\_product-title");

productTitle.innerHTML = product.name;

const productPrice = document.createElement("div");

productPrice.classList.add("popup\_\_product-price");

productPrice.innerHTML = toCurrency(toNum(product.priceDiscount));

const productDelete = document.createElement("button");

productDelete.classList.add("popup\_\_product-delete");

productDelete.innerHTML = "✖";

productDelete.addEventListener("click", () => {

myCart.removeProduct(product);

localStorage.setItem("cart", JSON.stringify(myCart));

popupContainerFill();

});

productWrap1.appendChild(productImage);

productWrap1.appendChild(productTitle);

productWrap2.appendChild(productPrice);

productWrap2.appendChild(productDelete);

productItem.appendChild(productWrap1);

productItem.appendChild(productWrap2);

return productItem;

});

productsHTML.forEach((productHTML) => {

popupProductList.appendChild(productHTML);

});

popupCost.value = toCurrency(myCart.cost);

popupDiscount.value = toCurrency(myCart.discount);

popupCostDiscount.value = toCurrency(myCart.costDiscount);

}

Этот код определяет функцию **popupContainerFill()**, которая заполняет всплывающий контейнер информацией о продукте из корзины покупок, хранящейся в локальном хранилище.

Функция сначала извлекает данные корзины из локального хранилища, а затем сопоставляет массив продуктов корзины, чтобы создать новый массив элементов HTML, представляющих каждый продукт в корзине. Каждый элемент продукта создается путем создания вложенных элементов **div**, которые содержат изображение продукта, заголовок, цену и кнопку удаления.

При нажатии кнопки удаления для данного продукта запускается прослушиватель событий, который удаляет этот продукт из корзины, вызывая метод **myCart.removeProduct()**, обновляет информацию о корзине в локальном хранилище, а затем обновляет всплывающий контейнер с обновленными данными. информацию о корзине, снова вызвав **popupContainerFill()**.

В итоге, функция устанавливает значение полей общей стоимости, скидки и стоимости со скидкой во всплывающем контейнере, чтобы отразить обновленную информацию о корзине.

**Задание**

Используя события на странице с перечнем товаров реализовать алгоритм покупки товара с добавлением в итоговую покупательскую корзину со следующими возможностями:

* + Добавление товаров в корзину покупателя
  + Удаление товаров из корзины
  + Автоматическое обновление итоговой суммы всех покупок.