

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR,  
INFORMATIKA SZAK**



**SAPIENTIA**  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM

Katalógus kereső bővítmény

**DIPLOMADOLGOZAT**

Témavezető:  
dr. Jánosi-Rancz Katalin Tünde,  
Adjunktus

Végzős hallgató:  
Tamás Szabolcs-Huba

**2022**

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,  
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA  
SAPIENTIA

Extensia de căutare a catalogului

LUCRARE DE DIPLOMĂ

Coordonator științific:  
dr. Jánosi-Rancz Katalin Tünde,  
Lector

Absolvent:  
Tamás Szabolcs-Huba

2022

**SAPIENTIA HUNGARIAN UNIVERSITY OF  
TRANSYLVANIA  
FACULTY OF TECHNICAL AND HUMAN SCIENCES  
COMPUTER SCIENCE SPECIALIZATION**



**SAPIENTIA**  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA

Catalog search extension

**BACHELOR THESIS**

Scientific advisor:  
dr. Jánosi-Rancz Katalin Tünde,  
Lecturer

Student:  
Tamás Szabolcs-Huba

**2022**

# Kivonat

Dolgozatom témája egy olyan bővítmény létrehozása volt, aminek a lényege, hogy egy olyan termékszámot kereshessünk a "CATALOG SEARCH" alkalmazásban, amit a felhasználó böngészőjében kiválaszthat majd jobb klikk esetén érje el a létrehozott bővítményt, amire rákattintva az alkalmazásba viszi a felhasználót, ahol hasonló termékek lesznek, mint amit kiválasztott.

Természetesen ezt úgy is meg lehet oldani, hogy a felhasználó egyik oldalon megjegyzi a termékkódot, majd bemegy az alkalmazásba, beírja a számot és meg is kapja az eredményeket. Ezzel az a probléma, hogy sokkal több időt igényel, mint az a bővítmény amit megalkottam.

Vannak olyan helyzetek az életben, amikor fontos, hogy ne csak pontosan, hanem a lehető leggyorsabban is meg tudjuk oldani ezeket a dolgokat, tehát számít az időtényező is. Dolgozatomban ez a cél volt kitűzve magam elé, hogy megkönnyítsem az alkalmazás felhasználóinak életét, illetve minél gyorsabban el tudják érni azt amire kíváncsiak. A dolgozat céljai között megemlítendő a bővítmény legfontosabb előnyei, emellett a felhasznált technológiákra is kitértem, mindent alaposan elemeztem, hogy miért érdemes ezeket használni. A céloom eléréséhez az internetet használtam, aminek a segítségével érdekes adatokra tettem szert.

Jelen dolgozatomban arra a kérdésre keresem a választ, hogy hogyan lehet kihasználni a bővítmények által kínált lehetőségeket, hol és hogyan kell tárolni a felhasználók adatait, miért használjunk localStoraget. Ezt a tulajdonságot használtam adatbázis helyett, amin keresztül tudtam belépni az alkalmazásba. Csak azok a felhasználók tudják használni ezt a bővítményt akiknek már van fiókjuk.

# Rezumat

Tema principală al lucrării mele a fost crearea unei extensii, a cărei esență este căutarea unui număr de produs în aplicația „CATALOG SEARCH”, pe care îl puteți selecta în browserul utilizatorului.

Bineînțeles, acest lucru poate fi rezolvat și prin memorarea de către utilizator a codului produsului pe o singură pagină, apoi introducerea aplicației, introducerea numărului și obținerea rezultatelor. Problema cu aceasta este că durează mult mai mult timp decât pluginul pe care l-am creat.

Există situații în viață în care este important să putem rezolva aceste lucruri nu doar cu precizie, ci cât mai repede posibil, așa că contează și factorul timp. În lucrarea mea, acest obiectiv mi-a fost stabilit pentru a ușura viața utilizatorilor aplicației și pentru a realiza cât mai repede ceea ce îi interesează. Dintre scopurile pluginului trebuie menționate cele mai importante avantaje ale extinderii, în plus, am acoperit și tehnologiile folosite, de ce merită să le folosesc. Pentru a-mi atinge scopul, am folosit internetul pentru a obține date interesante.

În această lucrare, caut răspunsul la întrebarea cum să profit de posibilitățile oferite de extensii, unde și cum să stochezi datele utilizatorului, de ce trebuie folosit localStorage. Am folosit această caracteristică în locul unei baze de date prin care puteam accesa aplicația. Numai utilizatorii care au deja un cont pot folosi această extensie.

# Abstract

The topic of my thesis was to create an extension, the essence of which is to search for a product number in the "CATALOG SEARCH" application, which you can select in the user's browser. than what you have chosen.

Of course, this can also be solved by the user memorizing the product code on one page, then entering the application, entering the number and getting the results. The problem with this is that it takes a lot more time than the plugin I created.

There are situations in life where it is important that we can solve these things not only accurately, but as quickly as possible, so the time factor also matters. In my thesis, this goal was set for me to make the life of the application users easier and to achieve what they are interested in as quickly as possible. Among the aims of the thesis, the most important advantages of the extension should be mentioned, in addition, I also covered the technologies used, I thoroughly analyzed everything, why it is worth using them. To achieve my goal, I used the internet to gain interesting data.

In this thesis, I am looking for the answer to the question of how to take advantage of the possibilities offered by extensions, where and how to store user data, why use localStorage. I used this feature instead of a database through which I could access the application. Only users who already have an account can use this extension.

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>10</b>
<b>2. Programok, technológiák bemutatása</b>	<b>13</b>
2.1. Visual Studio Code . . . . .	13
2.2. Manifest . . . . .	14
2.3. HTML . . . . .	15
2.4. CSS . . . . .	16
2.5. JavaScript . . . . .	16
<b>3. GYAKORLATI MEGVALÓSÍTÁS</b>	<b>18</b>
3.1. Program felépítése . . . . .	18
3.2. Felhasználói felületek elkészítése . . . . .	19
3.3. Felhasználók bejelentkezése . . . . .	21
3.4. BearerToken megírása . . . . .	22
3.5. Felhasználók adatainak mentése localStorageban . . . . .	23
3.6. CATALOG SEARCH oldal megnyitása termékkód által . . . . .	26
3.7. Felhasználók kijelentkezése . . . . .	27
<b>4. Követelmény specifikációk</b>	<b>29</b>
4.1. Felhasználói követelmények. Use case diagram . . . . .	29
4.2. Alkalmazás működése . . . . .	30
4.3. Login . . . . .	30
4.4. Logout . . . . .	31
4.5. LocalStorage . . . . .	31
4.6. Nem-funkcionális követelmények . . . . .	32
<b>5. Tervezés</b>	<b>33</b>
5.1. Applikáció általános architektúrája . . . . .	33
5.2. Clean architektúra . . . . .	34
<b>6. Összefoglaló</b>	<b>35</b>
<b>7. Továbbfejlesztési lehetőségek</b>	<b>36</b>
<b>Ábrák jegyzéke</b>	<b>39</b>
<b>Irodalomjegyzék</b>	<b>40</b>

# 1. fejezet

## Bevezető

Napjainkban rengeteg ember a számítógépezést böngészésre használja. Meg kell könnyítsük az emberek dolgát azzal, hogy minden böngészésre szánt dolgot a lehető legkönnyebb, leggyorsabb módon elérjenek.

A Google Chrome a Google által fejlesztett webböngésző. Jelenleg ez a leggyakrabban használt böngésző, piaci részesedése az összes internethasználó körében körülbelül 63, 8 százalék.[1] Miért használjuk pont a Google Chrome böngészőt? Az egyik legfontosabb ok, hogy a Chrome mellett döntsünk az a gyorsaság. A Chrome-ot úgy tervezték, hogy a leggyorsabb webböngésző legyen. Egy kattintással villámgyorsan betölti a weboldalakat, több lapot és alkalmazást. A Chrome egy gyorsabb és erősebb JavaScript motorral van felszerelve, aminek a neve V8. Ezen kívül a Chrome egy egyszerű böngésző. Amikor bezárjuk a Chrome-ot, emlékezni fog a megnyitott lapokra, így közvetlenül onnan folytathatja, ahonnan abbahagytuk. A Chrome beépített PDF nézegetővel is rendelkezik, a beépített rosszindulatú programok és adathalászat ellen biztonságban tart minket. A Chrome Webáruházban hozzáadhatunk különböző alkalmazásokat, bővítményeket, témákat. A témák lehetővé teszik, hogy színekkel és grafikával életre keltsük böngészőnket, a bővítmények pedig a legfrissebb termékenységet, játékokat, oktatást stb. kínálják. Akár saját témát vagy bővítményt létrehozhatunk.

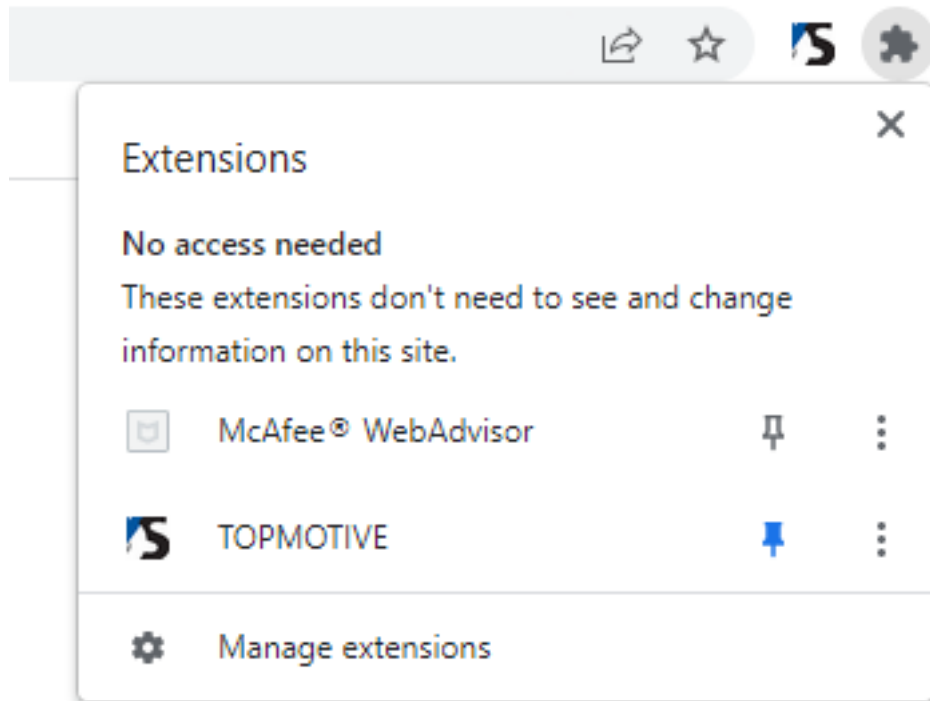
Nagyon sokan Chrome segítségével keresnek olyan oldalakra, ahol lehet rendelni termékeket. Mindennapjainkban rengeteg autót vehetünk észre az utakon, és sok ember az alkatrészek böngészésével tölt el sok időt. Ezen kívül a legfőbb probléma, hogy sok időt kell eltöltsön azért, hogy megkeressen egy másik ugyanolyan terméket, és hasonlítsa össze, hogy melyik az olcsóbb, melyik éri meg jobban. Céлом egy olyan bővítmény létrehozása volt ami által a felhasználó könnyen tud navigálni 2 autós alkatrész között. Ennek a bővítménynek a szerepe az, hogy amikor a felhasználó rámegy egy oldalra és kiválasztja azt az alkatrészt, amire szüksége van egy termékkód segítségével navigálja át egy másik oldalra, ahol ugyanazt vagy hasonló kategóriájú alkatrészt találhat.

### ***Mik a bővítmények? Miért használjunk bővítményeket?***

A bővítmények kis szoftverprogramok, amelyek testreszabják a böngészési élményt. Lehetővé teszik a felhasználók számára, hogy a Chrome funkcióit és viselkedését az egyéni igényekhez vagy preferenciákhoz szabják. Olyan webes technológiákra épülnek, mint a HTML, JavaScript és CSS. A bővítményeket a Chrome fejlesztői irányítópulton keresztül terjesztik, és közzéteszik a Chrome Webáruházban. A bővítményfájlok egyetlen .crx-csomagba vannak tömörítve, amelyet a felhasználó letölt és telepít. Elérésük nagyon

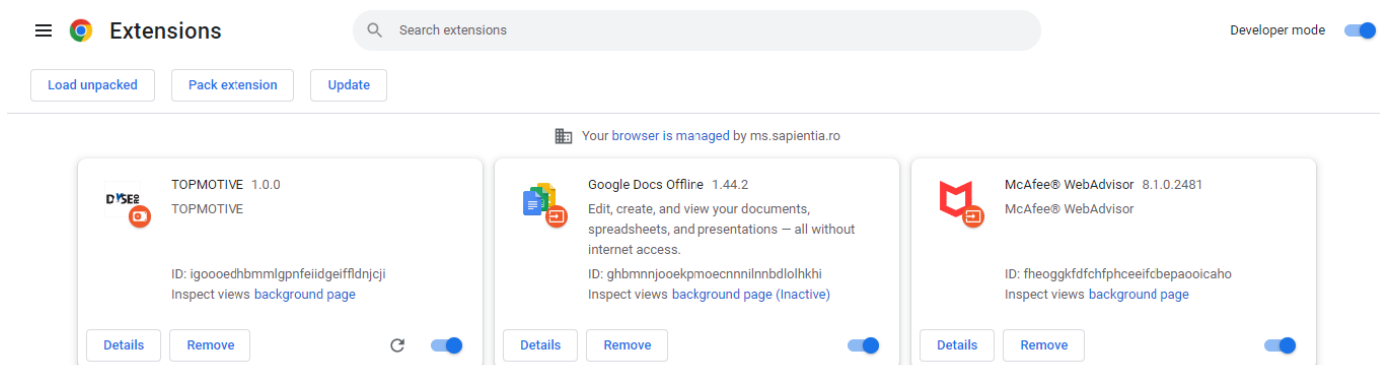


egyszerű, mert minden bővítmény esetén megjelenik egy kis ikon, amit ha a felhasználó megnyom meg is nyitja az oldalt. A bővítményeket ki lehet tűzni a Chrome fejlécében, ezt úgy tehetjük meg, hogy a kis puzzle ikonra rákattintunk, és a felugró ablakban láthatóak lesznek az elérhető bővítmények. Itt már csak rá kell kattintani a "pin" ikonra és meg is fog jelenni a fejlécen. [2]



1.1. ábra. Bővítmény megjelenítése

Ahhoz, hogy meg tudjuk nézni az elérhető bővítményeket, illetve ha újat akarunk létrehozni, a "Manage extensions" mezőre kell kattintsunk ahol a következő oldal fogad minket. Itt csak abban az esetben tudunk hozzáadni egy teljesen új bővítményt ha engedélyezzük a Developer mode-ot majd utána a Load unpacked gombra kattintunk ahol kiválaszthatjuk a mappát amit fel szeretnénk tölteni.



1.2. ábra. Bővítmény menedzselése

Sajnos nem bízhatunk meg teljesen a Google Chrome bővítményekben. Bár rengeteg hasznos extension van, a múltban már előfordultak olyan esetek, amikor rosszindulatú kó-

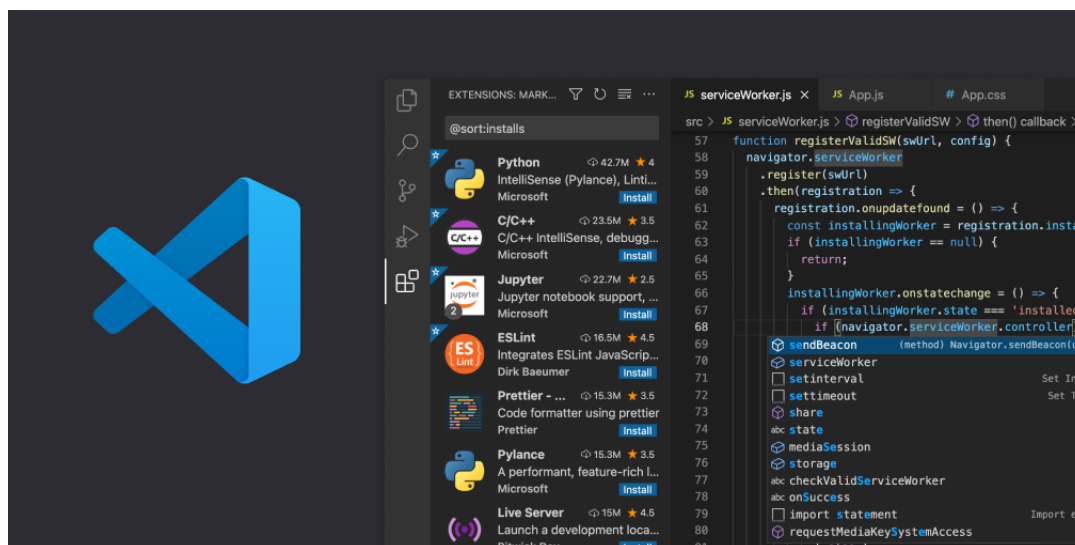
dok kerültek a felhasználók rendszerére hamis Google Chrome-bővítményeken keresztül. Néhány ilyen bővítmény akár 50 000 letöltés után is veszélyesnek minősül. Ennek érdekében a letöltés előtt meg kell tekinteni az adott Google Chrome-bővítményre vonatkozó véleményeket, értékeléseket és egyéb információkat. Ezenkívül a még nagyobb biztonság érdekében meg kell vizsgálni a bővítmény által kért engedélyeket, hogy ne adjunk szükségtelen hozzáférést az adatokhoz.

## 2. fejezet

# Programok, technológiák bemutatása

### 2.1. Visual Studio Code

Dolgozatomat a Visual Studio Code forráskódszerkesztő által alkottam meg. Ezt a környezetet a Windows fejlesztette, ami egyaránt működik Windowson, Linuxon és macOS-en.

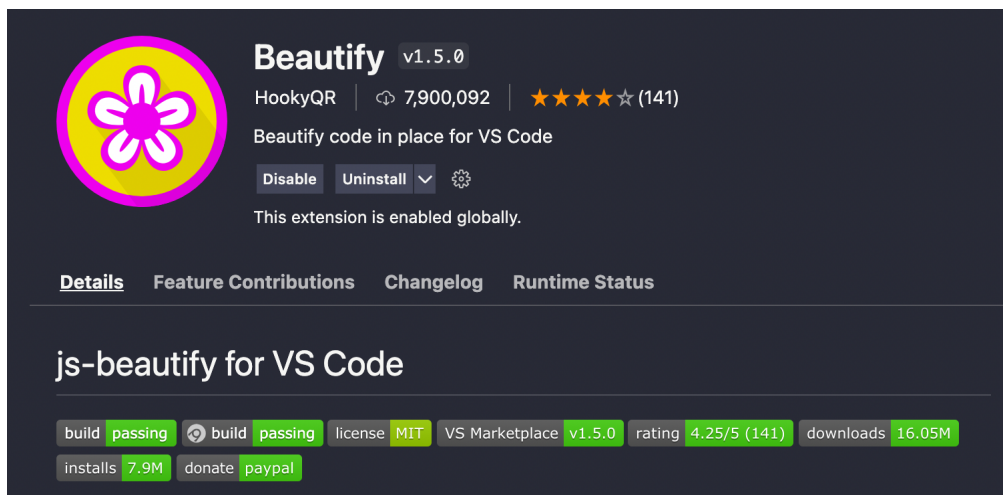


2.1. ábra. VS Code szerkesztő

#### *Miért használjuk a Visual Studio Code kódszerkesztőt?[3]*

A VS Code mellett, hogy ingyenes és villámgyors, tökéletes a mindennapi használatra. A több száz nyelv támogatásával a VS Code azonnali termelékenységet tesz lehetővé a szintaxis kiemelésével, a zárójelek illesztésével, az automatikus behúzással, a dobozkijelöléssel, a kódrészletekkel és még sok mással. Nagyon népszerű a weboldalak létrehozásában, a fejlesztők nagyon gyakran fordulnak a VS Codehoz a projektek létrehozása érdekében, mert nagyon jó eszközöket használ olyan webtechnológiákhoz, mint pl. React, HTML, CSS, Less és JSON. Az első lépésektől az új bővítmények telepítéséig a

VS Code-ban minden egyszerűnek és intuitívnek tűnik.[4] A tervezés nagyon szubjektív dolog, ezért a VSCode úgy döntött, hogy magáévá teszi ennek a tervezési megközelítésnek az alapelveit. A VS Code felhasználói felülete tiszta és jól megtervezett. Egy másik lényeges tulajdonsága az, hogy rengeteg bővítménnyel rendelkezik, amik a VS Code piacterén érhetőek el. Ezek a bővítmények megkönnyítik a felhasználó dolgát, könnyebb és még élvezhetőbb lesz számára a kódolás. Bővítmények nélkül a VS Code nem tartana itt, ahol most tart. A rengeteg bővítmény közül sok fontos bővítményt le is töltöttem. Ezek közül az egyiknek a neve "Beautify". Ez a bővítmény arra szolgál, hogy kitakarítsa a kódot. A fölösleges sorokat, szüneteket egy kattintás után ki is törli, így megkönnyítve a dolgunkat. A kódunk sokkal jobban átlátható lesz számunkra, és így sokkal könnyebben tudunk dolgozni majd. Egy tiszta kód rendkívül fontos egy programozó számára, hiszen meg kell tanuljunk tisztán dolgozni az összevisszaság elkerülése érdekében, gyorsan dolgozunk, így észre se vesszük, hogy mennyi fölösleges karaktert hagytunk a kódban, és, ha lehetőségünk van ilyen bővítmények használatára akkor mindenképp megéri használni, ráadásul nagyon kevés helyet foglal.



2.2. ábra. Beautify

## 2.2. Manifest

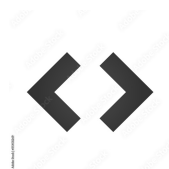
Dolgozatom ezen részében először vizsgáljuk meg azt a fájlt, ami nélkül nem működne a bővítmény. Minden bővítményhez egy manifest fájl szükséges JSON kiterjesztésben. A manifest egy olyan [JSON]-dokumentum, amely indítási paramétereket, és az alkalmazás alapértelmezett beállításait tartalmazza a webalkalmazás indításakor.[5] Itt tudjuk megformálni a bővítmény kinézetét(melyik legyen a kezdő ikon, melyik legyen az az ikon ami a fejlécben szerepel), ezen kívül meg kell adni a manifest verzióját, címét, azt a felugró ablakot, amit először látni fog a felhasználó a bővítmény megnyitása után. Ezen kívül meg kell adni a permissions-ban azokat az értéket amik az alkalmazásnak szükségesek.

```

{
  "name": "TOPMOTIVE",
  "description": "TOPMOTIVE",
  "version": "1.0.0",
  "manifest_version": 2,
  "icons": {"128": "images/dvse_icon.png"},
  "background": {
    "scripts": [".eventPage.js", ".popup-sign-in.js", ".popup_logout.html"],
    "persistent": true
  },
  "browser_action": {
    "default_icon": "images/icon_128.png",
    "default_popup": "popup.html"
  },
  "permissions": [
    "tabs",
    "storage",
    "notifications",
    "http://localhost:3000/",
    "*/**/*",
    "https://tm2.carparts-cat.com/*",
    "contextMenus"
  ]
}

```

2.3. ábra. Manifest



2.4. ábra. HTML logo

## 2.3. HTML

Mivel a bővítmények olyan webes technológiákra épülnek, mint a HTML, CSS, JavaScript, ezért dolgozatomban ezeket a technológiákat használtam.

A HTML egy olyan jelölőnyelv, amit weboldalak létrehozására használják, amelyek a világhálón jelennek meg. Minden oldal számos kapcsolatot tartalmaz más oldalakkal, amelyeket hiperhivatkozásoknak nevezünk. Minden megtekintett weboldal a HTML egy verziójával készült. A HTML kód biztosítja a szöveg és a képek megfelelő formázását az internetböngésző számára. HTML nélkül a böngésző nem tudná, hogyan jelenítsen meg szöveget elemként, illetve hogyan töltsön be képeket vagy egyéb elemeket.[6]

### ***Miért HTML? [7]***

Az ingyenesség mellett egy erős ok amiért sok fejlesztő választja, az a könnyen olvasható kód. Lehetővé teszi, hogy világos és leíró kódot lehessen írni, új és érdekes helyi tárolási funkcióval rendelkezik, a modern, népszerű böngészők mindegyike támogatja ezt a jelölőnyelvet. A HTML nem különbözteti meg a kis- és nagybetűket, és még elírási és szintaktikai hibák esetén is megjelenik, statikus weboldalakat hoz létre, amelyek nem frissülnek vagy változnak.

### ***Összehasonlítás más jelölőnyelvvvel***

A HTML és az XML jelölőnyelvek, hasonlóak a programozási nyelvekhez, de különböznek azoktól, mivel címkéket használnak a dokumentumok megjegyzésére. Hasonló szintaxist is használnak, például nyitó és záró címkéket.

A HTML kód kifejezetten a böngészőben való megjelenítéshez készült weboldalak tervezésére szolgál. Az XML csak adatátvitelre és tárolásra szolgál. Míg a HTML statikus, az XML dinamikus. A HTML-vel készült webhelyek általában nem változnak vagy frissülnek maguktól, míg az XML-t szinte mindig használják dinamikus alkalmazások előállítására. A HTML egy teljesen előre definiált jelölőnyelv, már definiált címkékkel és elemekkel. Nem hozhatunk létre saját HTML címkéket. Az XML inkább a jelölőnyelvek keretrendszere, a címkéket teljes egészében mi készítjük.

## 2.4. CSS



2.5. ábra. CSS logo

A HTML-re úgy gondolhatnánk, mint a weboldal csontjaira (struktúrájára), a CSS-re pedig úgy mint a bőrére (megjelenésére).

Egy weboldal futhat CSS nélkül, de biztosan nem szép. A CSS nagyon széppé teszi a weboldalak elejét, és nagyszerű felhasználói élményt biztosít. CSS nélkül a webhelyek kevésbé lennének kellemesek a szemnek, és sokkal nehezebb lenne navigálni. Az elrendezésen és a formátumon kívül a CSS felelős a betűszínért és egyebekért

### ***Miért CSS?***[8]

A CSS lehetővé teszi, hogy egyetlen CSS-szabályt használjon, és alkalmazza azt egy adott címke minden előfordulására egy HTML-dokumentumban. Jobb felhasználói élményt okoz, azaz a CSS nemcsak megkönnyíti a weboldalakot, hanem lehetővé teszi a felhasználóbarát formázást is. A CSS segítségével meghatározott formázási szabályokat és stílusokat alkalmazhat több oldalra egyetlen kódfüzérrel.

## 2.5. JavaScript

A JavaScript egy olyan szövegalapú programozási nyelv, amelyet kliens- és szervertoldalon is használnak, és amely lehetővé teszi a weboldalak interaktívvá tételét. A HTML és a CSS olyan nyelvek, amelyek szerkezetet és stílust adnak a weboldalaknak, a JavaScript interaktív elemeket és viselkedést ad a weboldalaknak.



2.6. ábra. JS logo

### ***Miért JavaScript?*** [9]

A JavaScript lehetővé teszi a felhasználók számára, hogy interakcióba lépjenek weboldallal. Rendkívül nagy szerepe van azoknál a weboldalaknál ahol be kell jelentkezni, vagy regisztrálni, hiszen JavaScript segítségével tölthetjük ki és küldhetjük el a szervernek az adatokat. Ezen kívül "életet" ad az egész weboldalnak, pl.a mezők kitöltése után változhat meg a gomb színe, hibás adatok esetén legyen piros a mezők színe és ne engedjen be az oldalra, kép nagyítása vagy kicsinyítése, stb.

A JavaScript a böngészők által használt nyelv. Könnyű elkezdni és megérteni. Más nyelvekkel ellentétben nem kell egy csomó programot telepíteni, mielőtt elkezdenénk a kódolást.

A JavaScript egy kliensoldali szkript is, amely felgyorsítja a program végrehajtását, mivel időt takarít meg a szerverhez való csatlakozáshoz. Mivel minden modern böngésző támogatja a JavaScriptet, szinte mindenhol látható. Minden híres cég használja a JavaScriptet eszközként, beleértve a Google, az Amazon, a PayPal. A JavaScriptet bármely weboldalba beágyazhatjuk, vagy egy másik programozási nyelv szkriptjébe. A JavaScript már képes front-end és back-end fejlesztésre is. A back-end fejlesztés a NodeJS-t használja, míg sok könyvtár segíti a front-end fejlesztést, például az AngularJS, ReactJS. A JavaScript a kód hosszának csökkentésével javítja a webes alkalmazások teljesítményét.[25]

## 3. fejezet

# GYAKORLATI MEGVALÓSÍTÁS

A munkám célja egy olyan bővítmény megalkotása volt, amely segíti a CATALOG SEARCH felhasználóit autóalkatrészek megrendelésére. Bármelyik ember rá tud keresni autóalkatrészekre böngésző által, viszont, amíg megkeresi ugyanazt a terméket egy másik oldalról az rengeteg időt venne el tőle. A CATALOG SEARCH felhasználóinak életét ebből a szempontból egy olyan bővítménnyel próbáltam megkönnyíteni, ahol a felhasználó ha rákeres egy termékre meg tudja nyitni azt a terméket vagy hasonló terméket a CATALOG SEARCH katalógusában. Minden terméknek van egy kódja és, amennyiben a felhasználó kiválasztja azt a termékkódot és rákattint jobb klikkel a bővítmény ikonja meg kell jelenjen, amit, ha megnyom átdobja a CATALOG SEARCH oldalra, hogy ellenőrizze le a hasonló termékeket.

Ezt a bővítményt csak a CATALOG SEARCH felhasználói használhatják. Minden felhasználónak van egy tokenje és csak abban az esetben szabad működjön a bővítmény, amennyiben a felhasználó be van jelentkezve. Tehát regisztráció nem volt szükséges a munkám során, egyedül az azonosításra volt szükségem. A bejelentkezés és a kijelentkezés oldalnak egy extensionnek megfelelő popup oldalt csináltam, ahol a felhasználó láthatja, hogy sikerült-e a bejelentkezés vagy nem.

### 3.1. Program felépítése

A program elkészítését öt nagyobb csoportra osztottam fel, majd logikai sorrendbe véve őket valósítottam meg:

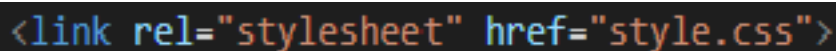
1. Felhasználói felületek elkészítése
2. Felhasználók bejelentkezése
3. Felhasználók adatainak mentése localStorageban
4. CATALOG SEARCH oldal megnyitása termékkód által
5. Felhasználók kijelentkezése



## 3.2. Felhasználói felületek elkészítése

A felhasználói felületnek rendkívül nagy befolyása van egy alkalmazás minőségében. A cél, hogy a rajta található dolgok funkciói mindenki számára könnyen érthető legyen, a felhasználó nem csak használni, de kihasználni is tudja a beépített funkciókat. Egy jó User Interface mindenki számára érthető kell legyen, ahol a felhasználó ki kell tudja használni a funkciókat, emellett egyszerű, letisztult. Egy alkalmazás elkészítésénél fontos dolog, hogy az alkalmazás legyen szép, az alkalmazás elemei legyenek rendezettek, szemnek tetszően elrendezve. Fontos, hogy kihasználjuk a vizuális effektusokat egy weboldalon belül, pl. használjuk ki a gombok tulajdonságait, változtassuk meg a gomb színét sikeres bejelentkezés esetén.

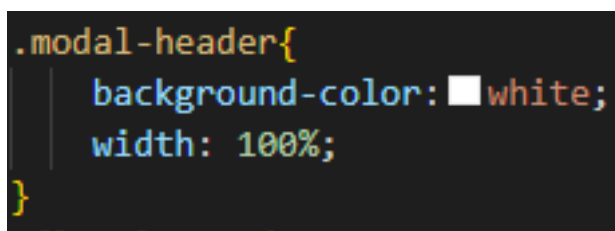
Weboldalam megalkotásához, illetve kifinomítására HTML-t és CSS-t használtam. A weboldalam igazából kettő darab HTML fájlból áll, amit akkor érünk el, ha rákattintunk a bővítmény ikonjára. Ezután fel fog ugrani egy popup ablak, ahol a felhasználó be kell jelentkezzen. A HTML fájlban megadtam azokat a mezőket, címeket amiket szükséges megjeleníteni bejelentkezéskor és az egész fájlt összekötöttem egy CSS fájlal, hogy ne legyen túl sok adat összetömörítve egy HTML fájlban. Ezt a következőképpen tettem meg a "<head>" részben:



```
<link rel="stylesheet" href="style.css">
```

3.1. ábra. HTML összekapcsolása CSS-el

Ahogy megtörtént az összekapcsolás, a HTML fájlomnak a body részét írtam meg. A bodyba került az egész weboldalam, ahol sok <div> címkét használtam. A <div> címkét a HTML elemek tárolójaként használtam, amelyet aztán CSS-vel stílusoztam. Ezt a címkét az osztály(class) attribútum használatával stílusoztam. Egy <div> címkén belül több hasonló címkét is használhatunk. A bodyn belül egy nagy <div> címkébe(modal-header) került a weboldalam szerkezete. Ezt a címkét a CSS-ben meghívtam, hogy állítsam be az oldalamnak a háttérszínét és szélességét.



```
.modal-header{  
  background-color: white;  
  width: 100%;  
}
```

3.2. ábra. Div címke stílusozása

Ezután adtam meg a mezők nevét, illetve a mezőket amiket ki kell töltsön a felhasználó. Ezeket mind 1-1 <div> classbe raktam, ahol megadtam a <label> címkehez tartozó mező nevét és az inputot, ahol be kell állítani a típusát és kell adjunk egy azonosítót(id) amit majd később el tudunk érni JavaScript által.

```
<div class="form-group">
  <label>Username</label>
  <input type="text" name="username" id="username" autofocus=""
</div>
```

### 3.3. ábra. Mező,mezőnév megadása

Ahhoz, hogy az oldalon szépen legyenek elrendezve ezek a mezők, megfelelő pozícióval, mérettel, színekkel a weboldalon szintén a CSS-hez fordultam, ahol meg kellett adjam a megfelelő pixeleket, betűtípusokat.

```
.form-group label{
  font-size: 20px;
  color: black;
  margin: 4px 18px;
  font-weight: bold;
}
.form-group input{
  padding: 5px 5px;
  border-radius: 20px;
  display: inline-block;
  border: 1px solid #ccc;
  box-sizing: border-box;
  margin: 6px 20px;
}
```

### 3.4. ábra. Mezők elrendezése CSS-vel

A HTML fájlban minden input elemnek van egy azonosítója(id). Ezeket a JavaScript által tudjuk meghívni illetve változóba elmenteni. Megoldottam azt, hogy amikor a felhasználó kitölti mindkét mezőt változzon meg a gomb színe. Kezdetben szürke és ahogy a második mezőbe elkezd gépelni a felhasználó az első után, a gomb színe legyen piros. Fontos, hogy a felhasználó lássa ezeket a vizuális részeket a weboldalnak, mert minél inkább kreatívabb annál inkább elnyeri majd a tetszését. A gomb megváltozásának a színét egy `addEventListener()` metódussal lehet elérni. Az `addEventListener()` egy beépített függvény a JavaScriptben, amely figyelembe veszi az eseményt, és egy második argumentumot hív meg, amikor a leírt esemény elindul. Egyetlen elemhez tetszőleges számú eseménykezelő adható a meglévő eseménykezelők felülírása nélkül. [3] Alapvető használat:

[4]

```
window.addEventListener('DOMContentLoaded', (event) => {  
    console.log('DOM fully loaded and parsed');  
});
```

### 3.5. ábra. AddEventListener használata

A DOMContentLoaded esemény akkor aktiválódik, amikor a kezdeti HTML-dokumentum teljesen betöltődött és elemzett, anélkül, hogy meg kellene várnia a stíluslapok, képek és alkeretek betöltésének befejezését.

Ezután egy olyan eseményt használtam ami nélkül nem tudtam volna ezt a gomb-színváltozást elérni, a neve onkeyup. A JavaScript onkeyup esemény egy olyan típusú esemény, amely akkor következik be, amikor az alkalmazást kezelő felhasználó elenged egy billentyűt a billentyűzeten. Ezzel a módszerrel kezelhetjük az eseményeket a származtatott osztályokban.

```
Object.onkeyup = function(){myFunction()};
```

### 3.6. ábra. Onkeyup használata

A fenti ábrán látható "myFunction()" függvényben kezeltem le azt, hogy hogyan változzon meg a gomb színe. Először is ID szerint lekértem a mezőket JavaScripten "document.getElementById" segítségével és eltároltam olyan változókba, amik globális hatókörűek, nem támogatják a blokk szintű hatókört. Majd if utasítás által megnéztem, hogy amennyiben ezek a lementett változók üresek, akkor a gomb színe legyen szürke, másképp változzon meg piros színűre.

## 3.3. Felhasználók bejelentkezése

Ezután egy olyan aszinkron eseményt hoztam létre, ahol azt kezeltem, hogy mikor legyen sikeres a bejelentkezés. Ehhez egy preventDefault() metódust használtam, ami törli az eseményt, ha az megszakítható, vagyis az eseményhez tartozó alapértelmezett művelet nem történik meg. Ez például akkor lehet hasznos ha a felhasználó a „Küldés” gombra kattint, mert megakadályozhatja, hogy űrlapot küldjön be.[5]

Aszinkron eseményt hoztam létre, amely lehetővé teszi, hogy a program elindítson egy hosszabb ideig futó feladatot, és továbbra is képes legyen reagálni más eseményekre, miközben a feladat fut, ahelyett, hogy meg kellene várnia, amíg a feladat befejeződik, ellenétben a szinkron eseménnyel, ahol az egyik esemény meg kell várja a másikat, hogy fejezze be a feladatát.

Ezután egy objektumban eltároltam a felhasználónév és jelszó mezőket, amin keresztül meg tudtam nézni, hogy ha az egyik mező nincs kitöltve akkor a "Sign In" gombot

```
document.body.onkeyup = function(){
    var text_username = document.getElementById("username");
    var text_password = document.getElementById("password");
    var text_button = document.getElementById("sbm");

    if(text_username.value == "" || text_password.value == ""){
        text_button.style.backgroundColor = "rgb(165, 163, 163)";
    }
    else{
        text_button.style.backgroundColor = "rgba(226, 19, 19, 0.801)";
    }
}
```

3.7. ábra. Gomb háttérszínének megváltoztatása

```
let addUser = async (ev) => {
    ev.preventDefault();
    let user = {
        uname: document.getElementById('username').value,
        pass: document.getElementById('password').value,
    }
}
```

3.8. ábra. Felhasználó hozzáadása aszinkron módon

megnyomva ne jelentkeztesse be a felhasználót és a mezők színe legyen piros, hogy lássa a felhasználó, hogy valami probléma van a kitöltéssel.

A Sign In gomb megnyomása után amikor sikeresen megkaptuk a bearer token-t és a bejelentkezés sikeresen megtörtént egy másik popup oldalra kerülünk, ahol a felhasználó ki tud jelentkezni. Ezt az oldalt egy függvényen belül hívtam meg(afterSuccessfulLoginActions()) "window.location.href" segítségével. Ez egy olyan tulajdonság, amely megmondja a böngésző aktuális URL-címét. A tulajdonság értékének módosítása átirányítja az oldalt. Amint megtörtént az átirányítás, létrehozom ugyanebben a függvényben a context menüt. A context menünek az a szerepe, hogy bejelentkezés után és a termékkód kijelölése után jelenjen meg az a kis ikon jobb klikk esetén ami átirányítja a CATALOG SEARCH weboldalra. Ezt a chrome.contextMenus.create() függvény segítségével hoztam létre, ahol nevet, azonosítót és kontextust adtam meg.

### 3.4. BearerToken megírása

Egy olyan függvényre volt szükségem, ami visszatérít egy bearer token-t, ami által majd a felhasználók be tudnak jelentkezni az alkalmazásba. Először is egy konstans változóba(ami által a változóra való hivatkozás nem módosítható) tároltam el a kezdetleges url-t. Ez az a web api amit felhívunk, ez kapja meg a felhasználónevet és jelszót, majd visszatéríti a kapott token-t. Annak érdekében, hogy kapjunk vissza egy token-t szükséges

```

function afterSuccessfullLoginActions() {

    window.location.href = "./popup_logout.html";

    let contextMenus;
    if(contextMenus!="spendMoney") {
        contextMenus = chrome.contextMenus.create({
            "title": "CATALOG SEARCH",
            "id": "spendMoney",
            "contexts": ["selection"]
        });
    }
}

```

3.9. ábra. afterSuccessfullLoginActions függvény

megadni a bearer token adatait. Ezt egy inputData nevű objektumban tároltam, ahol átadtam többek között a bearer token azonosítóját, a felhasználónevet, jelszót, beállítottam az angol nyelvet.

```

async function getBearerLoginToken(user_name, pass_word) {
    const requestURL = 'https://tm2.carparts-cat.com/data/TM.Next.Authority/api/shell/login/v1/GetAuthToken';
    var inputData = {
        "authId": "v2fa", //bearer token id-ja
        "traderId": 0,
        "customerNo": "",
        "username": user_name,
        "password": pass_word,
        "languageId": 4, //angol nyelv
        "clearExistingSessions": false,
        "loginInterface": ""
    };
}

```

3.10. ábra. getBearerLoginToken függvény

Ezután egy try-catch utasítást végeztem. A try utasítás lehetővé teszi, hogy meghatározzon egy kódblokkot, amelyet a végrehajtás során tesztelni kell a hibák szempontjából. A catch utasítás lehetővé teszi egy végrehajtandó kódblokk meghatározását, ha hiba történik a try blokkban.[22]

Szükségem volt arra, hogy JSON formátú kódban kapjam vissza a visszatérített token értékét, ezért a fetch módszert használtam. A JavaScriptben a fetch() módszer a szerver felé történő lekérésre és az információk weboldalakra való betöltésére szolgál, a kérelem bármely API-ra vonatkozhat. Itt visszatéríti a response objektumot, majd json formátumba alakítja át. Szintaxisa lejjebb látható, ahol a módszer paramétere(url) az a URL, amelyre a kérést el kell küldeni : [23]

### 3.5. Felhasználók adatainak mentése localStorageban

Amint megtörtént a bejelentkezés a felhasználó adatait localStorageban mentettem le. A localStorage egy olyan tulajdonság, amely lehetővé teszi a JavaScript alkalmazások számára, hogy kulcs-érték párokat mentsenek egy webböngészőben lejáratí dátum nélkül.

```

fetch(
  'https://carbonfootprint1.p.rapidapi.com/CarbonFootprintF
  {
    method: 'GET',
    headers: {
      'x-rapidapi-key': 'your_api_key'
    }
  }
).then(response => {
  console.log(response);
});

```

**3.11. ábra. Async fetch metódus**

Ez azt jelenti, hogy a böngészőben tárolt adatok a böngészőablak bezárása után is megmaradnak ellentétben a sessionStorage tulajdonsággal, ami csak akkor tart fenn tárolt adatokat amíg a böngésző nyitva van. Amikor bezárjuk a böngészőt a tárolt eljárások törlődni fognak.[6]

Egy users nevű tömbben tároltam a felhasználó adatait, úgy, hogy a felhasználó jelszavát base64-ben titkosítottam a kommunikációs problémák elkerülése miatt. A titkosításhoz a btoa() metódust használtam, ami egy Base64 kódolású ASCII karakterláncot hoz létre egy bináris karakterláncból, majd az atob() függvény segítségével újra dekódolhatjuk az adatokat. Szintaxisa: [7]

```

btoa(stringToEncode)

```

**3.12. ábra. Btoa metódus szintaxisa**

A users tömbbe beraktam a felhasználó felhasználónevét és titkosított jelszavát, push metódust használva, ami egy új elemet tesz a tömb végére, majd egy dolgunk maradt még, az, hogy a localStorageban megjelenítsük annak a felhasználónak az adatait aki bejelentkezett. Ezt a setItem() metódus segítségével lehet megtenni. A setItem() metódus beállítja a megadott Storage Object elem értékét. A setItem() metódus a Storage Object-hez tartozik, amely egy localStorage objektum. Szintaxisa: [8]

```

localStorage.setItem(keyname, value)

```

**3.13. ábra. setItem szintaxisa**

A titkosítást és a lementett adatokat a localStorageban a következő módon oldottam meg:

Észrevettem, hogy, ha bejelentkezés után üresbe kattintok, majd megnyitom megint a bővítményt visszadob a kezdőoldalra. Célom az volt, hogy ne jelentkezzen ki és a bővítmény újraindítása után ne dobjon vissza a login oldalra, hanem a logout-ra. Ehhez

```

if (bearerToken != null) {
    afterSuccessfulLoginActions();
    //base64
    var encodedString = btoa(user.pass); //accepts a "string"
    //console.log(encodedString);
    var users = [];
    users.push(user.username, encodedString);
    localStorage.setItem('UserList', JSON.stringify(users));
}
else {
    username_error.style.border = '2px solid red';
    password_error.style.border = '2px solid red';
}

```

3.14. ábra. Adatok elhelyezése localStorageban

szintén egy eseményt készítettem, ami a "Sign In" gomb megnyomása után betölti az adUser változót. Lekértem a localStorageból a felhasználók bejelentkezési adatait getItem() metódus segítségével, ahol meg kellett adjam a mező nevét(UserList), amit a setItem() metódusnál beállítottam. Ezeket az adatokat egy credentials nevű tömbbe mentettem le úgy hogy JSON.parse() metódust használtam. Ez a metódus egy JSON-karakterláncot elemez, létrehozva a karakterlánc által leírt JavaScript-értéket vagy objektumot.[9] Két változót adtam meg egyet a username-nek egyet a passwordnak. Kezdetben ezeket üres stringekkel láttam el, amiket majd felül fogok írni. Ez a felülírás csak akkor fog működni, ha a credentials értéket kap. Amennyiben megtörtént ez, a username felveszi a credentials tömb első(credentials[0]), illetve második értékét(credentials[1]). Ha a felhasználónév és jelszó értéke megváltozott akkor azért, hogy az oldalon maradjunk ismét meg kell hívjuk az afterSuccessfulLoginActions() függvényt. Ezután már bárhová kattinthatunk anélkül, hogy visszadobjon a kezdőoldalra.

```

document.getElementById('sbm').addEventListener('click', addUser);
var credentials = JSON.parse(localStorage.getItem('UserList'));
var username = "";
var password = "";
if(credentials){
    username = credentials[0];
    password = credentials[1];
}

if (username && password) {
    afterSuccessfulLoginActions();
}
});

```

3.15. ábra. Logout oldalon való maradás más helyre való klikkelés esetén

### 3.6. CATALOG SEARCH oldal megnyitása termékkód által

Bejelentkezés után történik a bővítményem legfontosabb része: egy random termékkód megnyitása a CATALOG SEARCH oldalon. Ehhez szintén az addListener eseményt használtam, amiben egy függvényt írtam meg, ami által történik a szám megnyitása egy másik URL-n. A függvényem egy if utasítással kezdődik, amiben azt tárgyaltam, hogy csak akkor történjen meg a termékkód megnyitása, ha a felhasználó a context menü "spendMoney" ikonjára kattint úgy, hogy előtte kijelölje azt a számot ami maga a termékkód.

```
if (clickData.menuItemId == "spendMoney" && clickData.selectionText) {
```

3.16. ábra. If utasítás a kijelölt szövegre

Ezután lekezeltem a szöveget, úgy, hogy csak az olyan szöveget fogadja el aminek a típusa természetes szám(Integer). Ezt a következő függvény segítségével oldottam meg: [10] Amennyiben a felhasználó nem természetes számot adott meg az else ágban alert-et használtam, ami egy felugró ablakot jelenít meg, a tartalom megváltozik és azonnal felhívja rá a felhasználó figyelmét.

```
function isInt(value) {  
    return !isNaN(value) &&  
        parseInt(Number(value)) == value &&  
        !isNaN(parseInt(value, 10));  
}
```

3.17. ábra. Természetes szám kezelése

Ha a termékkód sikeresen átmegy a természetes szám ellenőrzésen ismét egy tömbbe lekérem a felhasználó adatait JSON.parse metódus segítségével, a felhasználónév felveszi a tömb első elemét. Ezután a jelszó értékét dekódoltam atob függvény segítségével, ami a btoa fordítottja, base64-ből visszaállítottam az eredeti formájába majd elmentettem egy változóba.

```
let password = atob(credentials[1]);
```

3.18. ábra. Atob metódus

Ezután egy másik változóba lementettem és meghívtam az aszinkron getBearerLoginToken függvényt. Ezt a függvényt az await kulcsszóval tudtam elérni. Ez egy olyan operátor ami a várakozásra szolgál. Chrome.tabs API-t használtam a böngésző laprendszerével való interakcióhoz. Ezzel az API-val lapokat hozhatunk létre, módosíthatjuk és átrendezhetjük a böngészőben. [11] Itt megnyitottam a kezdő URL-t, ami tartalmazza a bearer token is, chrome.tabs.create segítségével. A bearer tokenre a link eléréséért van szükségem.



Az oldal létrehozása után el kell tárolni az aktív lapot. Ez általában a felhasználó aktuális lapjának tekinthető. Az alábbi példa bemutatja, hogy a háttérszkript hogyan tudja lekérni az aktív lapot az aktuálisan fókuszált ablakból. [11]

```
let bearerToken = await getBearerLoginToken(username, password);
chrome.tabs.create({ url: `https://tm2.carparts-cat.com/login?token=${bearerToken}&loginType=1&lang=4` });
var activeTabId;
chrome.tabs.query({ active: true, currentWindow: true }, function (tabs) {
    var currTab = tabs[0];

    if (currTab) {
        //eltaroljuk az aktív tabID-t
        activeTabId = currTab.id;
    }
});
```

3.19. ábra. Tabs.create, tabs.update használata

Azért, hogy még inkább látványos legyen az oldalak közötti navigálás azt próbáltam megoldani, hogy az aktív lap betöltése után egy kis idő elteltével töltsse be a keresett termékkód által kigenerált oldalt. Először is setTimeout metódust alkalmaztam. Ez a metódus a megadott idő után végrehajt egy kódblokkot. A metódus csak egyszer hajtja végre a kódot. A setTimeout() metódus egy intervallID-t ad vissza, ami egy pozitív egész szám.

A JavaScript setTimeout általánosan használt szintaxisa a következő: [12]

```
setTimeout(function, milliseconds);
```

3.20. ábra. SetTimeout szintaxisa

A termékkód által kigenerált oldalt a tabs.update függvénnyel valósítottam meg. Ennek a függvénynek a szerepe az, hogy navigáljon a lapon egy új URL-re, vagy módosítsa a lap egyéb tulajdonságait. A url-ben lementettem azt a linket amit a felhasználó meg kell kapjon a termékkóddal a végén ami a kijelölt szám lesz.

```
setTimeout(() => {
    chrome.tabs.update(activeTabId, {
        url: `https://tm2.carparts-cat.com/6yaBuIGw36oq1kmlrow4pk/parts/direct/list/direct?query=${clickData.selectionText}`,
    });
}, 7000)
```

3.21. ábra. Chrome.tabs.update

## 3.7. Felhasználók kijelentkezése

A logout oldalnak eléggé hasonló a szerkezete a login oldallal, ugyanolyan színeket is használtam. Ahhoz, hogy a felhasználó tudja, hogy sikeres volt a bejelentkezés egy üzenet

fogadja a nevével. Ezt úgy alkottam meg, hogy vizsgáltam, hogy abban az esetben, ha a localStorageben értékek vannak, akkor lekértem az adatokat getItem függvény segítségével a localStorageben, amit egy tömbben tároltam. Egy változóban lementettem a tömb első elemét, amit innerHTML segítségével kiírtam. Az innerHTML az elemnek olyan tulajdonsága, amely lehetővé teszi az elemen belüli HTML-jelölés lekérését vagy beállítását.

Azt oldottam meg, hogy a logout gomb megnyomása után a felhasználót dobja vissza a kezdőoldalra, illetve a localStorageből töröljem ki a felhasználó adatait. Ezt hasonlóan oldottam meg, mint az addUser-t bejelentkezéskor, tehát egy aszinkron eseményt használtam, ahol kitöröltem a localStorage elemeit. Ezt window.localStorage.clear metódussal oldottam meg, a clear() metódus eltávolítja a tartomány összes Storage Object elemét.

Kijelentkezés esetén a context menüt is töröltem azonosító szerint chrome.contextMenus.remove segítségével, majd a window.location.href parancs által visszavigazgálok a felhasználót a kezdőoldalra.

```
if(localStorage.getItem('UserList')!=null){
var credentials = JSON.parse(localStorage.getItem('UserList'));
let username = credentials[0];
document.getElementById("output").innerHTML = "Logged in as " + username;
}

let removeUser = async (ev)=>{
  ev.preventDefault();
  window.localStorage.clear();
  chrome.contextMenus.remove("spendMoney")
  window.location.href = 'popup.html';
}

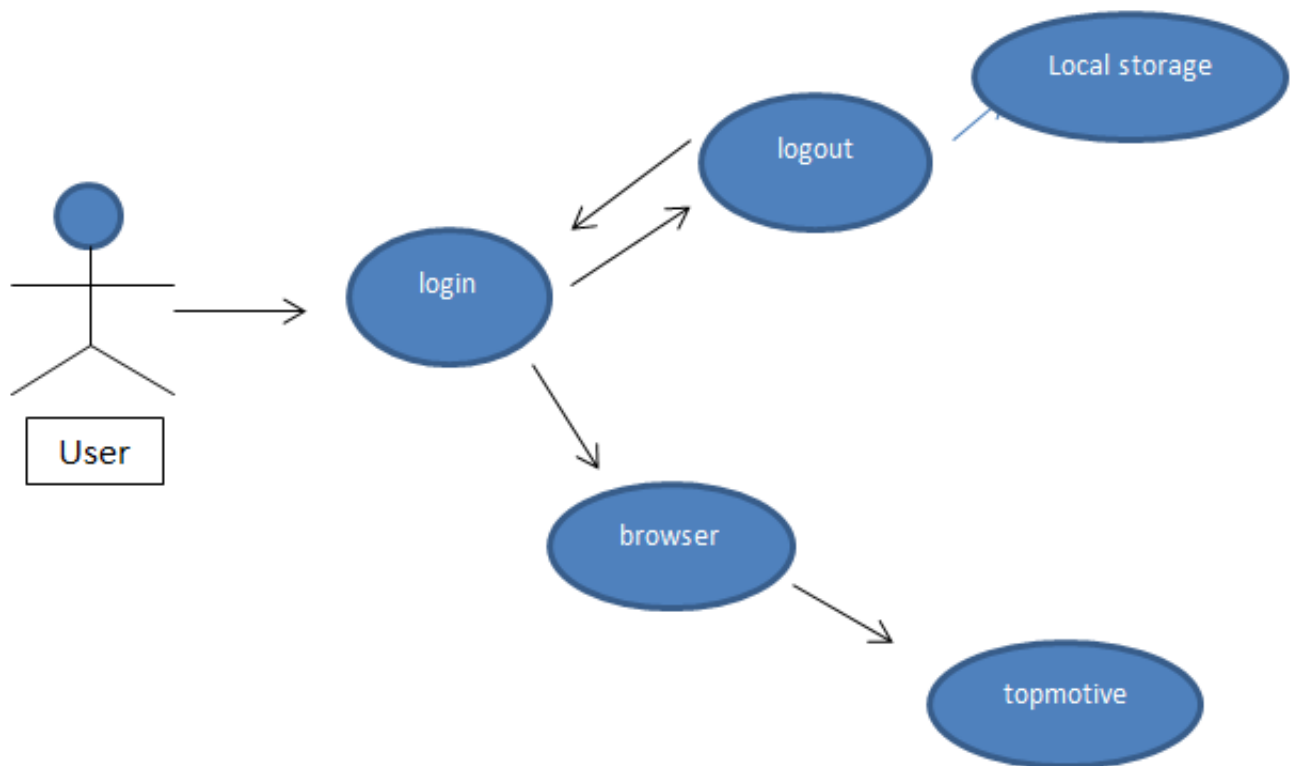
document.getElementById('logout').addEventListener('click',removeUser)
```

3.22. ábra. Logout JavaScriptben

## 4. fejezet

# Követelmény specifikációk

### 4.1. Felhasználói követelmények. Use case diagram



4.1. ábra. Use case diagram

**Login:** A login a kezdőoldala a bővítményemnek, ahol a felhasználó be kell jelentkezzen.

**Logout:** Login után a felhasználót bedobja a logout oldalra. Itt a felhasználó eldöntheti, hogy kijelentkezik és visszakerül a kezdőoldalra, vagy megnyitja inspecttel az applikációt, ahol megtekintheti a localStorage-t, ahol az adatait láthatja.

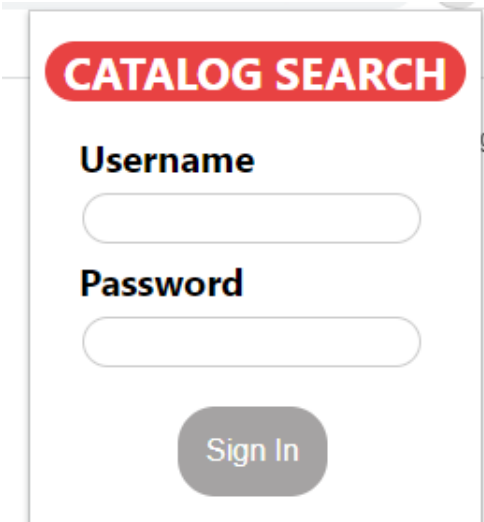
**Browser:** Amint megtörtént a bejelentkezés a felhasználó a Google Chrome browser segítségével kereshet autós alkatrész oldalak között, ahol kiválaszthatja azokat a termékeket amikre szüksége van.

**CATALOG SEARCH:** Amint a felhasználó kiválasztotta azt a terméket amit meg szeretne rendelni, minden terméknek van egy termékkódja, amit ha a user kijelöl egy context menü jelenik meg CATALOG SEARCH ikonnal, amit, ha megnyom bedobja a CATALOG SEARCH oldalra, ahol megkaphatja azokat a termékeket amit termékkód alapján kidob neki az oldal.

## 4.2. Alkalmazás működése

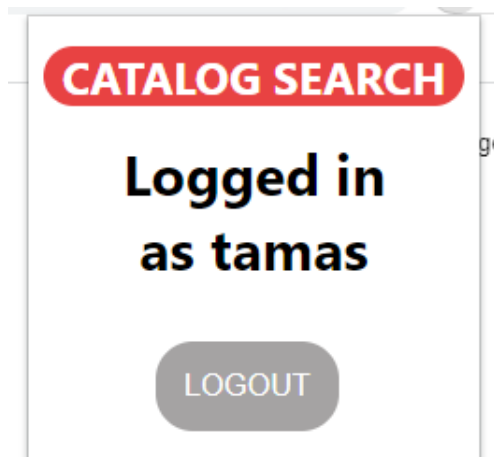
Az előző fejezetben a dolgozatom gyakorlati megoldásáról volt szó, ebben a fejezetben szemléltetem azokat amiket megalkottam.

## 4.3. Login

A login form titled "CATALOG SEARCH" in a red rounded rectangle. Below the title, there are two labels: "Username" and "Password", each followed by a white rounded rectangular input field. At the bottom of the form is a grey rounded button with the text "Sign In" in white.

4.2. ábra. Login oldal felhasználói felülete

A bővítmény megnyitása után egy popup oldal jelenik meg. Ez az oldal a kezdőoldal, itt kell bejelentkezzen a felhasználó. Csak azok tudnak bejelentkezni akiknek már fiókjuk van a CATALOG SEARCH oldalán, nem lehet itt regisztrálni. Amint a felhasználó kitölti mindkét mezőt még a Sign In gomb megnyomása előtt a gomb háttérszíne szürkéről átcserélődik pirossá. Amennyiben a kitöltés helyes a Sign In megnyomása után átnavigálja a felhasználót a logout oldalra.

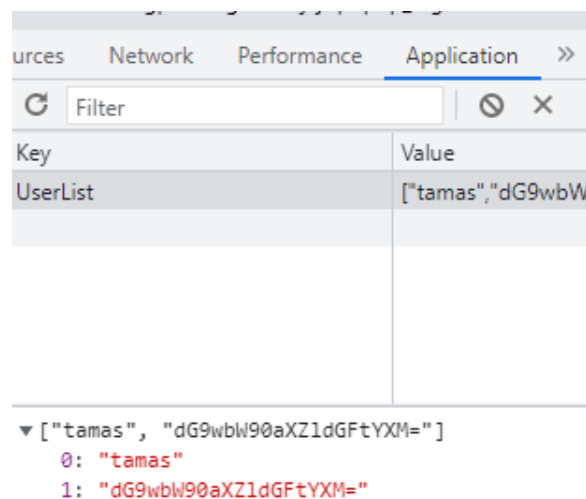


4.3. ábra. Logout oldal felhasználói felülete

## 4.4. Logout

Sikeres bejelentkezés esetén egy üzenet fogadja a felhasználót a saját felhasználói nevével. A felhasználó akkor jelentkezik ki, amikor szeretne a "Logout" gomb megnyomása által. Amielőtt kijelentkezne a felhasználó leellenőrizheti adatait a localStorageben. Ezt úgy lehet megtenni, hogy jobb klikket nyomunk az oldalnak, és rakattintunk az inspectre. Ezután a fejlécben ki kell választani az application mezőt, ami által megtekinthetők a felhasználó adatai.

## 4.5. LocalStorage



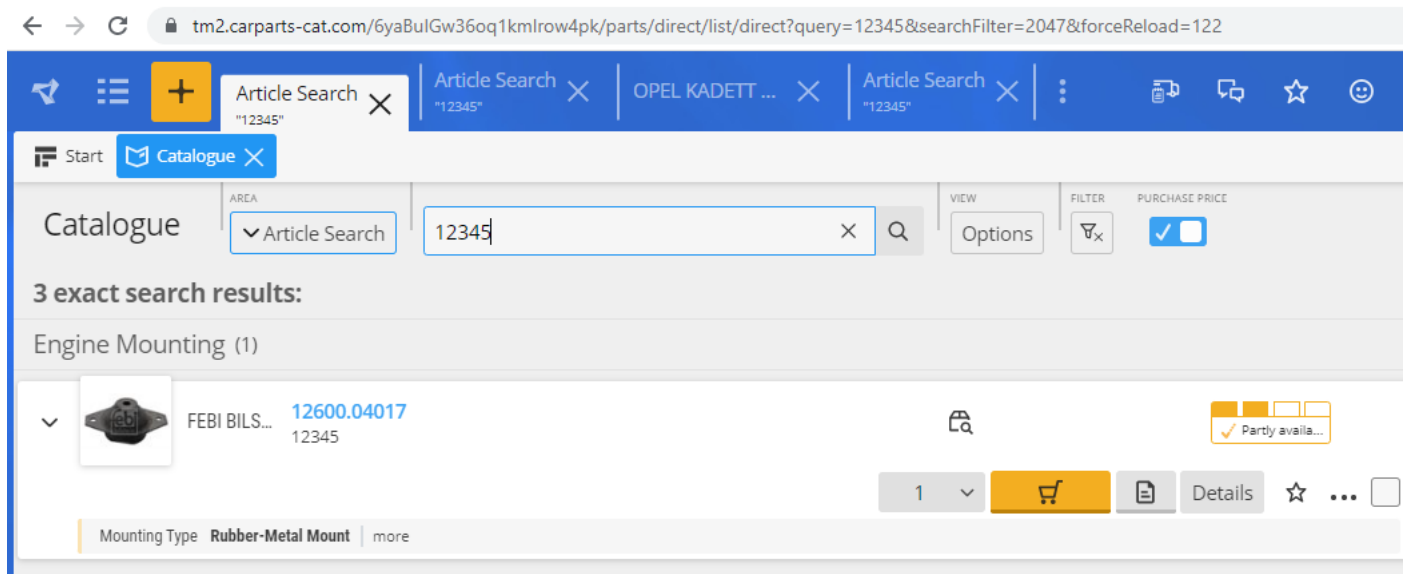
4.4. ábra. LocalStorage felhasználói felülete

Ezután, ha rákeresünk akármelyik autós weboldalra, minden terméknek van egy termékkódja, amit, ha a felhasználó kiválaszt és jobb klikket nyom akkor egy context menü jelenik meg "CATALOG SEARCH" felirattal.

CATALOG SEARCH

#### 4.5. ábra. Context menü

Az ikon megnyomása után a felhasználót a CATALOG SEARCH kezdőoldal fogadja, ami néhány másodperc alatt átvált egy olyan oldalra, ahol a kiválasztott termékkód termékei jelennek meg a CATALOG SEARCH oldalán.



#### 4.6. ábra. CATALOG SEARCH katalógusa

### 4.6. Nem-funkcionális követelmények

**Termékkel kapcsolatos követelmények:**

Alkalmazott szoftver	Leírás
Operációs rendszer	A Windows operációs rendszert választottuk, mert felhasználóbarát
Web Browser	Annak érdekében, hogy teszteljem a projekt működését Google Chrome böngészőt használtam

#### 4.7. ábra. Termékkel kapcsolatos követelmények

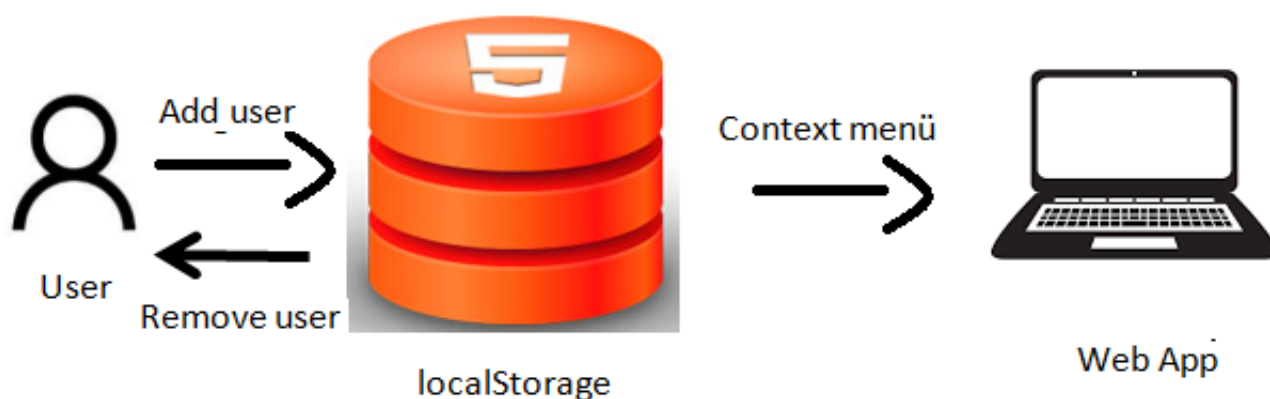
#### Fejlesztői követelmények

- HTML, CSS, JavaScript
- Fejlesztői környezet (Visual Studio Code)

## 5. fejezet

### Tervezés

#### 5.1. Applikáció általános architektúrája



5.1. ábra. Architektúra

Applikációm követi a szerepkörök szétválasztásának elvét, elkezdve a macro- egészen a micro architektúráig. A macro architektúra 3 nagy részből épül fel, egy Front-endből, egy Back-endből, és egy harmadik komponensből, ami a localStorage.

A Front end egy olyan fejlesztés, amely az alkalmazás vizuális elemeire összpontosít, amelyekkel a felhasználó interakcióba lép. Erre használtam a HTML, CSS, JavaScript technológiákat. A Back-end fejlesztés a webhely azon oldalára összpontosít, amelyet a felhasználók nem láthatnak. Ez teszi interaktívvá a webhelyet. Ehhez szintén JavaScriptet használtam ami használható úgy a front-endnél, mint a back-endnél. A harmadik komponens a localStorage, ami a weboldal tárhelye.

Webalkalmazás előnyei: [20]

- Könnyű adatfelhasználás
- Gyorsaság
- Egyszerű használat

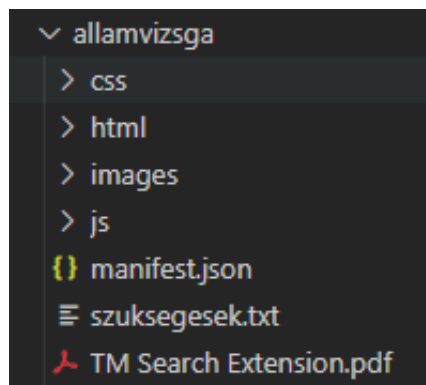
- Platformok közötti kompatibilitás
- Biztonságos élő adatok
- Kezelhetőség

LocalStorage előnyei: [21]

- A localStorage által gyűjtött adatokat a böngésző tárolja
- Az adatoknak nincs lejárat dátumuk
- Egyetlen kódsorral törölhető az összes localStorage elem
- Az adatok sokáig megmaradnak, ha a böngészőablak be van zárva

## 5.2. Clean architektúra

Dolgozatomat a clean architektúra segítségével valósítottam meg. A clean architektúra egy szoftvertervezési filozófia, amely a tervezés elemeit gyűrűszintekre választja szét[24]. A kódot több részre bontjuk, mindegyiknek lesz feladata. Ezáltal a szoftver kezelhetőbbé válik, jobban átlátható és mivel több komponensre vannak bontva a feladatok ezáltal a kezelhetőség is könnyebb lesz. Ezen kívül egy másik fontos előny, az, hogy az elemek könnyen cserélhetőek egymás közt.



5.2. ábra. Modulok



## 6. fejezet

# Összefoglaló

Projektemben mindent, amit elterveztem véghez is vittem. Sikerült létrehoznom egy olyan bővítményt, ami segíti az embereket az alkatrészek megvásárlására. Ehhez szükséges a bejelentkezés minden felhasználó részéről. Egy felhasználóbarát bővítményt hoztam létre, amit független az életkortól, minden személy kezelni tud.

Dolgozatomban a webfejlesztés 3 legalapvetőbb technológiáját használtam: HTML, CSS, JavaScript. A HTML és CSS segítségével egy szemnek tetsző alkalmazást alkottam meg, az elemek rendezett helyen vannak, a felhasználói felületem viszonylag kezdetleges, van még mit rajta csiszolni, de igyekeztem olyan színeket, formákat választani amik a felhasználók számára szimpatikus lehet.

JavaScript segítségével oldottam meg a dolgozatom lényegét. Ennek a programozási nyelvnek köszönhetően sikerült eltárolnom a felhasználói adatokat localStorageben, ami számomra teljesen új fogalom volt, de a projekt alatt sikerült megértenem a lényegét és fontosságát. JavaScript segítségével az adatok eltárolása mellett sikerült titkosítanom a felhasználók jelszavát beépített függvények segítségével, majd context menüt hoztam létre amit majd a felhasználó kell kiválasszon annak érdekében, hogy megnyissa az alkalmazást. A legnagyobb kihívás a tokenek használatával volt, mert minden felhasználónak volt egy felhasználóneve és jelszava, amit getBearer tokenek segítségével kezeltem le.

Ezután már csak a weboldalak betöltése volt hátra, az egyik ablakra egy másik felugró ablak pár másodperc híján megtörténe.

Köszönetet szeretnék nyilvánítani a DVSE RO cégnek, akik nélkül a projektem nem jött volna létre. Az ő kérésüknek megfelelően készítettem el a weboldalt, ők álmodták meg ezt a bővítményt, nekem csak az implementációban volt szerepem. Emellett hálás vagyok mentoromnak, Nadas Ákosnak, aki nagyon sok időt fektetett bele a munkámba, ha segítségre volt szükségem, ő volt az az ember akire mindig lehetett számítani, türelmes volt hozzám, és nagyon sok új dolgot tanultam meg tőle.

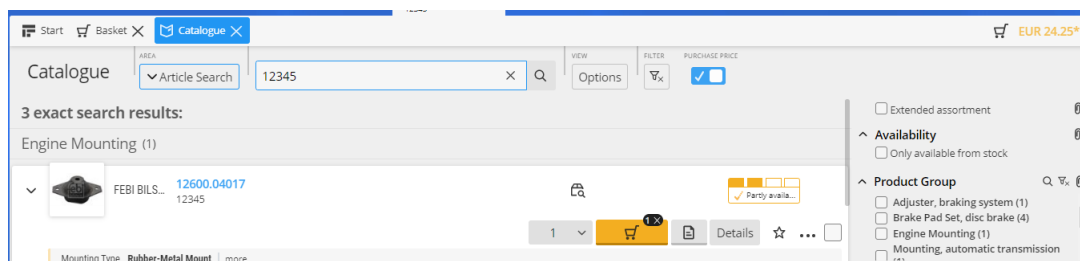
## 7. fejezet

# Továbbfejlesztési lehetőségek

Az alkalmazás végeredményben pontosnak, és megbízhatónak nevezhető, a felhasználó könnyen tud boldogulni ezzel a bővítménnyel. Ennek ellenére, a bővítmény egyelőre kezdetleges, ami rengeteg ötletnek szolgálhat biztos alapot majd a jövőben. Alkalmazásom jelen állapotában, új funkciók hozzáadása nélkül is hagy helyet a tovább fejlesztésnek. Annak érdekében, hogy még érdekesebb legyen a bővítmény több funkcióval kéne rendelkezzen, hogy a felhasználó mindent láthasson ami őt érdekli.

Először is a bővítmény rendelkezhetne kereső előzményekkel. Abban a pillanatban, ha a felhasználó rákattint egy termékkódra, az a szám mentődjön le a saját adatai közé. Ezt a számot linkként lehetne lementeni, hogy a felhasználó akármikor rá tudjon keresni még egyszer. Nagyon zsúfolt lenne, ha az összes előzmény látható lenne, ezért, hogy jobban és könnyebben átlátható legyen csak az utolsó 10 termékszámot mutassa. Ezt az előzményt a felhasználó adatai mellé lehetne lementeni. Az előbbi fejezetekben bemutattam, hogy néz ki a localStorage és kedvező lenne, ha az előzmények a 'UserList' mező alatt lenne lementve 'History' név alatt.

Egy másik fontos továbbfejlesztési lehetőség ami a munkám alatt eszembe jutott az a bevásárlókocsi megjelenítése lenne a bővítmény oldalán. A "Catalog Search" oldal rendelkezik ilyen eszközzel és minden terméket be lehet tenni a bevásárlókocsiba. Ezután, ha a felhasználó rákattint a kiválasztott termékek mezőre akkor láthatja, hogy eddig miket nézett ki magának és miket szeretne megvenni. A fent jobb oldalon látható kosarat és

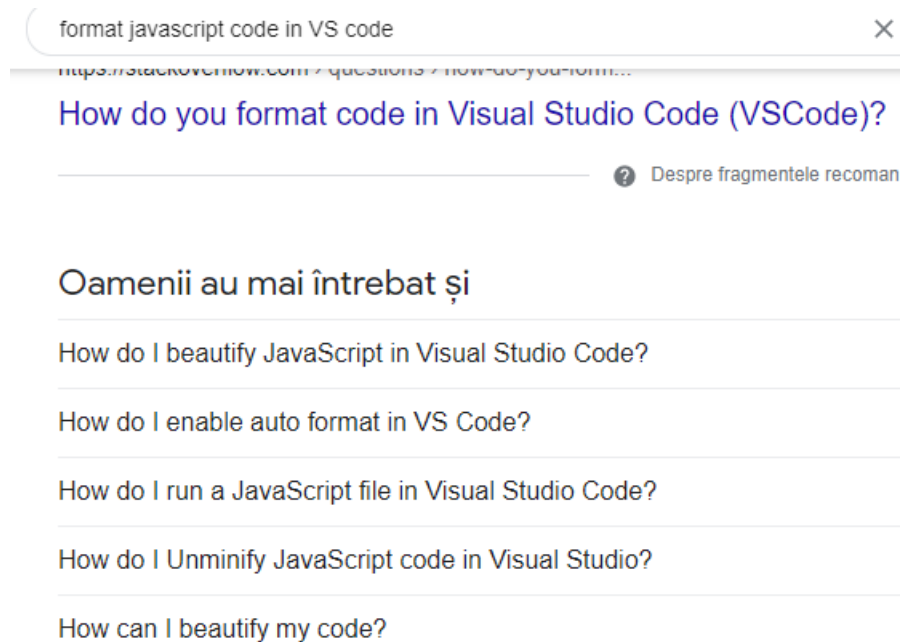


7.1. ábra. Shopping cart a weboldalon

az összárat lehetne megjeleníteni, amire rákattintva a felhasználó már a bővítményen keresztül láthatja azokat a termékeket, amiket a bevásárlókocsiba helyezett.

A 21.században élünk, és ebben az évszázadban nagyon gyakoriak a mesterséges intelligencia által fejlesztett technológiák. Az ember szinte észre se veszi de mindennap

találkozhat mesterséges intelligenciával már a böngészőben való keresés után is. Ahogy rákeresünk valamilyen témára, mindig kiad valami extra kérdéseket is amiket az emberek tettek fel hasonló kategóriájú témákban/kérdésekben, ezzel jelezve, hogy tudja mi után akarunk utánanézni. Legyen a következő példa:



## 7.2. ábra. Hasonló kérdések amiket emberek feltettek

A fenti példát alkalmazhatjuk a bővítményre is. Egy olyan látványosabb dolgot lehetne megalkotni, hogy a bővítmény tudjon "ajánlani" konkrét termékeket a kereső előzmények alapján. Például, ha egy olyan termékszámra keresett rá a felhasználó ami mondjuk olaj, akkor dobja fel a bővítmény a szűrőt is, mint ajánlat.

# Ábrák jegyzéke

1.1. Bővítmény megjelenítése . . . . .	11
1.2. Bővítmény menedzselése . . . . .	11
2.1. VS Code szerkesztő . . . . .	13
2.2. Beautify . . . . .	14
2.3. Manifest . . . . .	15
2.4. HTML logo . . . . .	15
2.5. CSS logo . . . . .	16
2.6. JS logo . . . . .	17
3.1. HTML összekapcsolása CSS-el . . . . .	19
3.2. Div címke stílusozása . . . . .	19
3.3. Mező,mezőnév megadása . . . . .	20
3.4. Mezők elrendezése CSS-vel . . . . .	20
3.5. AddEventListener használata . . . . .	21
3.6. Onkeyup használata . . . . .	21
3.7. Gomb háttérszínének megváltoztatása . . . . .	22
3.8. Felhasználó hozzáadása aszinkron módon . . . . .	22
3.9. afterSuccessfullLoginActions függvény . . . . .	23
3.10. getBearerLoginToken függvény . . . . .	23
3.11. Async fetch metódus . . . . .	24
3.12. Btoa metódus szintaxisa . . . . .	24
3.13. setItem szintaxisa . . . . .	24
3.14. Adatok elhelyezése localStorageban . . . . .	25
3.15. Logout oldalon való maradás más helyre való klikkelés esetén . . . . .	25
3.16. If utasítás a kijelölt szövegre . . . . .	26
3.17. Természetes szám kezelése . . . . .	26
3.18. Atob metódus . . . . .	26
3.19. Tabs.create,tabs.update használata . . . . .	27
3.20. SetTimeout szintaxisa . . . . .	27
3.21. Chrome.tabs.update . . . . .	27
3.22. Logout JavaScriptben . . . . .	28
4.1. Use case diagram . . . . .	29
4.2. Login oldal felhasználói felülete . . . . .	30
4.3. Logout oldal felhasználói felülete . . . . .	31
4.4. LocalStorage felhasználói felülete . . . . .	31
4.5. Context menü . . . . .	32

4.6.	CATALOG SEARCH katalógusa . . . . .	32
4.7.	Termékekkel kapcsolatos követelmények . . . . .	32
5.1.	Architektúra . . . . .	33
5.2.	Modulok . . . . .	34
7.1.	Shopping cart a weboldalon . . . . .	36
7.2.	Hasonló kérdések amiket emberek feltettek . . . . .	37

# Irodalomjegyzék

- [1] [https://ro.wikipedia.org/wiki/Google\\_Chrome](https://ro.wikipedia.org/wiki/Google_Chrome)
- [2] <https://developer.chrome.com/docs/extensions/mv2/overview/>
- [3] <https://code.visualstudio.com/docs/editor/whyvscode>
- [4] <https://blog.devgenius.io/the-reasons-why-you-must-use-visual-studio-code-b522f946a849>
- [5] <https://developer.mozilla.org/en-US/docs/Web/Manifest>
- [6] <https://www.computerhope.com/jargon/h/html.htm>
- [7] <https://kinsta.com/blog/xml-vs-html/>
- [8] <https://devmountain.com/blog/what-is-css-and-why-use-it/>
- [9] <https://www.hackreactor.com/blog/what-is-javascript-used-for>
- [10] <https://www.geeksforgeeks.org/javascript-addeventlistener-with-examples/>
- [11] [https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded\\_event](https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event)
- [12] [https://www.w3schools.com/jsref/event\\_preventdefault.asp](https://www.w3schools.com/jsref/event_preventdefault.asp)
- [13] <https://blog.logrocket.com/localstorage-javascript-complete-guide/>
- [14] <https://developer.mozilla.org/en-US/docs/Web/API/btoa>
- [15] [https://www.w3schools.com/jsref/met\\_storage\\_setitem.asp](https://www.w3schools.com/jsref/met_storage_setitem.asp)
- [16] [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/JSON/parse](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse)

- [17] <https://stackoverflow.com/questions/14636536/how-to-check-if-a-variable-is-an-integer-in-javascript>
- [18] <https://developer.chrome.com/docs/extensions/reference/tabs/>
- [19] <https://www.programiz.com/javascript/setTimeout>
- [20] <https://www.kcsitglobal.com/blogs/detail-blog/the-benefits-of-web-applications-in-todays-technological-era>
- [21] <https://www.atatus.com/blog/what-is-javascript-localstorage-a-complete-guide-for-beginners/>
- [22] [https://www.w3schools.com/js/js\\_errors.asp](https://www.w3schools.com/js/js_errors.asp)
- [23] <https://rapidapi.com/guides/fetch-api-async-await>
- [24] <https://www.techtarget.com/whatis/definition/clean-architecture>
- [25] <https://data-flair.training/blogs/advantages-disadvantages-javascript/>