

Projekat

Segment web prodavnice

Mentor:

Stevan Ljiljak

Radio:

Aleksandar Stanojevic

22.03.2023.

Sadržaj:

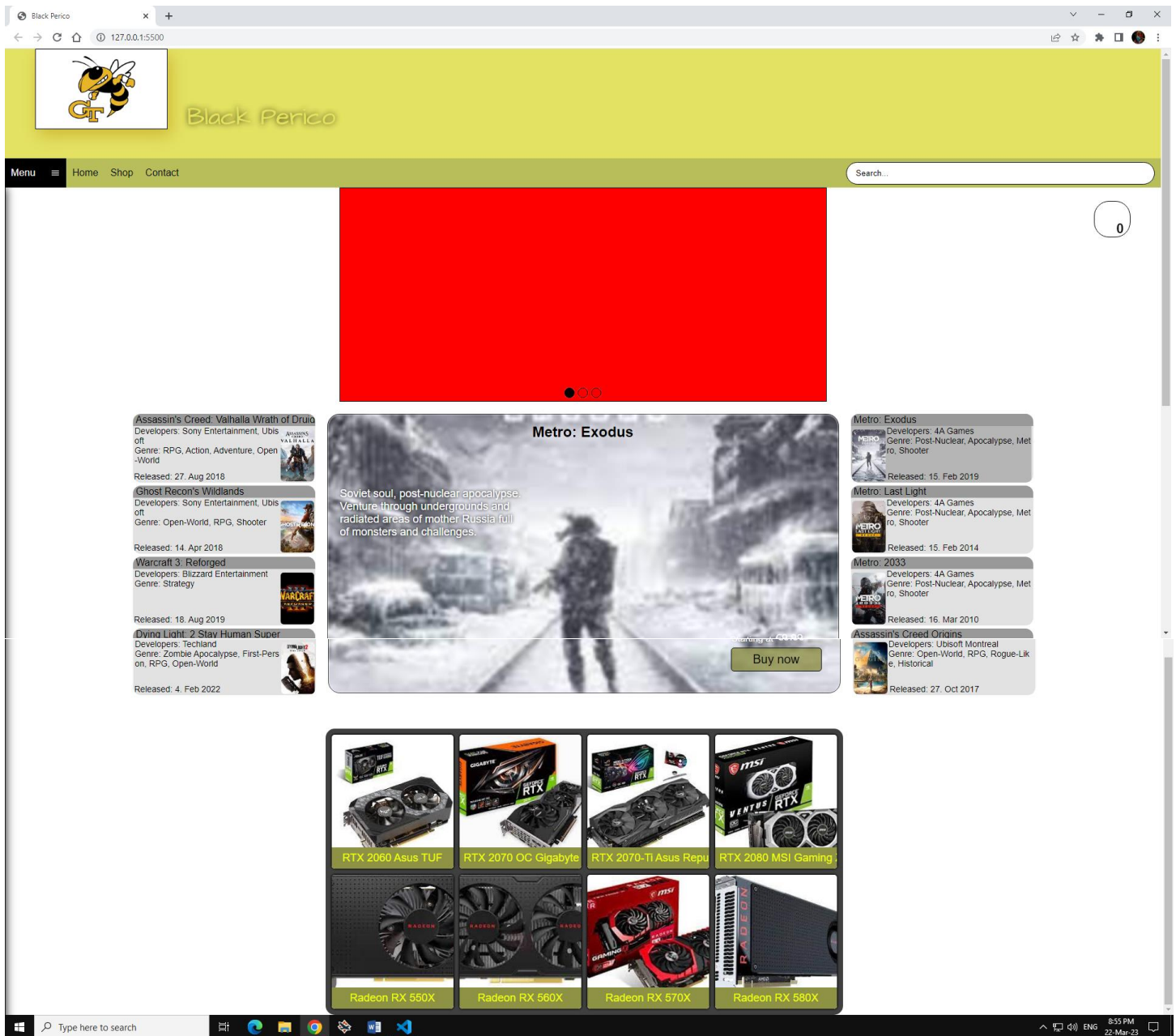
Uvod	3
Pocetna strana	4
Prvi element pocetne strane	5
Drugi element pocetne strane.....	6
Treci element pocetne strane.....	9
Shop strana	12
Filteri druge strane	15
Kupovina artikala sa obe strane	18

Uvod

Ovaj projekat sam napravio sa ciljem da kroz njega ucim i provezbavam javascript, sa ciljem da u potpunosti mogu da iznesem svoju ideju i zamisli i predstavim to na sajtu. Projekat je napravljen na mom prvom susretu sa JS, tako da pojedini(neki od prvih elemenata) pristupi u kodiranju nisu bas idealni u smislu da se prilikom zelje da se nesto na sajtu promeni, morao bi biti promenjen veci deo koda sa posebnom paznjom. Medjutim trudio sam se da kod 'ispoliram' sto vise na nacin da se tokom koriscenja sajta ne pojavljuju ispadi koji bi odstupali od ponasanja elementa suprotno od onoga sto sam ja zamislio i zeleo. Ali jos jednom, tu se moze naci i kod koji ne nudi optimalno resenje, vec bi se prilikom zelje da se na sajtu nesto promeni, veci deo koda morao menjati. Obratio sam paznju na ovaj pristup u odredjenoj meri pri kasnijim elementima koje sam izradjivao u projektu, gde sam npr. umesto koriscenja samih vrednosti u delovima koda, koristio promenljive, tako da bi se prilikom zelje da se nesto promeni na sajtu mogla promeniti samo vrednost jedne promenljive i ta vrednost bi se nasledila na vise delova koda.

Ovaj projekat ne treba da predstavlja nikakav vrhunac kreativne strane (koriscenje CSS svojstava radi postizanja sto boljih vizuelnih izgleda), vec je tacka fokusa na kodiranju u JS.

Pocetna strana

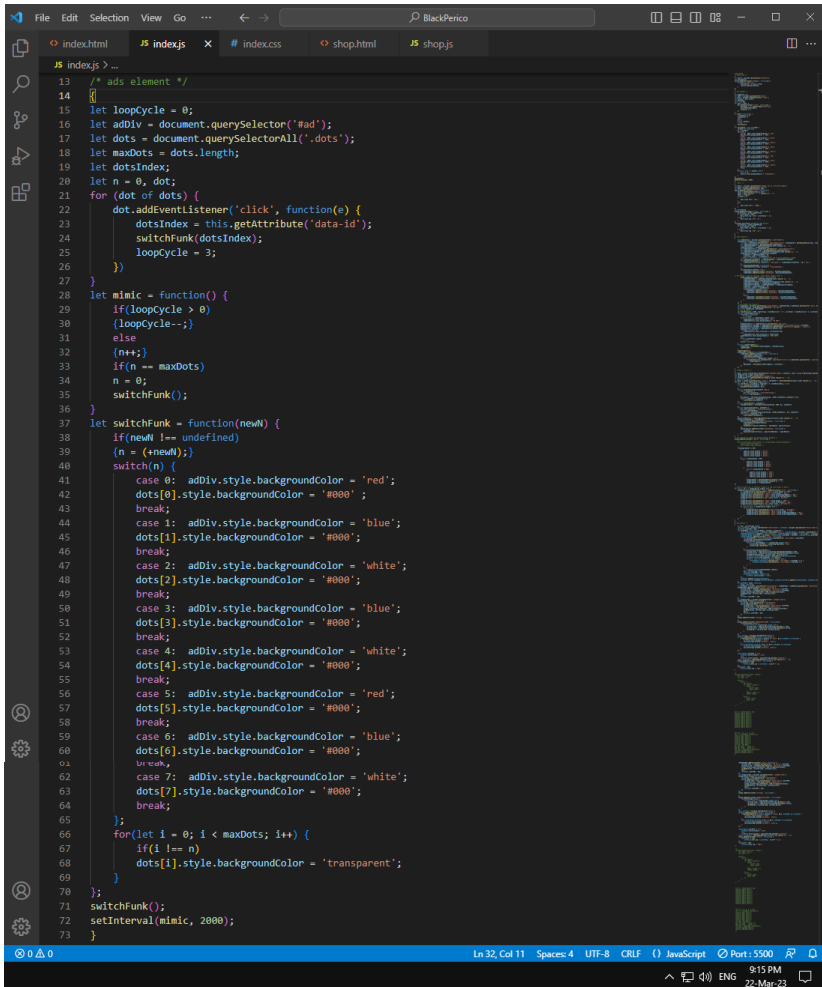


Pocetna strana je zamisljena tako da sadrzi:

- 1.element koji treba da sadrzi samo niz slika,
- 2.centralni element na kojem se predstavljaju podaci sa vise manjih elemenata koji ga okružuju i nude dugme za njihovu kupovinu,
- 3.element kontejner u kojem se nalaze vise elemenata za kupovinu.

Prvi element pocetne strane

Ovaj element je vrlo prosto zamisljen, on treba da sadrzi vise manjih div elemenata na dnu i pritiskom na njih se menja slika/pozadina kontejner elementa. Takodje postoji i automatsko prolazjenje kroz niz ovih manjih elemenata u odredenom vremenskom intervalu, a klikom na njih prikazuje se zeljena slika u glavnom elementu, pri cemu se interval vremena za dalje prolazjenje povecava za *3 pre nego sto nastavi svojim starim tokom.

A screenshot of a code editor with a dark theme. The editor shows a JavaScript file named 'index.js'. The code implements an advertisement slider. It starts with a comment '/* ads element */' and defines variables for 'loopCycle', 'adDiv', 'dots', 'maxDots', 'dotsIndex', and 'n'. A 'for' loop iterates over 'dots' to add event listeners. The 'mimic' function checks 'loopCycle' and increments 'n'. The 'switchFunk' function uses a switch statement to set background colors for 'adDiv' and 'dots' based on 'n'. It also sets 'dots[n].style.backgroundColor' to 'transparent' for other dots. A 'setInterval' call at the bottom triggers the 'mimic' function every 2000ms. The status bar at the bottom shows 'Ln 32, Col 11', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', 'Port: 5500', and the time '9:15 PM 22-Mar-23'.

```
13 /* ads element */
14
15 let loopCycle = 0;
16 let adDiv = document.querySelector('#ad');
17 let dots = document.querySelectorAll('.dots');
18 let maxDots = dots.length;
19 let dotsIndex;
20 let n = 0;
21 for (dot of dots) {
22   dot.addEventListener('click', function(e) {
23     dotsIndex = this.getAttribute('data-id');
24     switchFunk(dotsIndex);
25     loopCycle = 3;
26   })
27 }
28 let mimic = function() {
29   if(loopCycle > 0) {
30     loopCycle--;
31   } else {
32     (n++);
33     if(n == maxDots)
34       n = 0;
35     switchFunk();
36   }
37 }
38 let switchFunk = function(newN) {
39   if(newN !== undefined) {
40     n = (newN);
41     switch(n) {
42       case 0: adDiv.style.backgroundColor = 'red';
43               dots[0].style.backgroundColor = '#000';
44               break;
45       case 1: adDiv.style.backgroundColor = 'blue';
46               dots[1].style.backgroundColor = '#000';
47               break;
48       case 2: adDiv.style.backgroundColor = 'white';
49               dots[2].style.backgroundColor = '#000';
50               break;
51       case 3: adDiv.style.backgroundColor = 'blue';
52               dots[3].style.backgroundColor = '#000';
53               break;
54       case 4: adDiv.style.backgroundColor = 'white';
55               dots[4].style.backgroundColor = '#000';
56               break;
57       case 5: adDiv.style.backgroundColor = 'red';
58               dots[5].style.backgroundColor = '#000';
59               break;
60       case 6: adDiv.style.backgroundColor = 'blue';
61               dots[6].style.backgroundColor = '#000';
62               break;
63       case 7: adDiv.style.backgroundColor = 'white';
64               dots[7].style.backgroundColor = '#000';
65               break;
66     }
67     for(let i = 0; i < maxDots; i++) {
68       if(i !== n)
69         dots[i].style.backgroundColor = 'transparent';
70     }
71     switchFunk();
72     setInterval(mimic, 2000);
73 }
```

Ovaj element predstavlja moj prvi izazov sa JS te je stoga i veoma prosto(loše) odradjen i pri svakom promeni na sajtu dosta toga u kodu bi trebalo da se menja.

Imamo promenljivu `adDiv` koja sadrzi kontejner element i `dots` kao niz manjih div elemenata unutar njega od kojih svi imaju atribut `id` u HTML-u sa odredenim brojem, redom. Na liniji 21 prolazimo kroz ceo niz i svakom elementu stavljamo `addEventListener` koji prilikom klika na njega kupi atribut i skladišti ga u prom `dotsIndex`, poziva se metoda `switchFunk` kojoj se predaje taj index i na osnovu njega se u `switch` iskazu odredjuje rezultat(rezultat ovde predstavlja samo pozadinsku boju umesto pozadinsku sliku, ili prosto sliku unutar elementa). Zatim se prolazi kroz ceo niz i svim ostalim elementima se dodeljuje transparentna boja osim onoga koji je kliknut, njegova boja je crna. `switchFunk` metoda se poziva sama za prvi put na liniji 71 nakon cega se svake 2 sekunde poziva kroz `mimic` metodu.

U ovom scope-u imamo promenljivu `loopCycle`. Ona služi kao operand u množenju, prilikom klika na bilo koji element iz niza `dots` njegova vrednost se postavlja na 3, što znači da će metoda `mimic`(koja se poziva svake 2s) zapravo davati rezultat(poziv glavne metode-`switchFunk`) tek nakon 3 prolaska kroz nju(nakon 6s).

Drugi element pocetne strane

Ovaj element se sastoji iz 3 dela. Centralni deo sa svojim naslovom, textom, cenom i dugmetom, leva njegova strana koja sadrzi 4 manja elementa od kojih svaki sa nazivom, textom i slikom, i isti slucaj sa desne strane. Prilikom klika na neki od elemenata sa strana centralni element dobija vrednosti naslova iz njega, text koji je sadrzan u data- atributu HTML-a, cenu sadrzanu u atributu HTML-a kao i sliku.

Assassin's Creed: Valhalla
Wrath of Druids
Developers: Sony Entertainment, Ubisoft
Genre: RPG, Action, Adventure, Open-World
Released: 27. Aug 2018

Ghost Recon's Wildlands
Developers: Sony Entertainment, Ubisoft
Genre: Open-World, RPG, Shooter
Released: 14. Apr 2018

Warcraft 3: Reforged
Developers: Blizzard Entertainment
Genre: Strategy
Released: 18. Aug 2019

Dying Light 2 Stay Human
Super
Developers: Techland
Genre: Zombie Apocalypse, First-Person, RPG, Open-World
Released: 4. Feb 2022

Ghost Recon's Wildlands
Tom Clancy's Ghost Recon Wildlands is a third-person tactical shooter video game developed by Ubisoft Paris in collaboration with Ubisoft Barcelona, Ubisoft Reims, Ubisoft Montpellier, Ubisoft Annecy, Ubisoft Milan and Ubisoft Belgrade, and published by Ubisoft.

Starting at **€9.99**
[Buy now](#)

Metro: Exodus
Developers: 4A Games
Genre: Post-Nuclear, Apocalypse, Metro, Shooter
Released: 15. Feb 2019

Metro: Last Light
Developers: 4A Games
Genre: Post-Nuclear, Apocalypse, Metro, Shooter
Released: 15. Feb 2014

Metro: 2033
Developers: 4A Games
Genre: Post-Nuclear, Apocalypse, Metro, Shooter
Released: 16. Mar 2010

Assassin's Creed Origins
Developers: Ubisoft Montreal
Genre: Open-World, RPG, Rogue-Like, Historical
Released: 27. Oct 2017

```
File Edit Selection View Go ... BackPenico
index.html index.js index.css shop.html shop.js
110 /* Side elements */
111 {
112   let sideElements = document.querySelectorAll('.side-element');
113   sideElements.forEach(function(sideElement) {
114     let wrap = sideElement.querySelector('.side-element-wrap');
115     let sideElementComputed = getComputedStyle(sideElement);
116     let sideElementWidth = sideElementComputed.width.replace('px', '');
117     sideElementWidth = (+sideElementWidth);
118     let sideElementTitle = sideElement.querySelector('.side-element-title');
119     let sideElementTitleComputed = getComputedStyle(sideElementTitle);
120     let sideElementTitleWidth = sideElementTitleComputed.width.replace('px', '');
121     sideElementTitleWidth = (+sideElementTitleWidth);
122     wrapWidth = sideElementWidth - 62;
123     wrap.style.width = `${wrapWidth}px`;
124     /* Width is -10 because of +5 padding-left and +5 extra padding to right */
125     let sideElementTitleOffset = sideElementWidth - sideElementTitleWidth;
126     let mouseOverSideElement = function() {
127       sideElementTitle.style.transform = `translateX(' + (sideElementTitleOffset - 10) + 'px)`;
128     };
129     let mouseLeaveSideElement = function() {
130       sideElementTitle.style.transform = `translateX(0px)`;
131     };
132     if(sideElementTitleOffset < 0) {
133       sideElement.addEventListener('mouseover', mouseOverSideElement);
134       sideElement.addEventListener('mouseleave', mouseLeaveSideElement);
135     }
136   });
137   /* New values in case of resizing, side-element changes width */
138   window.addEventListener('resize', function(e) {
139     sideElementWidth = sideElementComputed.width.replace('px', '');
140     sideElementWidth = (+sideElementWidth);
141     sideElementTitleWidth = sideElementTitleComputed.width.replace('px', '');
142     sideElementTitleWidth = (+sideElementTitleWidth);
143     sideElementTitleOffset = sideElementWidth - sideElementTitleWidth;
144     wrapWidth = sideElementWidth - 62;
145     wrap.style.width = `${wrapWidth}px`;
146     if(sideElementTitleOffset < 0) {
147       sideElement.addEventListener('mouseover', mouseOverSideElement);
148       sideElement.addEventListener('mouseleave', mouseLeaveSideElement);
149     }
150     else {
151       sideElement.removeEventListener('mouseover', mouseOverSideElement);
152       sideElement.removeEventListener('mouseleave', mouseLeaveSideElement);
153     }
154   });
155   /* Cat element and looping */
156   let midElement = document.querySelector('#cat-middle');
157   let midElementImg = midElement.querySelector('img');
158   let midElementText = midElement.querySelector('.cat-text');
159   let n = 0;
160   let timeOutLoop, addTimeOut;
161   /* Time variables */
162   let timerRecursive = 5000;
163   let timerString = timerRecursive + 'ms';
164   let clickTimer = timerRecursive * 3;
165   let clickTimerString = clickTimer + 'ms';
166   function loopThrough(nClick) {
167     if(nClick != undefined) {
168       n = nClick;
169       for(let i = 0; i < sideElements.length; i++) {
170         sideElements[i].style.transition = '0ms';
171         sideElements[i].style.backgroundColor = '0% 100%';
172       }
173       midElementImg.src = sideElements[n].querySelector('img').src;
174       midElementTitle.innerHTML = sideElements[n].querySelector('.side-element-title').innerHTML;
175       midElementPrice.innerHTML = sideElements[n].getAttribute('data-price').replace('.', '&euro;');
176       midElementText.innerHTML = sideElements[n].dataset.text;
177       if(nClick != undefined) {
178         sideElements[n].style.transition = clickTimerString;
179       }
180       else {
181         sideElements[n].style.transition = timerString;
182         sideElements[n].style.backgroundColor = '100% 100%';
183         n++;
184         if(n == sideElements.length) {
185           n = 0;
186           nClick = undefined;
187         }
188       }
189       function loopThroughCall() {
190         timeOutLoop = setTimeout(loopThroughCall, timerRecursive);
191         loopThrough();
192       }
193       loopThroughCall();
194       sideElements.forEach(function(sideElement) {
195         sideElement.addEventListener('click', function() {
196           clearTimeout(timeOutLoop);
197           clearTimeout(addTimeOut);
198           for(let i = 0; i < sideElements.length; i++) {
199             if(sideElements[i].querySelector('.side-element-title') == sideElement.querySelector('.side-el-
200             loopThrough(i);
201           }
202           addTimeOut = setTimeout(loopThroughCall, clickTimer);
203         });
204       });
205     }
206   }
207   /* Category elements */
208   let upper = Array.from(document.querySelector('#category-uppen').children);
209   let lower = Array.from(document.querySelector('#category-lower').children);
210   let categoryArray = upper.concat(lower);
211   let category = document.querySelector('#category');
212   let categoryWidth = (+getComputedStyle(category).width.replace('px', ''));
213   /* Specs title and specs */
214   let specs = category.querySelector('.specs');
215   let specsWidth = (+getComputedStyle(specs).width.replace('px', ''));
```

Deklarisemo promenljivu `sideElements` koja predstavlja niz svih elemenata sa strane. Prolazimo kroz taj niz i za svaki element pojedinačno deklarismo prom `wrap` koja sadrzi HTML element(plavo na slici ispod) u koji smestamo text. Posto je element 'tesan' vodio sam racuna o njegovoj ukupnoj sirini (izracunavsi sirinu slike pored) tako da sam na liniji 114 deklariseo prom `wrapWidth` koja sadrzi njenu sirinu radi njene dalje promene u kodu.(+wrapComputed.width.replace('px', ''));



Posto su ovi elementi(`side-element`) reaktivni u odnosu na promenu sirine prozora takodje u prom `sideElementWidth` skladistimo sirinu celog elementa. Na liniji 137

Naslov elementa koji se nalazi na njegovom vrhu moze biti vece sirine od samog elementa tako da imamo prom `sideElementTitleWidth` u kojoj smestamo broj px koji naslov zauzima. Njega poredimo sa sirinom celog elementa(`sideElementWidth`) i ukoliko je prvi broj veci od drugog nad `sideElement` stavljamo `addEvenetListener` koji ce pomerati naslov u stranu kako bi ga prikazao u celosti. U CSS-u je definisano svojstvo `transition` u kome se navodi brzina njegove promene. `sideElementTitleOffset` je prom koja sadrzi razliku sirina celog elementa oca (`sideElement`) i sirinu naslova kao deteta tog elementa, I ukoliko razlika postoji `addeventlistener` se poziva. Posto se sirina celog elementa moze menjati, prilikom svake takve promene sirine se ispituju i po potrebi se `addeventlistener` uklanjanja/postavlja kako ne

bi dolazilo do gresaka da se naslov pomera a za tim nema potrebe jer se ceo vec vidi (linija 137).

Na liniji 156 se deklarise promenljiva `midElement` koja sadrzi HTML centralni element, i nakon njega na isti nacin i njegova deca elementi, `midElementImg`, `midElementTitle`, `midElementPrice` i `midElementText` kako bi u daljem kodu mogli tim elementima dodeljivati vrednosti koje cemo sakupljati sa elemenata koji ga okružuju.

Linija 160: `let timerRecursive = 5000, timerString = timerRecursive + 'ms', clickTimer = timerRecursive * 3, clickTimerString = clickTimer + 'ms'`; Ove promeljive ce se koristiti u daljem kodu za poziv metoda sa `setTimeout` kako bi neko u slucaju da zeli da promeni vremenski period poziva tih metoda mogao prostom promenom vrednosti jedne promeljive (`timerRecursive`) da promeni vremenski period poziva tih metoda po svojoj zelji, umesto da 'kopa' po kodu i vrsi brojnice promene.

Prolazi se kroz niz elemenata sa strane, svakom elementu je dodeljeno 5000ms (`timerRecursive`) da se prikaze na centralnom elementu, a prilikom klika na neki od elemenata sa strane taj period je 15000ms (*3-clickTimer).

Metoda `loopThrough` prvo prolazi kroz ceo niz elemenata i svima dodeljuje css svojstva 0 kako bi samo element koji je trenutno prikazan imao ta svojstva koja trebaju da vizuelno docaraju njegovo ucitavanje.

Imamo promenljivu `n` sa vrednosti 0 koja je globalna jer se moze menjati u daljem kodu van okvira ove metode, i ona se koristi da bi ova metoda od starta krenula da prikazuje n-ti element na centralnom. Metoda `loopThrough` moze imati parametar `nClick` prilikom njenog poziva. `nClick` je prom koja se u daljem kodu koristi za skladistenje rednog broja elementa koji je kliknut radi njegovog prikazivanja na centralnom elementu i ova metoda na startu proverava da li `nClick` postoji, ukoliko ne ona nastavlja svojim redovnim tokom, a ukoliko postoji ona nastavlja tokom od rednog broja elementa na koji smo kliknuli i nastavlja redom dalje.

Na liniji 172 se proverava da li je korisnik kliknuo na element ili metoda prosto tece svojim tokom. Ukoliko je korisnikov klik u pitanju metoda podesava vizuelni prikaz elementa sa strane na 15000ms, a ukoliko se radi o redovnom prolasku kroz niz elemenata sa strane dodeljuje 5000ms. Na kraju metode se `nClick` dodeljuje vrednost *undefined* i metoda nastavlja da se ponavlja svojim tokom redom od poslednjeg elementa koji je prosao kroz nju.

Na liniji 182 imamo rekurzivnu metodu `loopThroughCall` koja se poziva nakon odredjenog vremena i sluzi samo za pozivanje glavne metode `loopThrough`. `loopThroughCall` je redovna metoda koja se poziva iznova svakih 5s. Na liniji 187 prolazimo kroz ceo niz elemenata sa strane i svakom ponaosob dodeljujemo `addeventlistener` metodu koja se aktivira klikom na njih. Ova metoda prvo brise id prom `timeOutLoop` kako bi prekinula rekurzivno pozivanje metode i pozvala `loopThrough` metodu sa zeljenim elementom na koji smo kliknuli sto se odvija tako sto se na liniji 191 proverava da li je neki element iz niza svih elemenata jednak onome na koji smo kliknuli, ako jeste uzima se index smesten u prom `i` i poziva se metoda sa tim indexom (ona postaje onaj `nClick`). Na liniji 195 se ponovo poziva rekurzivna metoda `loopThroughCall` nakon odredjenog vremena ali takodje dodeljujuci prom `addTimeOut` id poziva te metode, kako bi se u slucaju ponovnog klika na neki od elemenata ta metoda takodje mogla prekinuti (na liniji 190).

Treci element pocetne strane

```
File Edit Selection View Go Run Terminal Help
index.html JS indexjs X # indexcss shop.html JS shopjs

JS indexjs > _
199 /* Category elements */
200 {
201   let upper = Array.from(document.querySelector("#category-upper").children), lower = Array.from(document.querySelector("#category-lower").children);
202   let categoryArray = upper.concat(lower);
203   let category = document.querySelector("#category");
204   let categoryWidth = (+getComputedStyle(category).width.replace('px', ''));
205   /* Specs title and specs */
206   let specs = category.querySelector('.specs'), specsWidth = (+getComputedStyle(specs).width.replace('px', '')), specsTitles = category.querySelectorAll('.specs-title'), specsTitleWidth = [];
207   for(let i = 0; i < specsTitles.length; i++) {
208     let enterRecall, leaveRecall, stopSlide = 0, leaveRecallDelay, n = 0;
209     function mouseEnter(widthDiff, div) {
210       mouseEnterSlide(widthDiff, div);
211     };
212     function mouseEnterSlide(widthDiff, div) {
213       if(n > widthDiff) {
214         div.style.transform = `translateX(${$n}px)`;
215         n = n + (100/widthDiff);
216       }
217       enterRecall = setTimeout(mouseEnterSlide, (1200/(-widthDiff)), widthDiff, div);
218       if(n <= widthDiff || stopSlide == 1)
219         clearTimeout(enterRecall);
220     };
221     function mouseLeave(div, widthDiff) {
222       leaveRecallDelay = setTimeout(mouseLeaveSlide, 2000, div, widthDiff);
223     };
224     function mouseLeaveSlide(div, widthDiff) {
225       div.style.transform = `translateX(${$n}px)`;
226       n = n + (100/widthDiff);
227       leaveRecall = setTimeout(mouseLeaveSlide, (1200/(widthDiff)), div, widthDiff);
228       if(n >= 0 || stopSlide == 0)
229         clearTimeout(leaveRecall);
230     };
231     specsTitleWidth[i] = (+getComputedStyle(specsTitles[i]).width.replace('px', ''));
232     if(specsTitleWidth[i] > specsWidth) {
233       specsTitles[i].addEventListener('mouseenter', function(e) {
234         clearTimeout(leaveRecallDelay);
235         stopSlide = 0;
236         mouseEnter(-(specsTitleWidth[i] - specsWidth), specsTitles[i]);
237       });
238       specsTitles[i].addEventListener('mouseleave', function(e) {
239         stopSlide = 1;
240         mouseLeave(specsTitles[i], (specsTitleWidth[i] - specsWidth));
241       });
242     }
243   }
244   /* Calculating new widths and adding/removing elements */
245   window.addEventListener('resize', function(e) {
246     /*
247     if(e.currentTarget.screen.width == e.currentTarget.screen.availWidth) {
248       console.log('fullscreen');
249     } else {
250       console.log('windowed');
251     }
252     */
253     if(categoryWidth == 840)
254     {
255       upper[3].style.display = 'block';
256       lower[3].style.display = 'block';
257       upper[2].style.display = 'block';
258       lower[2].style.display = 'block';
259     }
260     else if (categoryWidth > 630)
261     {
262       upper[3].style.display = 'none';
263       lower[3].style.display = 'none';
264       upper[2].style.display = 'block';
265       lower[2].style.display = 'block';
266     }
267     else if (categoryWidth >= 210)
268     {
269       upper[3].style.display = 'none';
270       lower[3].style.display = 'none';
271       upper[2].style.display = 'none';
272       lower[2].style.display = 'none';
273     }
274     categoryWidth = getComputedStyle(category).width;
275     categoryWidth = categoryWidth.replace('px', '');
276     categoryWidth = (+categoryWidth);
277   })
278   /* Going through array of Category elements to add click event on them */
279   for (let i = 0; i < categoryArray.length; i++) {
280     categoryArray[i].querySelector('.specs').addEventListener('click', function(e) {
281       if (categoryArray[i].querySelector('.specs').style.height == '100%')
282       {
283         categoryArray[i].querySelector('.specs').style.height = '35px';
284         categoryArray[i].querySelector('.specs').style.borderTopLeftRadius = '0px';
285         categoryArray[i].querySelector('.specs').style.borderTopRightRadius = '0px';
286       } else {
287         categoryArray[i].querySelector('.specs').style.height = '100%';
288         categoryArray[i].querySelector('.specs').style.borderRadius = '5px';
289         categoryArray[i].querySelector('.img').style.filter = 'blur(5px)';
290         /* Shuts down all other elements except one clicked */
291         for (let n = 0; n < categoryArray.length; n++) {
292           if (n != i) {
293             categoryArray[n].querySelector('.img').style.filter = 'blur(0px)';
294             categoryArray[n].querySelector('.specs').style.height = '35px';
295             categoryArray[n].querySelector('.specs').style.borderTopLeftRadius = '0px';
296             categoryArray[n].querySelector('.specs').style.borderTopRightRadius = '0px';
297           }
298         }
299       }
300     })
301   }
302 }
```



Ovaj element je zamisljen kao: element kontejner sa definisanom sirinom i visinom u koji ce se smestiti vise manjih elemenata koji ce takodje imati elemente decu koja se nalaze na njihovom dnu I prilikom klika na njih ona dobijaju punu visinu njihovih roditelja elemenata i otkrivaju vise informacija o samom *item-u*.

Ovaj element je podeljen na 2 dela: **upper** koji sadrzi 4 elementa gore i **lower** 4 elementa dole. Podeljen je na taj nacin da bi se prilikom promene sirine prozora po potrebi mogli skoloniti po 1 element sa krajeva oba niza, ali je radi odrzavanja stvari jednostavnijim deklarisan novi niz **categoryArray** koji sadrzi oba prthodna niza. Prom **categoryWidth** sadrzi broj koji predstavlja sirinu elementa oca(kontejnera), tj. celokupnog elementa.

Specs je promenljiva koja predstavlja dete element elementa iz niza **categoryArray**, posto su svi istih dimenzija uzima se samo jedna prom a ne niz svih, zatim se u promenljivoj **specsTitles** formira niz svih naslova unutar tih specs elemenata. **specsTitleWidth** je niz koji deklarise na pocetku ali ce biti koriscen u daljem kodu za skladistenje vrednosti sirina naslova kako bi mogli da proverimo da li je sirina naslova veca od sirine prostora koji ima za prikazivanje I na osnovu toga mozemo naslovima kojima je to potrebno dodeliti pomeranje teksta da bi mogao biti prikazan u potpunosti.

Na liniji 207 prolazimo kroz ceo niz **specsTitles** i u svakom deklarise promeljive **enterRecall** i **leaveRecall** koje ce biti koriscene u daljem kodu za skladistenje id poziva rekurzivnih metoda kako bi mogli po potrebi da ih 'iskljucimo'. Deklarise prom **stopSlide** koju cemo koristiti kao prekidac, da na osnovu njene vrednosti (0 ili 1) nastavljamo kod ili odustajemo. **leaveRecallDelay** je takodje promenljiva u kojoj smestamo id poziva metode.

Metoda **mouseenter** ima 2 parametra: **widthDiff** koji treba da predstavlja razliku izmedju sirine naslova I sirine prostora oca elementa i **div** kao konkretni element. Ova metoda sluzi za pozivanje druge metode **mouseenterSlide** kojoj predaje iste ove parametre. **mouseenterSlide** je rekurzivna metoda koja ciji je uslov za prekid ponovnog poziva ili da je promenljiva **n** dostigla maximum(razliku u sirinama) ili da je prekidac **stopSlide** jednak 1.

Metode **mouseleave** I **mouseleaveSlide** funkcionisu na isti nacin kao I prethodne 2 metode, prilikom svakog rekurzivnog poziva one dodeljuju CSS svojstvu broj **n** za pomeranje naslova, a svakim pozivom ga pomeraju za 1%.

Na liniji 231 dodeljujemo vrednosti u niz **specsTitleWidth** u vidu broja pixelsa sirine naslova kako bi mogli da ga poredimo u uslovu(jedna linija nize) sa sirinom prostora elementa oca tog naslova. Ukoliko je sirina naslova veca od prostora koji on ima da se prikaze onda se nad njim poziva **addevenetlistener** koji se aktivira prilikom ulaska misa u njegov prostor pri cemu se desava sledece:

234 – ukoliko se naslov vracao unazad u svoje prvobitno stanje to se zaustavlja(brise id promenljive **leaveRecallDelay**),

235 – **stopSlide** dobija vrednost 0, i

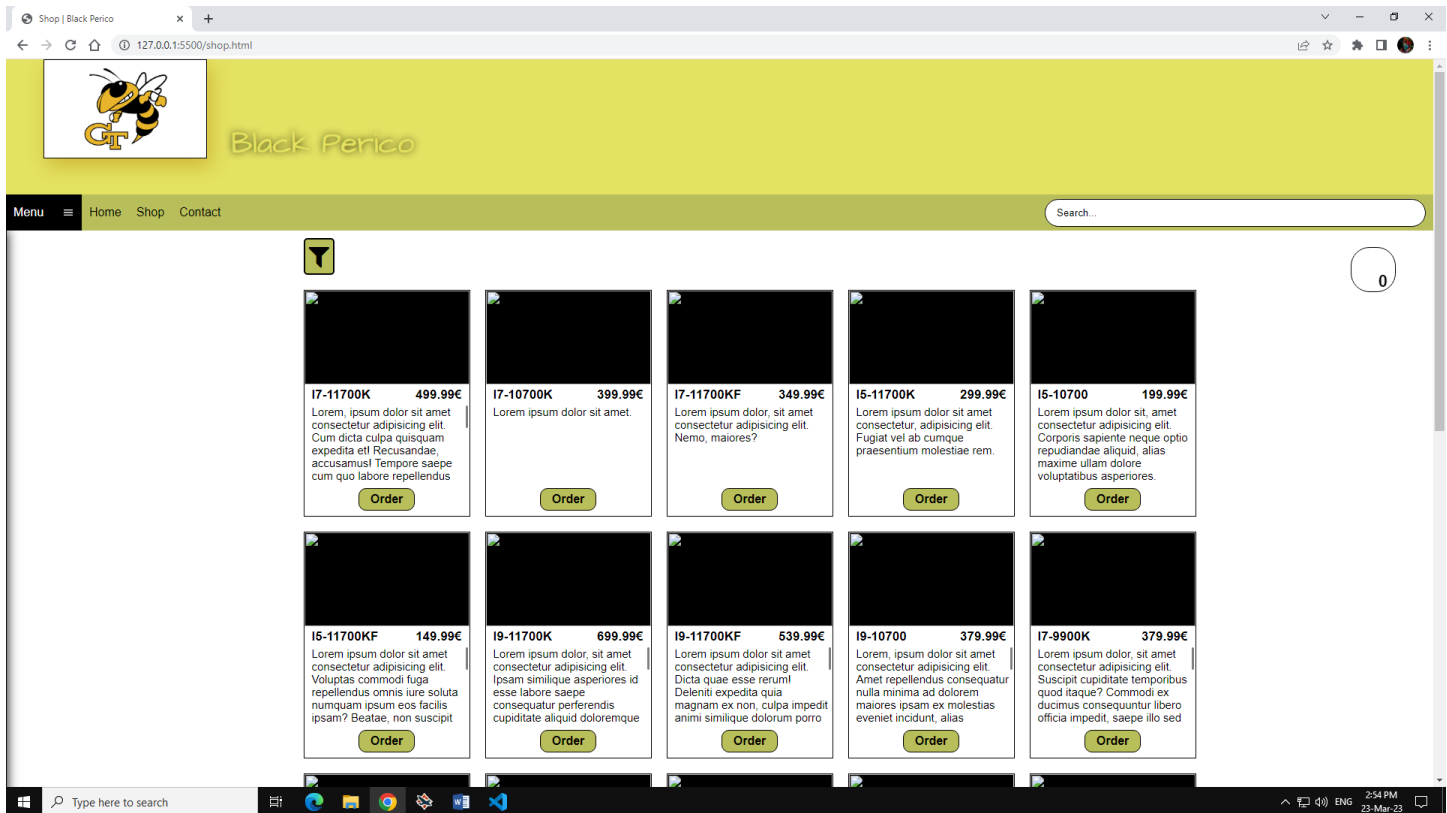
236 – poziva se metoda `mouseenter` i predaje joj se 1. rezultat oduzimanja sirine naslova od sirine prostora koji ima njegov otac element I kao 2. parametar se predaje sam HTML element naslova.

Na liniji 238 svim elementima takodje dodeljujemo i `addEventListener` koji pokrece metodu prilikom izlaska misa iz okvira elementa pri cemu se `slideUp` 'prekidac' dodeljuje vrednost 1 I poziva se funkcija `mouseleave`.

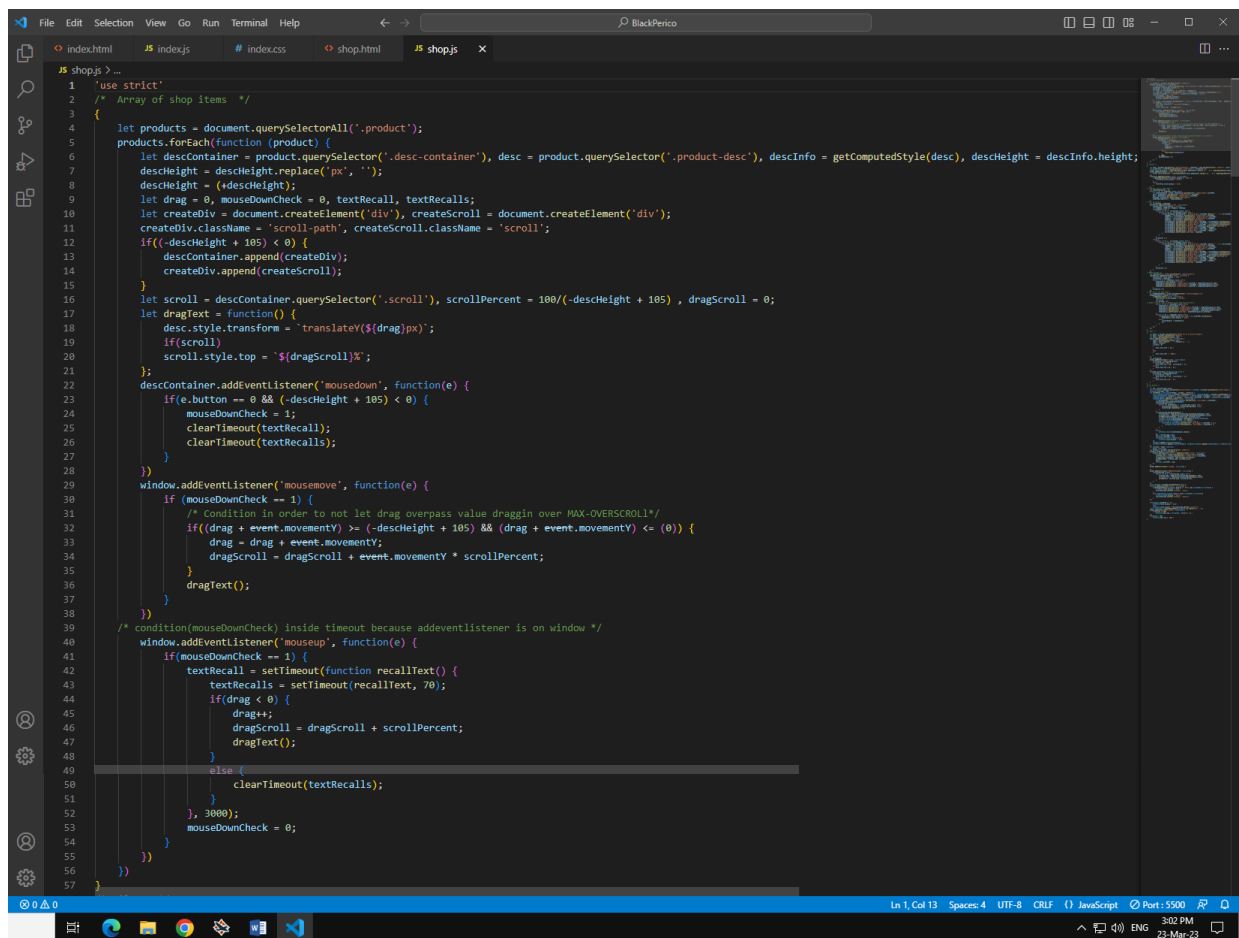
Posto je ovaj element podlozan promeni svoje sirine prilikom menjanja sirine prozora, nad `window` objektu se poziva `addEventListener` koji pokrece metodu svaki put kada se sirina prozora promeni i ova metoda ispituje da li je sirina elementa kontejnera jednaka odredjenom broju, i ukoliko jeste brisu/dodaju se zadnji elementi nizova sa gornje I donje strane.

Na liniji 278 se prolazi kroz ceo niz elemenata I na njih se kaci `addEventListener` koji na klik ovih elemenata pokrece metodu koja prvo ispituje da li je visina ovih elemenata 100% visine njihovih oceva elemenata, ukoliko jeste onda se elementi smanjuju na njihovu prvobitnu visinu(da se prikazuje samo naslov elementa), a ukoliko nije onda se visina podesava na 100% I dodaje se filter blur radi lepseg vizuelnog pristupa. Takodje se menjaju i ivice elemenata(border) kako bi se bolje uklapale. Nakon toga se takodje jos jednom prolazi kroz ceo niz, osim sto se izostavlja index elementa koji je 'selektovan' i svima se vracaju prvobitne vrednosti kako ne bi moglo da se selektuje vise od 1 elementa istovremeno.

Shop strana



Ova strana je zamisljena tako da sadrzi samo iste elemente, odredjene proizvode. Svaki element ima svoju sliku, ispod njega naziv I cenu, ispod toga text koji se nalazi u kontejner elementu sa odredjenom visinom, sto znaci da sam text moze prevazilaziti tu visinu ali kontejner ima svojstvo overflow: hidden, I na dnu dugme za kupovinu. Na stranici se takodje nalaze I filteri koji ove elemente mogu sortirati prema njihovoj ceni, ili prikazati samo odredjene elemente jer svaki od ovih elemenata(artikala) u HTML-u sadrzi atribut data-type koji oznacava da li je u pitanju CPU/GPU/Mouse/Keyboard...

A screenshot of a code editor window titled 'BlackPenCo' showing a JavaScript file named 'shop.js'. The code implements a scrollable text container. It starts with a 'use strict' directive and a comment '/* Array of shop items */'. A 'products' array is defined using 'document.querySelectorAll'. The code then iterates over each product, creating a 'descContainer' and a 'desc' element. It calculates the 'descHeight' and 'scroll' element. A 'drag' variable is initialized to 0, and a 'mouseDownCheck' is set to 0. A 'createDiv' function is defined to create a 'div' element. A 'scroll' element is created and appended to the 'descContainer'. A 'scroll' variable is calculated based on the 'descHeight' and 'scroll' element. A 'dragText' function is defined to update the 'desc' element's transform and the 'scroll' element's top position. A 'descContainer' event listener is added for 'mousedown', which sets 'mouseDownCheck' to 1, clears timeouts for 'textRecall' and 'textRecalls', and calls 'dragText'. A 'window' event listener is added for 'mousemove', which checks if 'mouseDownCheck' is 1 and if the drag is over the scroll area. If so, it updates 'drag' and 'scroll' and calls 'dragText'. A 'mouseup' event listener is added to the 'window', which clears the 'textRecalls' timeout and sets 'mouseDownCheck' back to 0. The code is written in a dark-themed editor with line numbers on the left and a status bar at the bottom showing 'Ln 1, Col 13', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', 'Port: 5500', and '3:02 PM 23-Mar-23'.

Promenljiva **products** predstavlja niz svih elemenata artikala na stranici. Prolazimo kroz niz i za svaki element pojedinačno deklariramo promenljive **descContainer** koji predstavlja element kontejner za text, **desc** koji je div element u kojem je smesten text, i **descHeight** koji sadrzi vrednost visine texta. **Drag** je promenljiva sa pocetnom vrednoscu 0 i u nju cemo smestati broj px kojim prevlacimo text kako bi ga dodelili css svojstvu koje ce pomerati text. **mouseDownCheck** je promenljiva koju koristimo kao prekidač, zato sto text koji cemo prevlaciti/skrolovati treba da nastavi sa prevlacenjem i kada je nas mis van okvira elementa artikla, I prom **textRecall** I **textRecalls** koje se koriste za skladistenje id poziva metoda.

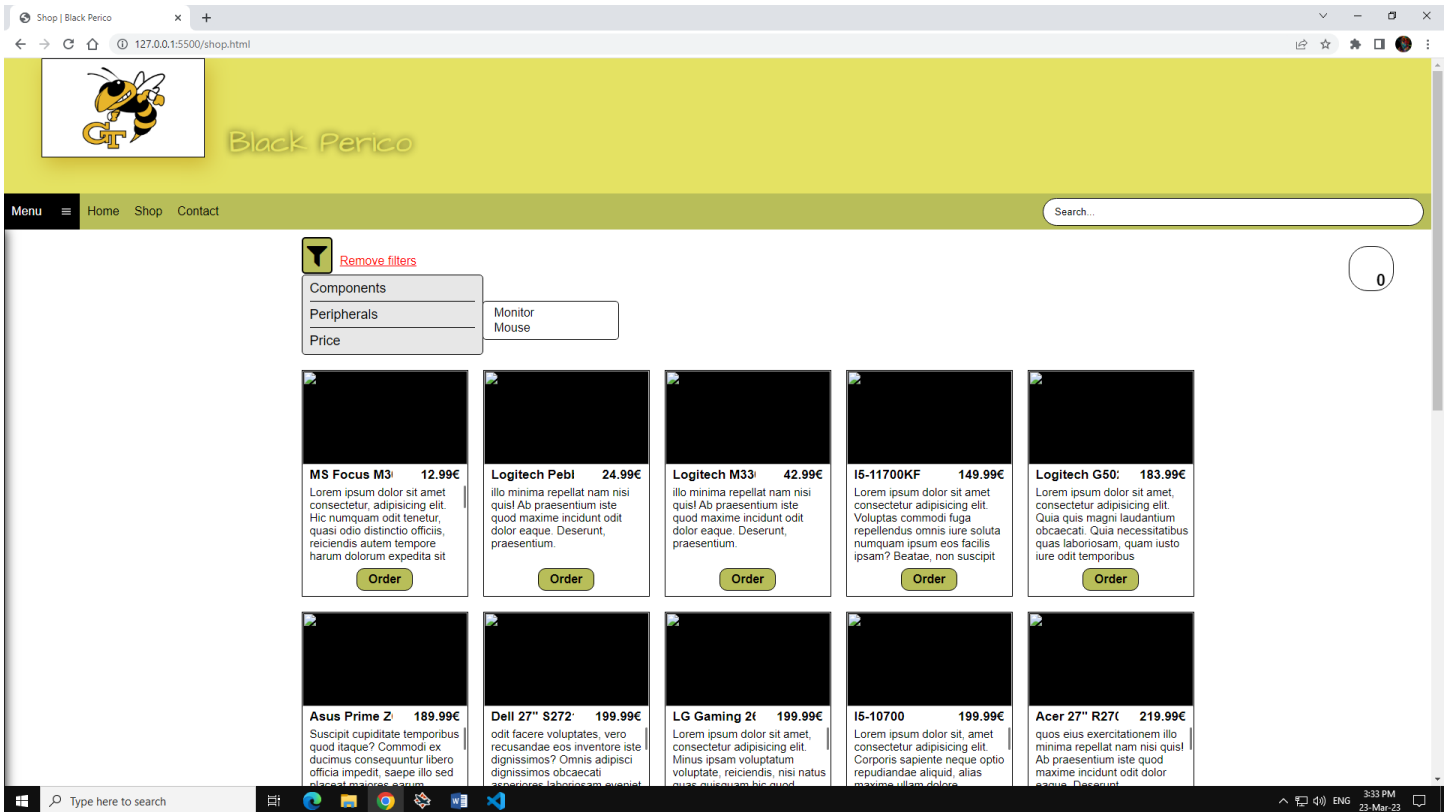
Posto znamo da je visina kontejner elementa(oca elementa) za text 105px na liniji 12 proveravamo da li visina samo texta vece od toga, ukoliko jeste kreiramo nove HTML elemente koji imaju samo vizuelni efekat.

Na liniji 17 deklariramo metodu **dragText** koja koristi globalnu promenljivu **drag** menja CSS svojstvo texta podizuci ga u gore za broj koji se nalazi u toj promenljivoj.

Dalje u kodu na liniji 22 I 29 stavljam **addeventlistener**, prvo nad elementom koji je otac elementa texta a zatim nad **window** objektom. Prvo se aktivira prilikom pritiska misa I poziva se metoda koja proverava da li je pritisnut taster levi mis I da li je visina texta vece od visine kontejnera, ukoliko jeste onda se prekidač **mouseDownCheck** postavlja na 1 i brisu se vrednosti promenljivih **textRecall** I **textRecalls** koje u daljem kodu predstavljaju id vrednosti poziva rekurzivnih metoda koje vraćaju text na podrazumevanu visinu. Drugi **addeventlistener** poziva metodu prilikom svakog pokreta misem I prvo ispituje da li je **mouseDownCheck** jednak, ukoliko jeste onda se **drag** promenljiva povećava za broj px koji smo misem prevukli, ali ne bez uslova da zbir drag i broj px koji prevlacimo ne prevazilazi visinu texta(kako bi izbegli 'overscroll'). Tako da svakim pokretom misa sve dok su svi uslovi zadovoljeni poziva se metoda koja menja CSS svojstvo i podize text, metoda koristi globalnu promenljivu tako da nije potrebno predavati joj parametre prilikom poziva vec je samo pozvati.

Takodje imamo i 3. `addeventlistener` koji se takodje poziva nad objektom `window` i on prilikom svakog otpustanja tastera misa pokrece metodu koja prvo proverava da li je `mouseDownCheck` jednak jedinici, i ukoliko jeste poziva se rekurzivna metoda i id njenog poziva se smesta u prom `textRecall`, a njeno dalje rekurzivno pozivanje smesta id tog poziva u `textRecalls`. Metoda `recallText` ima svrhu da text koji smo scrollovali vrati na podrazumevanu visinu, u prvobitno stanje, postepeno. To izvorsava tako sto ispituje da li je vrednost `drag` manja od 0, kako ne bi text podigli vise nego potrebno, i ukoliko jeste, svakih 70ms se podize za 1px, ukoliko nije rekurzivni pozivi smesteni u `textRecalls` se brisu. Na kraju se `mouseDownCheck` dodeljuje vrednost 0 sto oznacava da nije kliknuto. A `recallText` se poziva prvo nakon 3s od poslednjeg klika nakon cega slede rekurzivni pozivi svakih 70ms.

Filteri druge strane



Ovaj element prilikom klika na njega otvara listu: komponente, periferije i price. Prilikom klika na komponente otvara se dodatna lista koja nudi CPU/GPU/Motherboard i periferije koje nude Monitor/Mouse, i klikom na price automatski se ceo niz artikala sortira prvo po rastuocj ceni pa sledecim klikom po opadanju. Daljim klikom na Monitor/Mouse svi artikli koji ne sadrže ove nazive u HTML atributu data-type dobijaju CSS svojstvo display jednaku 'none' i prikazuju se samo zeljeni artikli. Takodje prilikom bilo kakvog sortiranja niza artikala, pored ikonice za filtere stvara se novi text 'Remove filters' cijim klikom ceo sortirani niz vracamo u prvobitno stanje i text nestaje.

```
File Edit Selection View Go Run Terminal Help
shop.html shop.js x
shop.js > ...
58 /* Filters */
59 {
60   let shop = document.querySelector('#shop-container'), shopItems = shop.querySelectorAll('.product'), filter = document.querySelector('#filter-container'), filterDrop = filter.querySelector('#filters'), filterIcon
61   let priceFilter = filterDrop.querySelector('.price-filter'), sortSwitch = 0;
62   filter.style.marginLeft = (+getComputedStyle(shop).paddingLeft.replace('px', '')) + (+getComputedStyle(shop).marginLeft.replace('px', '')) + 'px';
63   window.addEventListener('resize', function(e) {
64     filter.style.marginLeft = (+getComputedStyle(shop).paddingLeft.replace('px', '')) + (+getComputedStyle(shop).marginLeft.replace('px', '')) + 'px';
65   });
66   filterIcon.addEventListener('click', function(e) {
67     if(getComputedStyle(filterDrop).display == 'none') {
68       filterDrop.style.display = 'block';
69     }
70     else {
71       filterDrop.style.display = 'none';
72     }
73   })
74   /* Original array of shop items */
75   let shopItemsOriginal = [];
76   for(let i = 0; i < shopItems.length; i++) {
77     let shopItemsObject = {name: shopItems[i].querySelector('.product-name').innerHTML,
78     price: shopItems[i].querySelector('.product-price').innerHTML,
79     desc: shopItems[i].querySelector('.product-desc').innerHTML,
80     dataType: shopItems[i].getAttribute('data-type')}
81     shopItemsOriginal[i] = shopItemsObject;
82   }
83   /* Sort by price */
84   let currentShop = shopItems;
85   priceFilter.addEventListener('click', function() {
86     removeFilters.style.display = 'inline';
87     let tempName, tempPrice, tempDesc, tempType;
88     if(sortSwitch == 0) {
89       for(let i = 0; i < currentShop.length; i++) {
90         for(let n = 0; n < currentShop.length; n++) {
91           if(+currentShop[i].querySelector('.product-price').innerHTML.replace('€', '') < +currentShop[n].querySelector('.product-price').innerHTML.replace('€', '')) {
92             tempName = currentShop[i].querySelector('.product-name').innerHTML;
93             tempPrice = currentShop[i].querySelector('.product-price').innerHTML;
94             tempDesc = currentShop[i].querySelector('.product-desc').innerHTML;
95             tempType = currentShop[i].getAttribute('data-type');
96             currentShop[i].querySelector('.product-name').innerHTML = currentShop[n].querySelector('.product-name').innerHTML;
97             currentShop[i].querySelector('.product-price').innerHTML = currentShop[n].querySelector('.product-price').innerHTML;
98             currentShop[i].querySelector('.product-desc').innerHTML = currentShop[n].querySelector('.product-desc').innerHTML;
99             currentShop[i].setAttribute('data-type', currentShop[n].getAttribute('data-type'));
100             currentShop[n].querySelector('.product-name').innerHTML = tempName;
101             currentShop[n].querySelector('.product-price').innerHTML = tempPrice;
102             currentShop[n].querySelector('.product-desc').innerHTML = tempDesc;
103             currentShop[n].setAttribute('data-type', tempType);
104           }
105         }
106       }
107       sortSwitch = 1;
108     } else {
109       for(let i = 0; i < currentShop.length; i++) {
110         for(let n = 0; n < currentShop.length; n++) {
111           if(+currentShop[i].querySelector('.product-price').innerHTML.replace('€', '') > +currentShop[n].querySelector('.product-price').innerHTML.replace('€', '')) {
112             tempName = currentShop[i].querySelector('.product-name').innerHTML;
113             tempPrice = currentShop[i].querySelector('.product-price').innerHTML;
114             tempDesc = currentShop[i].querySelector('.product-desc').innerHTML;
115             tempType = currentShop[i].getAttribute('data-type');
116             currentShop[i].querySelector('.product-name').innerHTML = currentShop[n].querySelector('.product-name').innerHTML;
117             currentShop[i].querySelector('.product-price').innerHTML = currentShop[n].querySelector('.product-price').innerHTML;
118             currentShop[i].querySelector('.product-desc').innerHTML = currentShop[n].querySelector('.product-desc').innerHTML;
119             currentShop[i].setAttribute('data-type', currentShop[n].getAttribute('data-type'));
120             currentShop[n].querySelector('.product-name').innerHTML = tempName;
121             currentShop[n].querySelector('.product-price').innerHTML = tempPrice;
122             currentShop[n].querySelector('.product-desc').innerHTML = tempDesc;
123             currentShop[n].setAttribute('data-type', tempType);
124           }
125         }
126       }
127       sortSwitch = 0;
128     }
129   })
130   /* Remove filters */
131   let removeFilters = filter.querySelector('.remove-filters');
132   removeFilters.addEventListener('click', function() {
133     removeFilters.style.display = 'none';
134     currentShop = shopItems;
135     for(let i = 0; i < shopItems.length; i++) {
136       shopItems[i].style.display = 'block';
137       shopItems[i].querySelector('.product-name').innerHTML = shopItemsOriginal[i].name;
138       shopItems[i].querySelector('.product-price').innerHTML = shopItemsOriginal[i].price;
139       shopItems[i].querySelector('.product-desc').innerHTML = shopItemsOriginal[i].desc;
140     }
141     sortSwitch = 0;
142   });
143   /* Set filters for data-type */
144   let dropdownElements = filter.querySelectorAll('.filters-dropdown li');
145   dropdownElements.forEach(function(e) {
146     e.addEventListener('click', function() {
147       removeFilters.style.display = 'inline';
148       let x = 0;
149       currentShop = [];
150     });
151     /* Resets array of items to default values from price-sorting */
152     for(let i = 0; i < shopItems.length; i++) {
153       shopItems[i].style.display = 'block';
154       shopItems[i].querySelector('.product-name').innerHTML = shopItemsOriginal[i].name;
155       shopItems[i].querySelector('.product-price').innerHTML = shopItemsOriginal[i].price;
156       shopItems[i].querySelector('.product-desc').innerHTML = shopItemsOriginal[i].desc;
157       shopItems[i].setAttribute('data-type', shopItemsOriginal[i].dataType);
158     }
159     for(let i = 0; i < shopItems.length; i++) {
160       if(shopItems[i].getAttribute('data-type') != e.innerHTML.toLowerCase())
161         shopItems[i].style.display = 'none';
162       else {
163         currentShop[x] = shopItems[i];
164         x++;
165       }
166     }
167   });
168 }
```

Promenljiva **shop** sadrži HTML element koji je otac element svim artiklima-elementima, **shopItems** niz svih artikala-elemenata, **filter** predstavlja element oca svih elemenata koji proizilaze iz filtera, **filterDrop** predstavlja listu: komponente, periferije, cena kao i njihove podliste, **priceFilter** predstavlja zadnji element liste.

Linija 134 je greska koja mi se provukla koju tek sad vidim.

Posto `filter` nema CSS svojstvo koje bi odredilo njegov položaj u odnosu na kontejner artikala elemenata, njegov položaj određujemo u JS tako što je njegova leva margina jednaka broju levog paddinga + leva margina `shopa`, i da bi zadržao taj položaj prilikom svake promene širine prozora `addeventlistener` poziva metodu koja ponovo na isti način izračunava levu marginu filtera.

Na liniji 66 kažemo da prilikom klika na ikonicu filtera prikazujemo ili sakrivamo listu filtera u zavisnosti od njegove trenutne vrednosti CSS svojstva `display`.

Na liniji 75 deklarismo niz `shopItemsOriginal` koji sadrži podrazumevani sled artikala-elemenata kako bi u daljem kodu mogli prilikom klika na 'remove filters' da vratimo ceo niz artikala u predjasnje stanje.

Na liniji 84 deklarismo `currentShop` promenljivu i dodeljujemo joj trenutni niz artikala-elemenata koji zatim koristimo u liniji ispod, gde zadnjem elementu iz liste filtera('price') kacimo `addeventlistener` koji pri kliku na njega poziva metodu koja prolazi kroz ceo trenutni niz `currentShop` i sortira ga po opadajućem/rastućem redosledu. U kom smeru sortira niz zavisi od promenljive `sortSwitch`, jednom ce ga sortirati u opadajućem redosledu sledeći put u rastućem. `currentShop` promenljiva je potrebna jer niz koji sortiramo po ceni može prethodno već biti filtriran, npr. da se prikazuju samo artikli procesora, ili samo misevi.

Sortiranje niza po ceni se vrši tako da se zapravo samo vrednosti elemenata-artikala koji se porede menjaju unutar tih elemenata, ne menjaju se zapravo cvorovi, što bi možda bilo i bolje rešenje, ali i ovakav pristup ne pravi probleme prilikom daljeg korišćenja sajta i kodiranja.

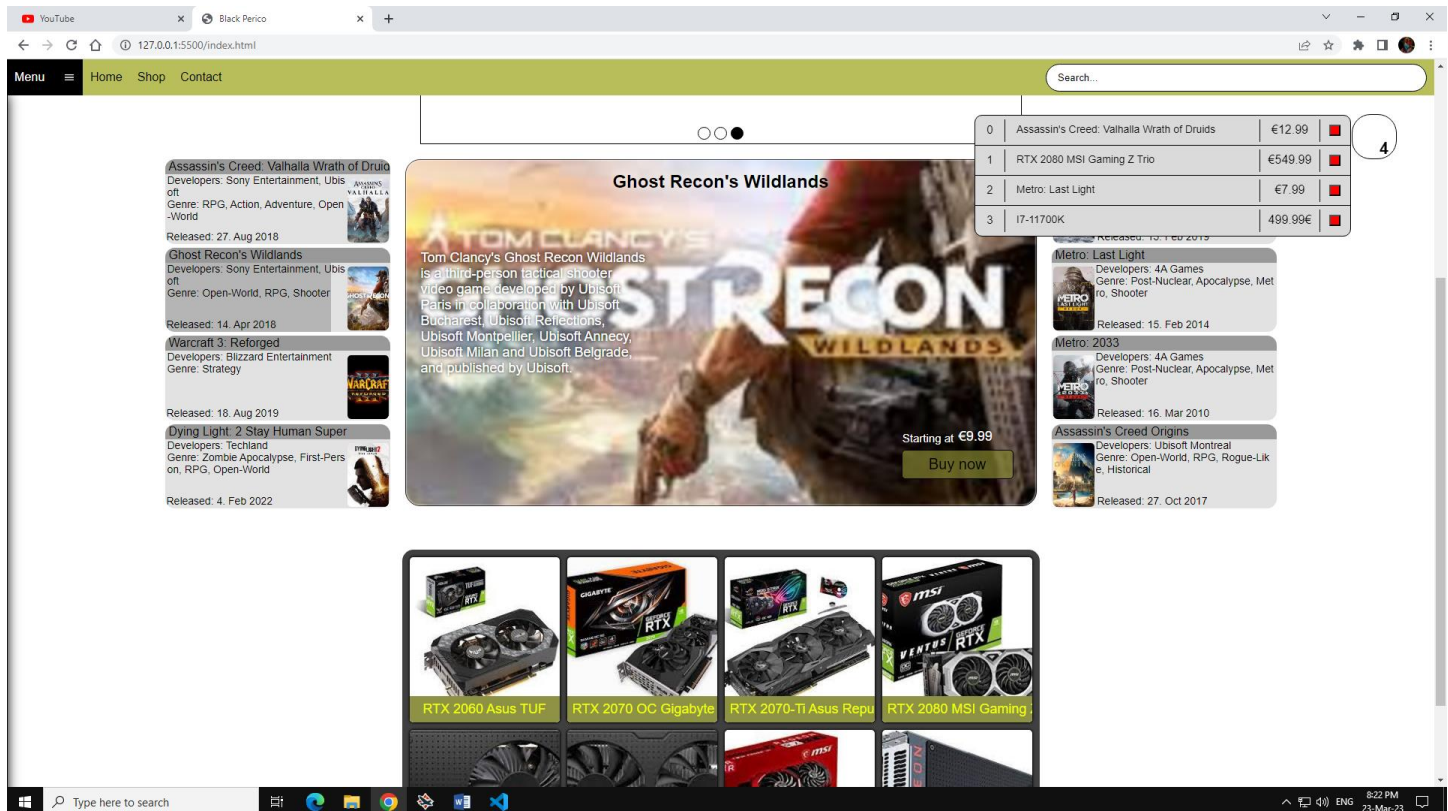
Na liniji 131 selektujemo element sa 'Remove filters' textom i dodeljujemo ga promenljivoj `removeFilters`. Zatim na njega stavljamo `addeventlistener` koji prilikom klika na taj element poziva metodu koja sakriva taj element, prolazi kroz ceo(izvorni) niz i dodeljuje svakom CSS svojstvu `display` elementa vrednost `block` kako bi ih prikazala i svaki element redom unutar sebe dobija vrednosti koje smo prethodno smestili u niz `shopItemsOriginal` čime svi artikli dobijaju izvorne vrednosti.

Na liniji 143 i ispod 'hvatamo' svaki element iz liste filtera(CPU/GPU/Motherboard/Mouse/..), zatim prolazimo kroz taj niz elemenata i kažemo da prilikom klika na njih prikazujemo 'Remove filters' jer ćemo upravo napraviti filtraciju niza elemenata-artikala, zatim prolazimo kroz celi niz artikala vraćajući ga na podrazumevani oblik. Nakon toga još jednom prolazimo kroz celi niz artikala gde ispitujemo da li je naziv elementa na koji smo kliknuli jednak nazivu atributa `data-type` artikla-elementa iz ovog niza artikala. Ukoliko su nazivi različiti taj element se sakriva tako što se vrednost njegovog CSS svojstva `display` podesava na 'none' a ukoliko su nazivi elementa iz liste i atribut HTML elementa isti onda se unutar `currentShop` niza smesta taj element. U tome je primena `currentShop` u ranijem delu koda, kako bi se sada po ceni mogli sortirati samo elementi koji nas interesuju(elementi iz `currentShop`).

Kupovina artikala sa obe strane

Na sajtu, na obe strane nalaze se razni proizvodi predstavljeni kroz drugacije dizajnirane elemente. Svaki od tih elemenata ima dugme za kupovinu. Moja ideja je bila da se pritiskom na njih negde ucitaju podaci o tom elementu(proizvodu) koji zelimo da kupimo, a da budu sacuvani prilikom navigacije kroz sajt i njene stranice. Zbog toga sam se odlucio da ove podatke smestam u **localStorage**.

Takodje je bilo neophodno da prikaz proizvoda koji zelimo da kupimo smestimo u neki element, u korpu, a da se ta korpa puni prvo elementima samim klikom na dugmad za kupovinu, ali prilikom ucitavanja druge strane da se ucitaju u istu korpu-element iz **localStorage**.



U gornjem desnom uglu vidimo jedan element u kojem se nalazi broj, klikom na taj element treba da se otvori prozor(njegov rodjak-element) sa njegove leve strane u kome se nalaze svi proizvodi koje zelimo da kupimo. Bilo bi lepse kad bi taj element u sebi sadrzao neku SVG ikonice kao sto je to bio i slucaj kod ikonice za filter. Broj unutar ovog elementa treba da prikazuje ukupan broj elemenata koji se nalazi u njegovom elementu-rodjaku. Ukoliko je taj broj 0, tj nema elemenata u drugom bloku, taj blok treba sam da se sakrije. Takodje kada kliknemo na element sa brojem u njemu, u **sessionStorage** pod odredjenim kljucem skladistimo vrednost 'block' ili 'none' i prilikom prelaska na drugu stranu blok sa proizvodima u njemu ostaje zatvoren/otvoren, na isti nacin kako smo stranicu i napustili.

Malo je nesrecno sto indexiranje proizvoda u cart-bloku pocinje od 0 umesto od 1, ali zbog zurbe sa vremenom nisam menjao u kodu svaki prolazak kroz niz, kroz localStorage, I kroz ove elemente...

```
File Edit Selection View Go Run Terminal Help
index.html index.js index.css shop.html shop.js

index.js > ...
301 /* Buy buttons */
302 {
303   let key = localStorage.length;
304   const cartBlock = document.querySelector("#cart-block"), cartCount = document.querySelector("#cart-items");
305   cartCount.innerHTML = key;
306   let cartModify = function(cartNumber, cartDesc, cartPrice) {
307     const createCartContainer = document.createElement('div'), createCartNumber = document.createElement('div'), createCartDesc = document.createElement('div'), createCartPrice = document.createElement('div'), createCartButton = document.createElement('div');
308     createCartContainer.className = 'cart-container', createCartNumber.className = 'cart-number', createCartDesc.className = 'cart-description', createCartPrice.className = 'cart-price', createCartButton.className = 'cart-button';
309     createCartNumber.innerHTML = cartNumber, createCartDesc.innerHTML = cartDesc, createCartPrice.innerHTML = cartPrice;
310     createCartButton.addEventListener('click', function(e) {
311       let deletedIndex = +(this.parentElement.querySelector('.cart-number').innerHTML);
312       localStorage.removeItem(deletedIndex);
313       if(localStorage.length) {
314         for(let i = deletedIndex; i < localStorage.length; i++) {
315           localStorage.setItem(i, localStorage.getItem(i + 1));
316           localStorage.removeItem(i + 1);
317         }
318       }
319       if(localStorage.key(deletedIndex)) {
320         cartItem.name = JSON.parse(localStorage.getItem(deletedIndex)).name;
321         cartItem.price = JSON.parse(localStorage.getItem(deletedIndex)).price;
322         cartModify(deletedIndex, cartItem.name, cartItem.price);
323         cartBlock.children[deletedIndex].replaceWith(cartBlock.lastChild);
324         cartBlock.children[deletedIndex + 1].remove();
325         for(let i = 0; i < cartBlock.children.length; i++) {
326           if(cartBlock.children[i].querySelector('.cart-number').innerHTML != i) {
327             cartBlock.children[i].querySelector('.cart-number').innerHTML = i;
328           }
329         }
330       }
331       else {
332         cartBlock.children[deletedIndex].remove();
333         key = localStorage.length;
334         cartCount.innerHTML = key;
335         if(cartBlock.children.length == 0) {
336           cartBlock.style.display = 'none';
337         }
338         cartBlock.append(createCartContainer);
339         cartBlock.lastChild.append(createCartNumber, cartBlock.lastChild.append(createCartDesc), cartBlock.lastChild.append(createCartPrice), cartBlock.lastChild.append(createCartButton));
340       }
341       let cartItem = {name: undefined,
342         price: undefined};
343       let catMiddle = document.querySelector("#cat-middle"), catMiddleBuy = catMiddle.querySelector('.buy-button');
344       catMiddleBuy.addEventListener('click', function() {
345         cartItem.name = catMiddle.querySelector('.cat-title').innerHTML;
346         cartItem.price = catMiddle.querySelector('.cat-price').innerHTML;
347         window.localStorage.setItem(key, JSON.stringify(cartItem));
348         cartModify(key, cartItem.name, cartItem.price);
349         key++;
350         cartCount.innerHTML = key;
351       })
352       let categoryItems = document.querySelectorAll('.category-item');
353       categoryItems.forEach(function(item) {
354         let button = item.querySelector('.buy-button');
355         button.addEventListener('click', function() {
356           cartItem.name = item.querySelector('.specs-title').innerHTML;
357           cartItem.price = item.getAttribute('data-price');
358           window.localStorage.setItem(key, JSON.stringify(cartItem));
359           cartModify(key, cartItem.name, cartItem.price);
360           key++;
361           cartCount.innerHTML = key;
362         })
363       });
364       window.addEventListener('storage', function(e) {
365       });
366       window.addEventListener('DOMContentLoaded', function(e) {
367         if(localStorage.length) {
368           for(let i = 0; i < localStorage.length; i++) {
369             cartItem.name = JSON.parse(localStorage.getItem(i)).name;
370             cartItem.price = JSON.parse(localStorage.getItem(i)).price;
371             cartModify(i, cartItem.name, cartItem.price);
372           }
373         }
374       });
375       const cartIcon = document.querySelector("#cart");
376       cartIcon.addEventListener('click', function(e) {
377         if(getComputedStyle(cartBlock).display == 'block' && e.target == cartIcon) {
378           cartBlock.style.display = 'none';
379           sessionStorage.setItem('carticon', 'none');
380         }
381         else if(cartBlock.children.length > 0 && e.target == cartIcon){
382           cartBlock.style.display = 'block';
383           sessionStorage.setItem('carticon', 'block');
384         }
385       });
386       if(cartCount.innerHTML == '0') {
387         cartBlock.style.display = 'none';
388       }
389       else {
390         cartBlock.style.display = sessionStorage.getItem('carticon');
391         const currentTop = +(getComputedStyle(cartIcon).top.replace('px', ''));
392         window.addEventListener('scroll', function(e) {
393           if(scrollY <= 100) {
394             cartIcon.style.top = currentTop - scrollY + 'px';
395           }
396           if(scrollY > 100) {
397             cartIcon.style.top = '70px';
398           }
399         });
400       }
401     }
402   }
403 }
```

Prvo deklarismo prom **key** koja je jednaka ukupnoj duzini **localStorage**-a kako prilikom kupovine u **localStorage**-u ne bi dolazilo do toga da se neki artikal postavlja na key koji je vec postojan.

cartBlock predstavlja konstantu koja sadrzi kontejner element u koji se smestaju deca elementi koja sadrže podatke o proizvodima koje zelimo da kupimo. **cartCount** je konstanta koja sadrzi HTML element pasus koji u daljem kodu koristimo za promenu broja artikala koje smestamo u 'korpu', taj broj prilikom ucitavanja treba da bude jednak **key** promenljivoj kako bi u svakom trenutku imali tacan broj.

`cartModify` je metoda koja se koristi za kreiranje podelemenata unutar `cartBlock`-a, znaci ona kreira HTML elemente sa podacima iz artikala koje zelimo da kupimo. Tako da je ovoj metodi neophodno dostaviti parametre o indexu(rednom broju) elementa koji zelimo da kupimo, samo ime proizvoda, njegovu cenu, kao I na kraju da se kreira dugme koje ce u daljem kodu biti opisano kao dugme za prisanje tog elementa.

Vec na liniji 310 postavljamo da se prilikom kreiranja dugmeta takodje definise i `addEventListener` koji poziva metodu prilikom klika koja radi sledece:

1. deklarise se promenljiva sa nazivom `deletedIndex` u koju se smesta redni broj elementa iz korpe koji zelimo da obrisemo,
2. iz `localStorage` se brise objekat sa tim indexom,
3. proverava se da li je sada `localStorage` prazan i ukoliko nije prolazi se kroz niz objekata unutar `localStorage`-a, ali pocevsi od broja(indexa) koji je obrisan kako bi svim narednim objektima smanjili vrednost kljuc za 1 kako bi dalje u isti taj niz mogli nesmetano da dodajemo artikle redom. Jer npr. ako imam 4 objekta u `localStorage`, obrisemo objekat sa kljucem 2, kljucevi ce biti 1,3,4 i prilikom prolaska kroz ceo niz `localStorage`-a dobicemo error za index 2, tako da se svi kljucevi od onog obrisanog pomeraju za 1 broj nazad,
4. zatim se proverava da li u `localStorage`-u postoji objekat sa kljucem ciji je broj `deletedIndex`, ovo proveravam jer ukoliko idalje postoji objekat sa tim brojem kljuc to znaci da je na njegovo mesto vec dosao neki drugi objekat, ili preciznije **objekat koji smo obrisali iz `localStorage`-a nije poslednji**. Ukoliko takav objekat postoji, tj. na mesto izbrisanog objekta je vec dosao neki drugi objekat, onda kupi vrednost imena i cene tog objekta, poziva se `cartModify` kojoj se predaju ti parametri imena I cene ali prvo rednog broja, jer redni broj tog elementa koji se ponovo dodaje postaje redni broj `deletedIndex`, `cartModify` dodaje takav element na poslednje mesto, ali mi ga zamenjujemo sa elementom koji se sad nalazi na redni broj `deletedIndex` I brisemo naredni element. Ukoliko je obrisani objekat iz `localStorage`-a poslednji u nizu, tj. ne postoji nijedan objekat sa tim kljucem sada, onda se prosto brise poslednji HTML element iz `cartBlock`-a jer on nesumnjivo predstavlja proizvod koji smo obrisali I iz `localStorage`-a,
5. zatim se u `for` petlji na liniji 325 prolazi kroz ceo niz podelemenata elementa `cartBlock` i ukoliko redni broj artikala tamo ne odgovara promenljivoj `i` koja ide redom od 0 +, onda njihov redni broj postaje `i`,
6. `key` promenljiva dobija novu vrednost nakon brisanja jer je sada niz `localStorage`-a manji za 1 i isto tako se menja broj o ukupnim artiklima u `cartBlock`-u koji sad postaje jednak `key`-u, i
7. na kraju se proverava da li je sada nakon brisanja objekta iz `localStorage`-a I brisanja istog artikla(HTML elementa) iz `cartBlock`-a, postoji idalje dete element unutar `cartBlock`-a, ukoliko ne postoji onda se `cartBlock` sakriva.

Na liniji 341 deklarismo objekat `cartItem` sa promenljivama `name` i `price` koji u daljem kodu koristimo kako bi koristeci se ovim objektom postavili u `localStorage` ime i cenu proizvoda koristeci `JSON.stringify`, i takodje prebacivati iz `localStorage` u `cartBlock` podelemente koristeci `JSON.parse`.

Na linijama od 343 do 351 'hvatam' centralni element u kome se nalaze podaci o proizvodu, postavljam `addEventListener` da klikom na njegovo dugme u objekat `cartItem` smestam podatke o imenu i ceni tog proizvoda koje zatim smestam u `localStorage` te odma pozivam i metodu `cartModify` predajuci joj iste parametre kako bi odmah kreirao u `cartBlock` artikle kao podelemente. Zatim se globalna promenljiva `key` inkrementira i broj koji treba da prikazuje ukupan broj elemenata u korpi se povecava takodje i postaje jednak `key` promenljivoj.

Od linije 352 do 363 'hvatam' elemente koji predstavljaju artikle i na isti nacin njihovim dugmetima dodajem `addEventListener` koji se ponasa skroz isto kao gore navedeno.

Na liniji 367 oznacavam da kada se DOM sadrzaj ucita proverava se da li je `localStorage` prazan, ukoliko nije prolazi se kroz ceo niz objekata u njemu i vrednosti svih tih objekata se predaju kao parametri metodi `cartModify` koja ‘prenosi’ sve te objekte kao HTML elemente unutar `cartBlock`-a.

Na liniji 376 u konstantu `cartIcon` smestam HTML element, koji se u slici gore vidi kao krug sa brojem unutra. Nad njime stavljam `addEventListener` koji prilikom klika poziva metodu koja ispituje vrednost CSS svojstva `display` kako bi, ukoliko je `cartBlock` prikazan da ga sakrije, i obrnuto, odmah se ta vrednost takodje sacuva u `sessionStorage` pod odredenim kljucem kako bi prilikom ucitavanja druge stranice `cartBlock` ostao sakriven/prikazan(linija 388-391). Takodje se u uslovu proverava da li je kliknuti element jednak konstanti `cartIcon`, jer unutar HTML-a ovaj element predstavlja element oca svim drugim elementima vezanim za ‘korpu’ tako da bi se izbeglo da npr. klikom na dugme za brisanje elementa(sto se vidi na slici) ne bi pokretala `addEventListener` metoda.

Na liniji 392 i ispod se nalazi kod koji sluzi da daje celokupnom elementu ‘korpi’ fiksiranu poziciju, iako je u CSS-u oznaceno da njegova pozicija i jeste fiksirana, potrebno je odvojiti ‘korpu’ od elementa navigacije koja takodje ima fiksiranu poziciju u CSS-u pa se zbog izbegavanja da se ova 2 elementa preklapaju racuna scroll vrednost I ‘korpi’ dodaje potrebna margina gore. Linije 397 i 398 sluze cisto da se u slucaju osvezavanja strane ‘korpa’ ne bi ponovo nasla preklopljena sa navigacijom