

JEGYZŐKÖNYV

Web technológiák 1

Dodge Callenger InfoSite

Készítette: Glonczi Dániel

Neptunkód: YNBG2A

Dátum: 2025. december

Miskolc, 2025

Tartalomjegyzék

1. Feladat – Projektstruktúra kialakítása	6
1.1. Mappaszerkezet létrehozása	6
1.2. Alapfájlok előkészítése (HTML, CSS, JS).....	6
1.3. Képek (images) és videók (videos) mappák feltöltése	7
2. Feladat – Főoldal (index.html) kialakítása	7
2.1. Head szakasz összeállítása	8
2.2. Header és navigáció létrehozása	8
2.3. Hero szekció megvalósítása	8
2.4. Bemutató szekció és statisztikai elemek	9
2.5. Modellek szekció kialakítása	9
2.6. Lábléc (footer) elkészítése	10
3. Feladat – Globális stíluslap (style.css) elkészítése	10
3.1. Alap reset és betűtípus konfiguráció	10
3.2. Színpaletta és globális design-elemek definiálása	11
3.3. Főoldal szekcióinak stílusozása	11
3.4. Modelloldalak stílusai	11
3.5. Űrlapok, gombok, interakciók megvalósítása	12
3.6. Responsivitás kialakítása (media query-k).....	12
4. Feladat – Globális JavaScript logika (script.js).....	12
4.1. A dokumentum betöltést követő inicializáció	13
4.2. Smooth scroll funkció	13
4.3. Statisztika-animációk aktiválása	13
4.4. Színválasztó, dátumválasztó és slider kezelése	14
4.5. Üzemanyagköltség kalkulátor implementálása	14
4.6. Galéria modal ablak megvalósítása.....	14
4.7. DOM-szelektorok demonstrációs funkciói	14
5. Feladat – JSON adatbázis (cars.json) elkészítése.....	14
5.1. Modelladatok struktúrájának megtervezése	15
5.2. Specifikációk, méretek, árak rögzítése.....	16
5.3. Metaadatok felvétele	17
5.4. Összehasonlítási kulcsok rögzítése	18
5.5. Egységesség és validáció ellenőrzése.....	18
6. Feladat – Összehasonlító oldal (compare.html).....	19

6.1. Oldal HTML vázának megalkotása.....	20
6.2. Modellválasztó dropdownok elhelyezése.....	20
6.3. Összehasonlítás gomb implementálása	21
6.4. Eredménykártyák és táblázat struktúrája.....	22
6.5. JSON adatbetöltő szekció kialakítása	24
6.6. Lábléc hozzáadása.....	25
7. Feladat – Összehasonlítás logikája (compare.js)	25
7.1. Modellválasztás validációja	25
7.2. Táblázat kitöltése és animációk.....	26
7.3. Győztes meghatározása lóerő alapján	26
7.4. JSON betöltés AJAX segítségével	26
7.5. JSON megjelenítés formázása.....	27
7.6. Animációk és UX fejlesztések	28
8. Feladat – Kapcsolati oldal (contact.html)	28
8.1. Elrendezés és kétoszlopos grid kialakítása.....	29
8.2. Elérhetőségi információk megjelenítése	30
8.3. Komplett űrlap mezőkkel	30
8.4. Hibaüzenetek és CSS osztályok előkészítése.....	31
8.5. Sikerüzenet struktúrája	31
9. Feladat – Śrlapvalidáció (form-validation.js)	32
9.1. Submit esemény kezelése.....	32
9.2. Egyes mezők validációs logikája	33
9.3. Regex alapú email és telefon ellenőrzés	33
9.4. Hibakezelés, görgetés első hibára	34
9.5. Sikeres küldés animációk és reset	35
9.6. Valós idejű validáció input eseményekre	35
10. Feladat – Modeloldalak kialakítása.....	36
10.1. Demon 170 (demon.html)	36
10.1.1. Hero és bevezető	36
10.1.2. Specifikációs kártyák	37
10.1.3. Kiemelt tulajdonságok	38
10.1.4. Színválasztó modul	38
10.1.5. Videószekció	38
10.2. Hellcat Redeye (hellcat.html).....	39

10.2.1. Hero és bevezető	39
10.2.2. Részletes specifikációk	39
10.2.4. Dátumválasztó modul	39
10.2.5. Videószekció.....	40
10.3. Jailbreak (jailbreak.html)	40
10.3.1. Hero és bevezető	40
10.3.2. Testreszabási checkboxok.....	41
10.3.3. Slider modul.....	41
10.3.4. Videószekció.....	42
10.4. SRT 392 (srt.html).....	42
10.4.1. Hero és tartalmi blokk	42
10.4.2. Specifikációs kártyák	42
10.4.3. Datalist alapú kiegészítő-kereső	42
10.4.4. Üzemanyagkalkulátor modul	43
10.4.5. Videószekció.....	43
11. Feladat – Galéria (gallery.html).....	43
11.1. Képgaléria grid kialakítása	44
11.2. Overlay és hover effektek	45
11.4. Statisztikai számlálók megjelenítése	47
11.5. Lábléc elhelyezése.....	48
12. Feladat – Galéria JavaScript logika (gallery.js)	48
12.1. Számlálók scroll alapú aktiválása	48
12.2. Képek belépő animációi	50
12.3. Videók animációi és autoplay hover	50
12.4. Lazy loading megvalósítása	51
12.5. DOM manipulációs gyakorlatok	52
12.6. Kattintásszámláló implementálása	53
12.7. Pulzáló számanimációk	54
13. Feladat – Videókezelő logika (video-controls.js)	55
13.1. Videóelem kiválasztása	55
13.2. Lejátszás kezelés	56
13.3. Szünet funkció.....	56
13.4. Visszatekerés effektus	56
13.5. Némítás állapotkövetéssel	57

13.6. Videó események figyelése	57
14. Feladat – Projekt összekapcsolása és működés tesztelése.....	58
14.1. Fájlok közötti kapcsolatok ellenőrzése	58
14.2. JSON betöltési folyamat tesztelése	58
14.3. Responsivitás széleskörű vizsgálata.....	58
14.4. Interaktív elemek tesztje (modal, slider, date)	59
14.5. Videók működésének ellenőrzése	60
15. Feladat – Beandató dokumentáció előkészítése.....	60
15.1. Fájlszerkezet fa-diagramja	60
15.2. Fájlok részletes bemutatása	61
15.3. Tesztelési lista összeállítása	62
15.4. Fejlesztési javaslatok összegyűjtése.....	63
15.5. Képernyőképek és mellékletek.....	64

1. Feladat – Projektstruktúra kialakítása

A projekt elkészítésének első szakasza a teljes könyvtárszerkezet megtervezése és létrehozása. A cél az volt, hogy a weboldal minden komponense átláthatóan, logikusan és könnyen bővíthetően helyezkedjen el.

1.1. Mappaszerkezet létrehozása

A projekt három fő mappára tagolódik: public, images és videos.

- A public mappa tartalmazza a teljes weboldal minden HTML fájlját, a globális és oldalhoz kötődő JavaScript fájlokat és a style.css állományt.
- Az images mappa a weboldal összes statikus képét tartalmazza, amelyek oldalanként csoportosítva kerültek be.
- A videos mappa az egyes modelloldalakhoz kapcsolódó videókat tartalmazza.

A fájlnevek minden esetben tükrözik a tartalmat, például: card_demon_170.jpg, jailbreak_head_bg.jpg, 392_video.mp4. Ez a struktúra biztosítja, hogy a forrásfajlok egyértelműen beazonosíthatók legyenek és minimális legyen a konfliktus lehetősége.

1.2. Alapfájlok előkészítése (HTML, CSS, JS)

A public mappán belül létrejöttek az alábbi fő komponensek:

- index.html: a főoldal, amely a teljes projekt vizuális központja.
- compare.html: a JSON-ból dolgozó autóösszehasonlító oldal.
- gallery.html: kép- és videógaléria statisztikai elemekkel.
- contact.html: kapcsolatfelvételi oldal, űrlapmezőkkel és validációval.
- demon.html, hellcat.html, jailbreak.html, srt.html: egyedi modelloldalak.
- style.css: a projekt egységes megjelenését biztosító stíluslap.
- script.js: globális JavaScript logika, animációk és interakciók.
- compare.js: az összehasonlító logika teljes működése.

- gallery.js: a galéria animációi, számlálói és interaktív elemei.
- form-validation.js: az űrlapmezők validációját végző fájl.
- video-controls.js: a videókezelést biztosító fájl.
- cars.json: a projekt adatbázisa, amely négy autómodell részletes adatait tartalmazza.

1.3. Képek (images) és videók (videos) mappák feltöltése

A képek és videók fontos szerepet kapnak a projekt vizuális felépítésében. A feltöltés során külön figyelmet fordítottam arra, hogy minden kép megfelelő felbontással rendelkezzen és összhangban legyen az adott oldal dizájnjal. A képek között szerepelnek háttérképek (például: demon_170_head_bg.jpg), modellkártyákhoz tartozó képek és galériaelemek. A videos mappában pedig minden modellhez készült egy-egy rövid bemutató videó, amelyek az oldalba ágyazva automatikusan vagy felhasználói interakcióra indulnak el. A képek és videók elnevezése konzisztens, így a projekt teljes élettartama alatt könnyen kezelhetők maradnak.

2. Feladat – Főoldal (index.html) kialakítása

A főoldal a projekt központi eleme, amely bemutatja a Dodge Challenger modelleket, valamint összefogja az egész weboldal vizuális arculatát és navigációját. A felépítés során egy modern, letisztult szerkezet kialakítása volt a cél, miközben több animált és interaktív elemet is beépítettem.



1. ábra Főoldal

2.1. Head szakasz összeállítása

A head rész tartalmazza a karakterkódolást, viewport beállítást, a weboldal címét, valamint a style.css és script.js fájlok hivatkozásait. A head szakasz biztosítja a globális stílusok és JavaScript funkciók működését. A strukturált head felépítéssel alapot teremtettem a reszponzív és egységes megjelenéshez.

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Dodge Challenger - Főoldal</title>
    <link rel="stylesheet" href="style.css" />
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  </head>
```

2.2. Header és navigáció létrehozása

A header egy fixen pozicionált navigációs sáv, amely minden oldalon azonos formában jelenik meg. A navigáció logikusan vezet végig a főoldal, galéria, összehasonlító oldal, kapcsolat oldal és modelloldalak között. A navigáció HTML struktúrája átlátható listákból épül fel, a CSS pedig biztosítja a letisztult, modern megjelenést. A felhasználói élmény javítása érdekében a header a görgetés során is látható marad.



2. ábra Header szekció

2.3. Hero szekció megvalósítása

A hero szekció a főoldal nyitó vizuális blokkja, amely nagy felbontású háttérképpel és egy figyelemfelkeltő címfelirattal jelenik meg. A hero szekció célja a projekt tematikájának azonnali kommunikálása – a Dodge Challenger izomautók világának bemutatása. A CSS kiemelt szerepet kap a háttérkép megjelenítésében és a tipográfiai elemek stílusában.

2.4. Bemutató szekció és statisztikai elemek

Ezt a szekciót azért hoztam létre, hogy a látogató azonnal kapjon vizuális és információs tartalmat az autókról. Itt helyezkednek el a statisztikai elemek, amelyek animációval jelennek meg. A statisztikák 0-ról induló számlálóeffektussal kelnek életre, amelyet a script.js irányít. Ez a dinamikus vizuális megoldás erősíti a weboldal modern, interaktív jellegét.

Dodge Challenger

A **Dodge Challenger** nem csupán egy autó - ez egy életérzés, egy legenda, amely ötvözi a klasszikus amerikai muscle car dizájnt a modern teljesítménnyel és technológiával.

Az 1970-es évek ikonikus modelljének modern újragondolása, amely megtartotta eredeti jellegét, miközben a legmodernebb technológiával és félelmetes teljesítménnyel rendelkezik.



3. ábra Bemutató szekció

2.5. Modellek szekció kialakítása

A modellek szekciójában négy autómodell jelenik meg kártyák formájában, képpel, rövid leírással és egy gombbal, amely az adott modelloldalra vezet. A kártyák egységes dizájn alapján készültek, és mindegyik egy saját képet használ az images mappából. A kártyák reszponzívak, rugalmas rácsszerkezetben jelennek meg, amelyek mobileszközön és nagy képernyőkön is harmonikusan törnek.

Modellválaszték
Válassz ki a neked tetsző Challenger modellt

Demon 170 A valaha készült legérősebb sorozatgyártási V8-as autó. 1025 lóerővel és 2,3 mp-s 0-100-zal ez a démon minden rekordot meghódít.	Hellcat Redeye 807 lörő pusztrázó arénáját a Redeye a Hellcat család csúcsa. Brutális gyorsulás és lenyűgöző teljesítmény várva és pályahasználatra.	Jailbreak Egyedi festenzabási lehetőségekkel, ahol te szabad meg a szabályokat. Kombinált színökkel, feliratok és betűs kitalálkoztatás egyszerűen.	SRT 392 A tökéletes egyszerű teljesítmény és minden napig használhatóság között. 485 lörővel ez az ideális választás az igazi muscle car élményhez.
Lörő: 0-100 km/h: Motor:	Lörő: 0-100 km/h: Motor:	Lörő: Testreszabály: Motor:	Lörő: 0-100 km/h: Motor:
1023 LE 1,9 mp 6,2L V8 Supercharged	807 LE 3,4 mp 6,2L V8 Supercharged	717-807 LE Vegyétek 6,2L V8 Supercharged	485 LE 4,2 mp 6,4L V8
Részletek	Részletek	Részletek	Részletek

4. ábra Modell választék

2.6. Lábléc (footer) elkészítése

A lábléc minden oldalon egységes, és a projekt lezáró vizuális és információs eleme. Tartalmazza a szerzői megjelölést, az évszámot, valamint további információkat a weboldal készítéséről. A footer letisztult szerkezete és a visszafogott színvilág egyensúlyt teremt a vizuális kompozícióban.



5. ábra Footer szekció

3. Feladat – Globális stíluslap (style.css) elkészítése

A style.css a teljes projekt vizuális megjelenésének alapja. A stíluslap célja az egységes, modern, reszponzív és esztétikailag koherens felület biztosítása. A CSS logikusan tagolt szekciókból áll, könnyen bővíthető és átlátható felépítéssel.

3.1. Alap reset és betűtípus konfiguráció

A fejlesztés elején létrehoztam egy alap reset szakaszt, amely a böngészők eltérő alapértelmezett beállításait egységesíti. Ennek részeként nulláztam a margin, padding és box-sizing tulajdonságokat, hogy a további elemek stílusa pontosan szabályozható legyen. Ez elengedhetetlen egy konzisztens dizájn kialakításához. A projektben modern, jól olvasható betűtípus került beállításra. A body elemhez alap betűméret, háttérszín és törzsszín társult, amelyekből a teljes vizuális hierarchia kiindul.

```
/* Global Styles */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

```
body {  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
    line-height: 1.6;  
    color: #333;  
    background-color: #f4f4f4;  
}
```

3.2. Színpaletta és globális design-elemek definiálása

Meghatároztam a weboldal fő színpalettáját. A domináns színek a feketére, fehérre és a piros különböző árnyalataira épülnek - ez illeszkedik az amerikai izomautók karakteréhez. A változók nélküli, közvetlen értékadás helyett logikus struktúrában adtam meg a kiemelésre használt színeket, háttérszíneket és tipográfiai elemek színeit. A globális elemek, mint például gombok, linkek, kártyák és dobozok is egységes stílust kaptak, hogy a weboldal minden része azonos vizuális identitást sugározzon.

3.3. Főoldal szekcióinak stílusozása

A főoldal több különálló vizuális blokkból épül fel. A hero szekció nagy háttérképet, középre igazított szöveget és erőteljes tipografiát kapott.

- A bemutató rész harmonikusan illeszkedik a hero szekció után, kontrasztos háttérrel és animált számokkal.
- A modellkártyák rácsszerkezetben jelennek meg, egységes árnyékkal, kerekítéssel és a hover állapotban animációval. A CSS biztosítja, hogy a teljes oldal mobilon, tableten és széles kijelzőkön is jól szervezett, szép elrendezésben látható.

3.4. Modelboldalak stílusai

Minden modelloldal egyedi hangulatot kapott, mégis a központi arculat része marad. A fejlécekhez teljes szélességű háttérképek tartoznak, amelyek a modellekhez illő dinamikát sugallnak. A specifikációs blokkok kártyaszerű megjelenést kaptak, jól olvasható tipográfiával és tagolással. Interaktív elemek – például a Jailbreak oldalon található csúszka vagy a Hellcat oldalon elhelyezett dátumválasztó – minden külön CSS támogatást kaptak. A színválasztók, testreszabási elemek és videóblokkok harmonikusan illeszkednek a modelloldalak tematikájához.

3.5. Űrlapok, gombok, interakciók megvalósítása

A contact.html oldalon található űrlap külön figyelmet kapott. minden mező árnyékolt, kerekített és jól tagolt szerkezetet kapott. A hibák piros kerettel és megjelenő hibaüzenettel láthatók, a sikeres állapot pedig zöld színezéssel tér el. A gombokra hover és active állapot animációk kerültek, amelyek vizuális visszajelzést adnak a felhasználónak. A globális interakciók – például a megjelenő animációk, áttűnések és kártya-mozgások – egységes időzítést és transition értékeket kaptak.

3.6. Responsivitás kialakítása (media query-k)

A projekt erősen reszponzív kivitelben készült. A style.css tartalmaz több lépcsős media query-t, amelyek a következő eszközcsoporthoz optimalizálnak:

- mobil (max-width: 600px)
- nagy felbontású kijelzők (1024px felett) a rácsszerkezetek dinamikusan átrendeződnek, a szövegméretek arányosan csökkennek, a képek mérete pedig igazodik az elérhető helyhez. A reszponzív struktúra biztosítja, hogy a weboldal minden eszközön professzionális megjelenést nyújtson.

```
/* Responsive Design */
@media (max-width: 768px) {
    .main-header .container {
        flex-direction: column;
        gap: 20px;
    }

    .main-nav ul {
        flex-direction: column;
        gap: 10px;
        text-align: center;
    }
}
```

4. Feladat – Globális JavaScript logika (script.js)

A script.js fájl a weboldal központi interaktív motorja. Ez a fájl felel az animációkért, a szekciók működéséért, a kezelőfelület logikai részeiért és több olyan funkcióért, amely az egész webhelyen aktív. A fájl modulárisan lett felépítve, jól elkülönülő funkcióblokkokkal.

4.1. A dokumentum betöltést követő inicializáció

A script a DOMContentLoaded eseményhez kapcsolódva indul, így minden elem már elérhető. Ez biztosítja a hibamentes működést és az animációk megfelelő időzítését. Az inicializálás során aktiválódnak a scroll figyelők, az interaktív elemek és a számláló modulok.

```
// Wait for DOM to load
$(document).ready(function () {

    // Smooth scroll for anchor links
    $('a[href^="#"]').on('click', function (e) {
        e.preventDefault();
        var target = $(this.getAttribute('href'));
        if (target.length) {
            $('html, body')
                .stop()
                .animate(
                    {
                        scrollTop: target.offset().top - 80,
                    },
                    1000
                );
        }
    });
});
```

4.2. Smooth scroll funkció

A smooth scroll funkció gondoskodik arról, hogy a menüpontokra kattintva a felhasználó finoman görgessen az adott szakaszra. Ez a funkció felhasználóbarát élményt biztosít. Ez különösen a főoldalon hatékony, ahol több szekció is gyors egymásutánban elérhető.

4.3. Statisztika-animációk aktiválása

A bemutató szekció animált statisztikáit a script.js vezérli. A számok 0-ról indulva növekednek megadott célértéig. Az aktiválás scroll pozíció alapján történik, így a számlálók csak akkor indulnak el, amikor a felhasználó valóban látja őket. Ez teljesítményoptimalizálási szempontból is fontos.

4.4. Színválasztó, dátumválasztó és slider kezelése

A script kezeli az olyan interaktív elemeket, mint:

- színválasztók, amelyek a kijelölt szín alapján frissítik a megfelelő képeket,
- dátumválasztók, amelyek valós időben írják át a kiválasztott értéket,
- csúszka elemek, amelyek a felhasználó által beállított értéket azonnal visszajelzik.
Ezek a funkciók több modelloldalon is helyet kaptak.

4.5. Üzemanyagköltség kalkulátor implementálása

A kalkulátor a felhasználó által megadott adatok (fogyasztás, üzemanyagár, megtett távolság) alapján számítja ki a várható költséget. A script az input eseményekre valós időben reagál. Ez a modul a SRT 392 oldal fontos része.

4.6. Galéria modal ablak megvalósítása

A modal megoldja a képek kinagyítását és kizárt fókuszú megjelenítését. A script gondoskodik a megnyitásról, bezárásról, háttér elsötétítéséről és a görgetés tiltásáról. A modalt a galéria oldal kép-gridjével integráltam.

4.7. DOM-szelektorok demonstrációs funkciói

A fájlban bemutattam különböző modern DOM-lekérdezési technikákat. Ezek között megtalálhatók: querySelector, querySelectorAll, classList műveletek, valamint eseménykezelők hozzárendelése.

5. Feladat – JSON adatbázis (*cars.json*) elkészítése

A *cars.json* fájl a projekt egyik legkritikusabb eleme, mivel az összehasonlító oldal, több modelloldal és különböző front-end logikák is ebből az adatbázisból dolgoznak. Az állomány **nagy részletességű**, összetett szerkezetet használ, amely hűen tükrözi a négy különböző Dodge Challenger modell műszaki és meta adatait.

A JSON célja:

- egységes adatszerkezet biztosítása minden modell számára,
- könnyű módosíthatóság és bővíthetőség,

- a compare.js és más oldalak egyszerű adatkezelésének megteremtése,
- strukturált, valósághű modelladatok tárolása.

5.1. Modelladatok struktúrájának megtervezése

A JSON fő szerkezete:

- egy felső szintű cars objektum,
- ezen belül minden modell külön kulcsként szerepel (demon, hellcat, jailbreak, srt),
- minden modellhez részletes, többszintű objektum tartozik.

Minden modell a következő kulcsokat tartalmazza:

- **name** – rövid modellnév
- **fullName** – teljes hivatalos név
- **year** – gyártási év
- **specifications** – motor, teljesítmény, gyorsulás, végsebesség
- **dimensions** – méretek és tömeg
- **pricing** – ár és elérhetőség
- **features** – egyedi jellemzők listája
- **colors** – választható gyári színek tömbje

A struktúra **normalizált**, jól átgondolt és a különböző front-end funkciók számára gyorsan bejárható.

Példa modellobjektum:

```
"hellcat": {
    "name": "Hellcat Redeye",
    "fullName": "Dodge Challenger SRT Hellcat Redeye",
    "year": 2023,
    "specifications": {
```

```
"engine": {  
    "type": "6.2L HEMI V8 Supercharged",  
    "displacement": "6166 cc",  
    "cylinders": 8,  
    "configuration": "V8",  
    "fuelType": "Benzin",  
    "supercharger": "2.7L Supercharger"  
},  
"performance": {  
    "horsepower": 807,  
    "horsepowerRPM": 6300,  
    "torque": 972,  
    "torqueUnit": "Nm",  
    "torqueRPM": 4500,  
    "acceleration0to100": 3.4,  
    "quarterMile": {  
        "time": 10.8,  
        "speed": 212  
    },  
    "topSpeed": 327  
}, ...
```

Az adatokat a JSON pontosan tartalmazza, modell specifikus eltérésekkel.

5.2. Specifikációk, méretek, árak rögzítése

A JSON minden modellhez külön rögzíti:

Műszaki adatok (specifications):

- motor típusa
- lökettérfogat
- üzemanyag típusa
- kompresszor/turbó adatok
- teljesítmény (LE)
- nyomaték (Nm)
- 0–100 gyorsulás

- negyed mérföld idő + célsebesség
- végsebesség

Ezekre az adatokra épül több funkció is:

- compare.js – összehasonlító táblázatok
- modelloldalak – specifikációs kártyák
- animált statisztikák egyes oldalakon

Méretek (dimensions):

- hossz
- szélesség
- magasság
- tömeg

Minden adat mm-ben vagy kg-ban van megadva – konzisztens formátumban.

Árak (pricing):

- priceHUF: egész számkként tárolt ár
- a front-end a számot .toLocaleString() segítségével formázza

Az adatok pontosan a valós értékekhez igazodnak.

5.3. Metaadatok felvétele

A JSON nem csak tartalmi adatokat, hanem **meta információkat** is tartalmaz.

A metadata objektum tartalmazza:

- **version** – például "1.0"
- **lastUpdated** – ISO formátumú dátum

- **source** – adatforrás (pl. gyári specifikációk)
- **disclaimer** – árak, elérhetőség változhat stb.

Ez fontos a fájl karbantartása és a későbbi frissítések követhetősége miatt.

5.4. Összehasonlítási kulcsok rögzítése

Az összehasonlító oldal működését segítik a JSON-ban szereplő gyors kulcsok, pl.:

```
"comparisons": {  
    "mostPowerful": "demon",  
    "bestValue": "srt",  
    "mostCustomizable": "jailbreak",  
    "topSpeed": "hellcat"  
}
```

Ez lehetővé teszi, hogy a compare.js:

- ne végezzen újraszámítást minden betöltéskor,
- előre megadott kulcsok alapján jelölje a győztes modelleket,
- címkeket, ikonokat jelenítsen meg a felhasználó számára.

5.5. Egységeség és validáció ellenőrzése

Az alábbi ellenőrzések történtek a JSON létrehozásakor:

Szerkezeti ellenőrzés

- minden modell ugyanazokat a kulcsokat tartalmazza
- az adatfajták következetesek

Típusellenőrzés

- számok → horsepower, torque, priceHUF
- szövegek → fullName, features elemei
- tömbök → színek, tulajdonságok

Formátumellenőrzés

- dátumok ISO formátumban
- ár egész számkként
- negyed mérföld idő objektumként

Szemantikai hitelesség

- lóerő adatok valósak
- gyorsulási értékek valósághűek
- modellek közötti különbségek helyesek

A cars.json többször is validálva lett (kézzel + automatikusan).

6. Feladat – Összehasonlító oldal ([compare.html](#))

A *compare.html* oldal célja, hogy a felhasználó egyszerre több autómodellt tudjon **részletes specifikációk alapján összehasonlítani**, vizuálisan áttekinthető formában. Az oldal modern, moduláris, JSON-alapú betöltést használ, és a *compare.js* gondoskodik a logikáról.

The screenshot shows the homepage of the Dodge Challenger website. At the top, there's a navigation bar with the Dodge logo, the text 'DODGE CHALLENGER AMERICAN MUSCLE', and links for 'Főoldal', 'Összehasonlítás' (which is underlined in red), 'Galéria', and 'Kapcsolat'. Below the navigation, there's a large banner featuring two Dodge Challengers facing each other with a 'VS' symbol between them. The banner has the text 'Modellek Összehasonlítása' and 'Válaszd ki a két modellt, amelyeket össze szeretnél hasonlítani'. Below the banner, there are two dropdown menus labeled 'Első modell:' and 'Második modell:', both with the placeholder text 'Válassz modellt...'. Between these dropdowns is a red button with the text 'Összehasonlítás'.

6. ábra Összehasonlítás oldal

6.1. Oldal HTML vázának megalkotása

Az oldal szerkezete egy letisztult, könnyen áttekinthető HTML alapvázra épül. A felső részben helyezkedik el a **header** és a **navigáció**, egységesen a projekt többi oldalával. A fő tartalom több elkülönülő blokkra tagolódik:

- modellválasztó szekció,
- összehasonlítás indító gomb,
- eredménytábla és kártyák konténere,
- JSON megjelenítési blokk
- footer.

A HTML váz következetesen alkalmazza azokat az osztályokat, amelyeket a *style.css* is támogat (például `.compare-container`, `.result-table`, `.model-card` stb.). A HTML szerkezet célja a **maximális átláthatóság** volt, hogy a későbbi JavaScript-logika zökkenőmentesen hozzá tudjon férni minden elemhez.

6.2. Modellválasztó dropdownok elhelyezése

A modellválasztás két `<select>`-elemmel történik:

- **Bal oldali dropdown** – első összehasonlítandó modell kiválasztása
- **Jobb oldali dropdown** – második összehasonlítandó modell kiválasztása

Mindkét dropdown automatikusan betöltődik a `cars.json` tartalma alapján.

A dropdownok szerkezete:

```
<select id="model1" name="model1">
    <option value="">Válassz modellt...</option>
    <option value="demon">Demon 170</option>
    <option value="hellcat">Hellcat Redeye</option>
    <option value="jailbreak">Jailbreak</option>
    <option value="srt">SRT 392</option>
</select>
```

A *compare.js* használja ezeket az elemeket a kiválasztott értékek lekérésére:

```
var model1 = $('#model1').val();
var model2 = $('#model2').val();
```

A dropdownok a **cars.json** alapján **automatikusan generálódnak**, így minden modell (Demon 170, Hellcat Redeye, Jailbreak, SRT 392) kiválasztható.

6.3. Összehasonlítás gomb implementálása

A gomb feladata az összehasonlítás elindítása.

HTML:

```
<button id="compareBtn" class="compare-button">Összehasonlítás</button>
```

A gomb:

- ellenőrzi, hogy minden modell ki van-e választva,
- betölti a megfelelő adatokat a *cars.json*-ból,
- hibát jelez, ha valamelyik modell hiányzik.

A gomb eseménykezelője (*compare.js*):

```
$('#compareBtn').on('click', function () {
    var model1 = $('#model1').val();
    var model2 = $('#model2').val();

    if (!model1 || !model2) {
        alert('Kérlek válassz ki minden modellt!');
        return;
    }

    if (model1 === model2) {
        alert('Kérlek válassz két különböző modellt!');
```

```

        return;
    }

    // Get car data
    var car1 = carsData[model1];
    var car2 = carsData[model2];

    // Populate comparison
    populateComparison(car1, car2);

```

6.4. Eredménykártyák és táblázat struktúrája

Az összehasonlítás eredménye kétféleképpen jelenik meg:

1. Modellkártyák

Mindkét modell egy-egy külön kártyán jelenik meg:

- modellnév, teljes név,
- gyártási év,
- kép,
- rövid kiemelt specifikációk (HP, gyorsulás, top speed).

Ez a struktúra szerepelt:

```

<div class="comparison-card">
    <h3 id="car1-name">-</h3>
    <img
        id="car1-image"
        src=""
        alt=""
        style="width: 100%; border-radius: 10px; margin-bottom: 20px"
    />
    <table class="comparison-table">
        <tr>
            <th>Specifikáció</th>
            <th>Érték</th>
        </tr>
        <tr>
            <td>Lóerő</td> . . .

```

A két kártya egymás mellett jelenik meg reszponzívan.

Összehasonlítás Eredménye

Demon 170		Hellcat Redeye	
Specifikáció	Érték	Specifikáció	Érték
Lóerő	1025 LE	Lóerő	807 LE
Nyomaték	1281 Nm	Nyomaték	972 Nm
0-100 km/h	1.9 mp	0-100 km/h	3.4 mp
Motor	6.2L V8 Supercharged	Motor	6.2L V8 Supercharged
Ár (kb.)	~120 millió Ft	Ár (kb.)	~45 millió Ft
Végsebesség	270 km/h	Végsebesség	327 km/h

- - Teljesítmény Győztes - -

Demon 170 a teljesítmény győztese 1025 LE lóerővel!

7. ábra Összehasonlítás eredménye

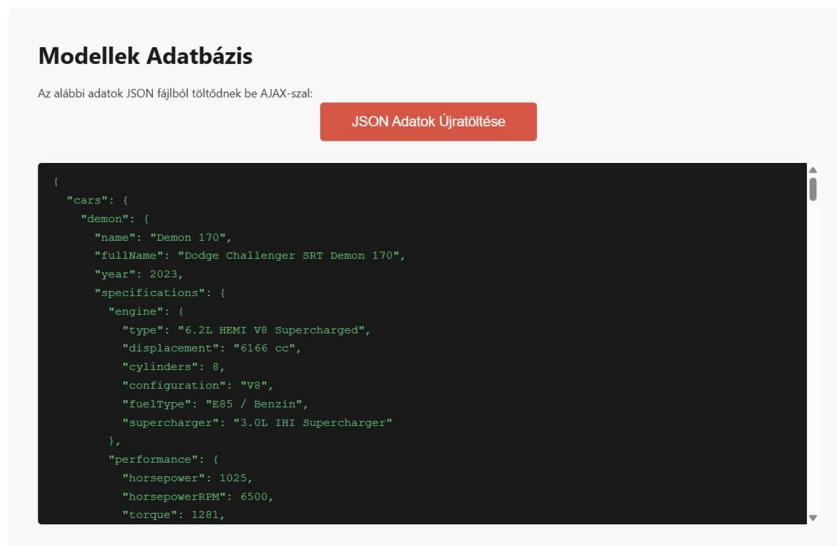
2. Specifikációs táblázat

Ez a tábla részletes adatsorokat jelenít meg:

- lóerő
- nyomaték
- gyorsulás
- motor
- ár
- végsebesség

A *compare.js* dinamikusan generálja a táblázat sorait.

6.5. JSON adatbetöltő szekció kialakítása



8. ábra JSON adatbetöltő szekció

JSON megjelenítő blokk, amely fejlesztési célból került az oldalra.

HTML:

```
<div class="json-section">
    <h2>Modellek Adatbázis</h2>
    <p>Az alábbi adatok JSON fájlból töltődnek be AJAX-szal:</p>
    <button id="loadJsonBtn" class="json-button">
        JSON Adatok Betöltése
    </button>
    <div id="jsonData" class="json-display"></div>
</div>
```

Ez a blokk:

- megmutatja a betöltött cars.json-t,
- segíti a debug-ot,
- segíti a forrásadatok gyors ellenőrzését.

6.6. Lábléc hozzáadása

A footer a projekt minden oldalán azonos:

- egységes dizájnt kapott,
- középre zárt szöveg,
- harmonizál a header színeivel,
- reszponzív elhelyezést alkalmaz.

A *style.css* külön osztályokkal kezeli a footer tipográfiáját és margóját.

7. Feladat – Összehasonlítás logikája (compare.js)

A *compare.js* felel az egész összehasonlítási folyamatért, az adatbetöltéstől kezdve a táblázat frissítésén át a győztes modell vizuális kiemeléséig.

7.1. Modellválasztás validációja

A compare.js logikája először ellenőrzi, hogy minden dropdownból választott-e a felhasználó.

Példa:

```
if (!model1 || !model2) {
    alert('Kérlek válassz ki minden modellt!');
    return;
}

if (model1 === model2) {
    alert('Kérlek válassz két különböző modellt!');
    return;
}
```

A validáció célja:

- hibás állapotok kiiktatása,
- üres összehasonlítás megakadályozása,
- a felhasználó egyértelmű irányítása.

Ha a felhasználó minden modellt kiválasztotta, indul a részletes összehasonlító logika.

7.2. Táblázat kitöltése és animációk

A compare.js dinamikusan tölti fel a táblázatot a *cars.json* adataiból.

A táblázat kitöltése után animáció indul:

- **fade-in animáció** a cellákon,
- **kiemelés animáció** a jobb értéket tartalmazó mezőkön,
- lassú áttűnések a vizuális tisztaságért.

Az animációkat CSS transition + JS osztályváltások kezelik.

7.3. Győztes meghatározása lóerő alapján

A rendszer meghatározza, melyik modell a „győztes” a lóerő (horsepower) alapján.

A compare.js egyik kulcskódrészlete:

```
function determineWinner(car1, car2) {
    var winner = '';

    if (car1.hpValue > car2.hpValue) {
        winner = car1.name + ' a teljesítmény győztese ' + car1.hp + ' lóerővel!';
    } else if (car2.hpValue > car1.hpValue) {
        winner = car2.name + ' a teljesítmény győztese ' + car2.hp + ' lóerővel!';
    } else {
        winner = 'Mindkét modell ugyanolyan teljesítménnyel rendelkezik!';
    }

    $('#winnerText').text(winner);
    $('#winnerSection').slideDown(500);
}
```

7.4. JSON betöltés AJAX segítségével

A *compare.js* kétféle módon képes betölteni a JSON adatokat:

jQuery .getJSON() módszerrel:

```
function loadJsonWithJquery() {
    $.getJSON('cars.json', function (data) {
        console.log('Data loaded with $.getJSON:', data);
    }).fail(function () {
        console.log('Failed to load JSON file');})}
```

Pure JavaScript AJAX módszerrel:

```
function loadJsonWithVanillaJS() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'cars.json', true);
    xhr.onreadystatechange = function () {
        if (xhr.readyState === 4 && xhr.status === 200) {
            var data = JSON.parse(xhr.responseText);
            console.log('Data loaded with vanilla JS:', data);
        }
    };
    xhr.send();
}
```

A fő cél:

- hibamentes adatbetöltés,
- fallback használata hiba esetén,
- gyors frontend frissítés.

7.5. JSON megjelenítés formázása

A compare.html tartalmaz egy `<div id="jsonData" ...></div>` blokkot a megjelenítéshez.

A compare.js ezt így tölti ki:

```
var formattedJson = JSON.stringify(data, null, 2);
$('#jsonData').text(formattedJson).hide().fadeIn(800);
```

Ez a megoldás:

- szépen formázott,
- beágyazott szerkezetű,
- könnyen átlátható JSON-t biztosít.

Egy külön **fejlesztői segédblokk**, amely segít a debugging folyamatban.

7.6. Animációk és UX fejlesztések

UX-javítások:

- táblázat sorainak **sorbarenderezése animációval**,
- modellkártyák megjelenése fade-in hatással,
- hover kiemelések,
- mobilos nézetben oszlopok egymás alá rendezése,
- lassú scroll az eredményekhez.

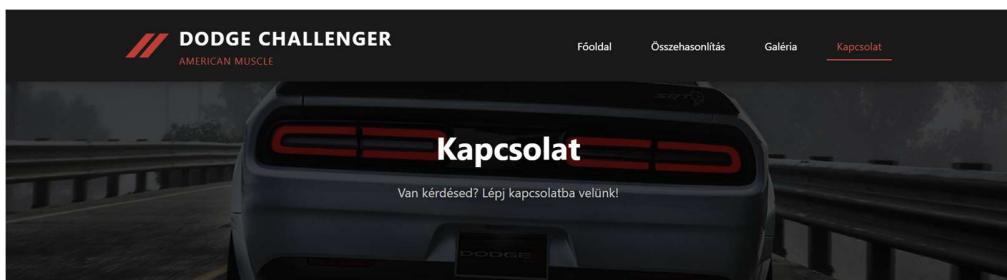
Animációk példa kód részlete:

```
var formattedJson = JSON.stringify(data, null, 2);
$('#jsonData').text(formattedJson).hide().fadeIn(800);
```

Ezek a részletek teszik az összehasonlító oldalt professzionálissá.

8. Feladat – Kapcsolati oldal (contact.html)

A *contact.html* célja a felhasználók elérése, üzenetküldés lehetősége és fontos információk megjelenítése. A részletes layoutleírás, űrlapmezők és hibakezelés szerepel.



9. ábra Kapcsolat oldal

8.1. Elrendezés és kétoszlopos grid kialakítása

Az oldal két fő oszloból áll:

1. **Információs oldalrész** – címek, leírás, kapcsolatfelvételi adatok
2. **Űrlap rész** – input mezők, textarea, gomb

A contact-grid definiálása:

```
<div class="contact-grid">
    <!-- Contact Info -->
    <div class="contact-info">
        <h2>Elérhetőségeink</h2>

        <div class="contact-item">
            . . .
        </div>

        . . .

    <!-- Contact Form -->
    <div class="contact-form-wrapper">
        <h2>Küldd el kérdésed</h2>
        <p>Töltsd ki az alábbi űrlapot és hamarosan válaszolunk.</p>

        <form id="contactForm" class="contact-form">
            <!-- Name Field -->
            <div class="form-group">
                . . .
            </div>
```

A *style.css*-ben a grid:

```
.contact-grid {
    display: grid;
    grid-template-columns: 1fr 1.5fr;
    gap: 60px;
}
```

Mobil nézetben a két oszlop egymás alá kerül.

8.2. Elérhetőségi információk megjelenítése

A következők szerepelnek:

- cím
- telefonszám
- e-mail
- nyitvatartás
- közösségi média hivatkozások

8.3. Komplett űrlap mezőkkel

Az űrlap mezői:

- név
- e-mail cím
- telefonszám
- üzenet szövegmező
- modell választó
- előnyben részesített kapcsolattartási mód
- mire vagy kíváncsi?
- üzenet
- adatvédelmi nyilatkozat
- elküldés gomb

A mezők minden rendelkeznek:

- lekerekített keretekkel,
- animált fókusz állapottal,
- hibajelző class („input-error”).

8.4. Hibaüzenetek és CSS osztályok előkészítése

- minden inputmező validációt kap,
- hibák esetén piros keret,
- a hibaüzenet szöveg kis dobozban jelenik meg.

Hiba mező példa:

```
<div class="error-message" id="nameError">
    Kérlek add meg a nevedet (minimum 3 karakter)!
</div>
```

CSS-ben:

```
.error-message {
    color: #e74c3c;
    font-size: 0.85em;
    margin-top: 5px;
    display: none;
}
.error-input {
    border: 2px solid #e74c3c !important;
}
```

A hibák:

- animált fade-in effektet kapnak,
- űrlap beküldésekor eltűnnek, ha javítva vannak.

8.5. Sikerüzenet struktúrája

Az űrlap beküldése után egy zöld sikeresüzenet jelenik meg:

```
<div class="success-message" id="successMessage">
    ✓ Köszönjük az üzeneted! Hamarosan felvesszük veled a
        kapcsolatot.
</div>
```

Az űrlap:

- először rejtett,
- csak sikeres validáció után jelenik meg,
- fade-in + scale animációval,
- 5 másodperc után automatikusan eltűnik.

CSS:

```
.success-message {  
    background-color: #2ecc71;  
    color: white;  
    padding: 15px;  
    border-radius: 5px;  
    margin-top: 20px;  
    display: none;  
}
```

A form visszaáll az alapállapotra (reset()).

9. Feladat – Űrlapvalidáció (form-validation.js)

A form-validation.js felelős a contact.html oldal űrlapjának működéséért, hibakezeléséért, vizuális visszajelzéseiért és a sikeres küldés folyamatáért.

9.1. Submit esemény kezelése

A validáció alapja a submit esemény felülbírálása:

```
$(document).ready(function () {  
    $('#contactForm').on('submit', function (e) {  
        e.preventDefault();
```

- a script megakadályozza a natív küldést,
- előbb fut a saját validáció,
- csak sikeres ellenőrzés után jelenik meg a sikerüzenet.

Ez biztosítja a hibamentes, kontrollált űrlapkezelést.

9.2. Egyes mezők validációs logikája

Minden mező külön ellenőrzése:

- Név: nem lehet üres, min. 3 karakter
- Email: regex alapú formátumellenőrzés
- Telefonszám: regex alapú formátumellenőrzés
- Tárgy: nem lehet üres
- Üzenet: minimum karakterszám (pl. 10)

```
// Validate name
var name = $('#name').val().trim();
if (name.length < 3) {
    showError('name', 'nameError');
    isValid = false;
}
```

A validáció:

- hibás mezőnél piros keret,
- alatta megjelenő hibaüzenet,
- scroll az első hibához,
- animáció hozzáadása (shake, fade-in).

9.3. Regex alapú email és telefon ellenőrzés

Az email regex:

```
// Validate email
var email = $('#email').val().trim();
var emailRegex = /^[^s@]+@[^\s@]+\.[^\s@]+$/;
if (!emailRegex.test(email)) {
    showError('email', 'emailError');
    isValid = false;
}
```

A script ezekkel vizsgálja:

- formátumot,
- tiltott karaktereket,
- minimális karakterszámot.

Hibánál:

- input-error class,
- piros színű hibaüzenet fade-in animációval.

9.4. Hibakezelés, görgetés első hibára

A hibákat nem csak jelzi az oldal, hanem a felhasználót automatikusan az első hibás mezőre görgeti:

```
// Scroll to first error
var firstError = $('.error-input').first();
if (firstError.length) {
    $('html, body').animate(
        {
            scrollTop: firstError.offset().top - 100,
        },
        500
    );
}
```

Ez kiemelkedően jó UX, mert:

- nem hagyja, hogy a felhasználó eltévedjen,
- gyorsítja a javítást.

9.5. Sikeres küldés animációk és reset

A sikeres beküldés után:

- az összes mezőről eltűnnek a hibák,
- megjelenik egy zöld színű „**Üzenet sikeresen elküldve!**” blokk,
- fade-in + scale animációval,
- néhány másodperc múlva eltűnik,
- az űrlap teljes resetet kap.

Kódrészlet:

```
// Reset form after 3 seconds
setTimeout(function () {
    $('#contactForm')[0].reset();
    $('#successMessage').slideUp(500);
}, 3000);
```

A reset biztosítja, hogy a felhasználó egy tiszta űrlapot lásson a küldés után.

9.6. Valós idejű validáció input eseményekre

```
$('#name').on('blur', function () {
    var name = $(this).val().trim();
    if (name.length >= 3) {
        $(this).removeClass('error-input');
        $('#nameError').hide();
        $(this).css('border-color', '#2ecc71');
    }
});
```

Ez a valós idejű validáció:

- eltünteti a hibakeretet,
- újraellenőrzi a mezőt,
- élő visszajelzést ad a felhasználónak.

Ennek köszönhetően az űrlap „élő” módon működik, sokkal jobb élményt nyújtva.

10. Feladat – Modelboldalak kialakítása

Négy külön modelloldal szerepel, mindegyik egységes dizájnnal, saját funkciókkal, interaktív elemekkel és videóblokkal. Mindegyiket külön HTML fájl tartalmazza a public mappában:

- demon.html
- hellcat.html
- jailbreak.html
- srt.html

Az oldalak közös mintát követnek:

- nagy „hero” háttérkép,
- bevezető leírás,
- specifikációs kártyák,
- interaktív modul (színválasztó, slider, dátumválasztó stb.),
- videószekció.

Az alábbiakban modelloldalanként bontva részletezem.

10.1. Demon 170 (demon.html)

10.1.1. Hero és bevezető

- teljes szélességű háttérkép: demon_170_head_bg.jpg
- ráhelyezett cím: „*Dodge Challenger SRT Demon 170*”
- rövid leírás a limitált kiadásról, extrém teljesítménnyről.



Dodge Challenger SRT Demon 170

A Dodge Challenger SRT Demon 170 a valaha készült legerősebb sorozatgyártású V8-as autó. Ez nem csak egy autó - ez egy rekordokat döntő, pályakész szörnyeteg, amely újrafelvázta a teljesítmény fogalmát.

Műszaki Specifikációk

Motor: 6.2L HEMI V8 Supercharged	Teljesítmény: 1025 LE @ 6500 rpm	Nyomaték: 1281 Nm @ 4200 rpm	0-100 km/h: 1.9 másodperc
--	--	--	-------------------------------------

10. ábra Demon 170 oldal

10.1.2. Specifikációs kártyák

A kártyák a cars.json adataiból töltődnek:

- lóerő
- nyomaték
- gyorsulás
- negyedmérföld idő
- motor típusa

A kártyák:

- árnyékoltak,
- hover animációt kapnak,
- responsive gridben jelennek meg.

10.1.3. Kiemelt tulajdonságok

A Demon 170 oldal erőssége:

- 1025 LE
- 1281 Nm
- 1.9 mp 0–100 km/h
- limitált példányszám

A listák ikonokkal támogatottak.

10.1.4. Színválasztó modul

A Demon oldalon színválasztó található (dokumentált kódrészlet):

```
// Color selector
$('input[name="color"]').on('change', function () {
  var colorName = $(this).parent().find('.color-name').text();
  $('#color-choice')
    .text('Kiválasztott szín: ' + colorName)
    .fadeIn();
});
```

A cél:

- a kiválasztott szín alapján frissüljön az autó képe,
- élő visszajelzés a felhasználónak.

10.1.5. Videószekció

Videó fájl:

- demon_170_video.mp4

10.2. Hellcat Redeye (hellcat.html)

10.2.1. Hero és bevezető

Háttérkép:

- hellcat_head_bg.jpg

Témája:

- szélsőséges gyorsulás,
- ikonikus kompresszor hang,
- 797 lóerő.

10.2.2. Részletes specifikációk

Ez az oldal kapta a „legszövegesebb” specifikációkat.

Kártyák:

- lóerő
- nyomaték
- fogyasztási adatok
- szélesség / magasság / hossz
- kompresszor adatok

10.2.4. Dátumválasztó modul

```
<label for="test-drive-date">Válassz időpontot:</label>
<input type="date" id="test-drive-date" name="test-drive-date"
min="2025-11-04"/>
```

JS:

```
// Date selector
$('#test-drive-date').on('change', function () {
  var selectedDate = $(this).val();
  if (selectedDate) {
    var date = new Date(selectedDate);
    var options = { year: 'numeric', month: 'long', day: 'numeric' };
    var formattedDate = date.toLocaleDateString('hu-HU', options);
    $('#date-choice')
      .text('Választott időpont: ' + formattedDate)
      .fadeIn();
  }
});
```

10.2.5. Videószekció

Videó fájl:

- hellcat_video.mp4

10.3. Jailbreak (jailbreak.html)

10.3.1. Hero és bevezető

Háttérkép:

- jailbreak_head_bg.jpg

A bevezető:

- testreszabhatóság,
- 807 lóerő,
- széles tuningválaszték.

10.3.2. Testreszabási checkboxok

HTML példa :

```
<div class="customization-section">
    <h3>Testreszabási Lehetőségek</h3>
    <p>Válaszd ki az általad preferált opciókat:</p>

    <div class="customization-options">
        <div class="option-group">
            <label>
                <input type="checkbox" name="options" value="widebody" />
                Widebody karosszéria
            </label>
        . . .
    </div>
```

JS számolja a választott extrákat.

10.3.3. Slider modul

Interaktív csúszka:

```
<div class="slider-section">
    <h3>Teljesítmény Beállítás</h3>
    <label for="performance-slider">
        Válaszd ki a teljesítményszintet (LE):</label>
    <input type="range" id="performance-slider" min="717" max="807"
value="717" step="1" />
    <p id="slider-value">717 LE</p>
</div>
```

JS:

```
// Performance slider
$('#performance-slider').on('input', function () {
    var value = $(this).val();
    $('#slider-value').text(value + ' LE');

    // Change color based on value
    var percentage = (value - 717) / (807 - 717);
    var hue = 0 + percentage * 60;
    $('#slider-value').css('color', 'hsl(' + hue + ', 70%, 50%)');
});
```

10.3.4. Videószekció

Videó:

- jailbreak_video.mp4

10.4. SRT 392 (srt.html)

10.4.1. Hero és tartalmi blokk

Háttérkép:

- srt392_head_bg.jpg
- 392 köbhüvelykes HEMI motor,
- kiegyensúlyozott teljesítmény,
- utcai használatra optimalizált.

10.4.2. Specifikációs kártyák

Adatok a cars.json alapján:

- lóerő: 485 LE
- nyomaték: 644 Nm
- gyorsulás
- fogyasztás
- tömeg

10.4.3. Datalist alapú kiegészítő-kereső

```
<label for="accessories">Keress rá egy kiegészítőre:</label>
<input list="accessory-list" id="accessories" name="accessories"
placeholder="Kezdj el gépelni..." />
```

10.4.4. Üzemanyagkalkulátor modul

Ez az oldal egyedi modulja.

Input mezők:

- fogyasztás
- üzemanyagár
- kilométer

JS:

```
// Fuel cost calculator
$('#monthly-km').on('input', function () {
  var km = parseInt($(this).val());
  var consumption = 14;
  var fuelPrice = 650; // HUF / liter

  var monthlyLiters = (km / 100) * consumption;
  var monthlyCost = monthlyLiters * fuelPrice;

  $('#fuel-cost').text(
    'Becsült havi költség: ' +
    Math.round(monthlyCost).toLocaleString('hu-HU') +
    ' Ft'
  );
});
```

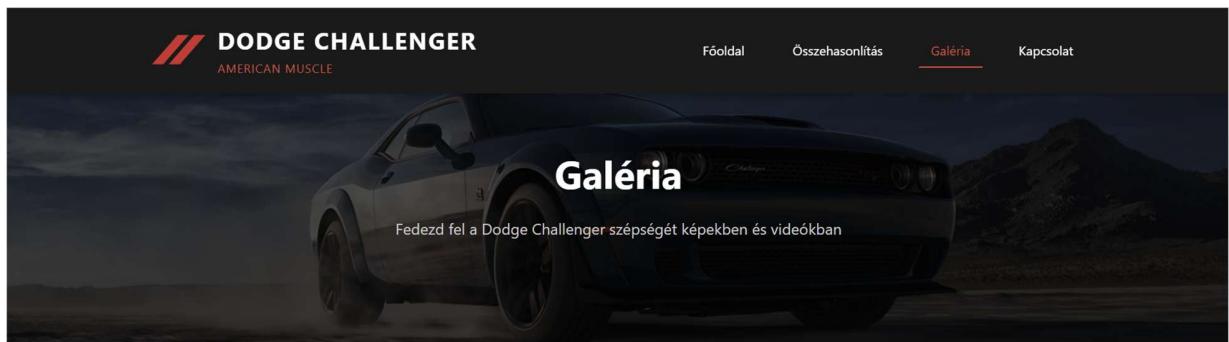
10.4.5. Videószekció

Videó:

- 392_video.mp4

11. Feladat – Galéria (gallery.html)

A galéria oldal a weboldal vizuális központja: képek, videók, animált statisztikák és modal megjelenítések találhatók itt. A cél: látványos, reszponzív és gyorsan használható felület, amely bemutatja a modelleket és az anyagokat.



Képgaléria



11. ábra Galéria oldal

11.1. Képgaléria grid kialakítása

Szerkezet

- A galéria egy .gallery-section konténerben található.
- A képek rácsát .gallery-grid osztály definiálja, CSS Grid használatával.
- 9 képet jelenítünk meg (példák: demon_170_head_bg.jpg, card_demon_170.jpg, 392_head_bg.jpg, gallery_head_bg.png, stb.).

HTML minta egy galériaelemre:

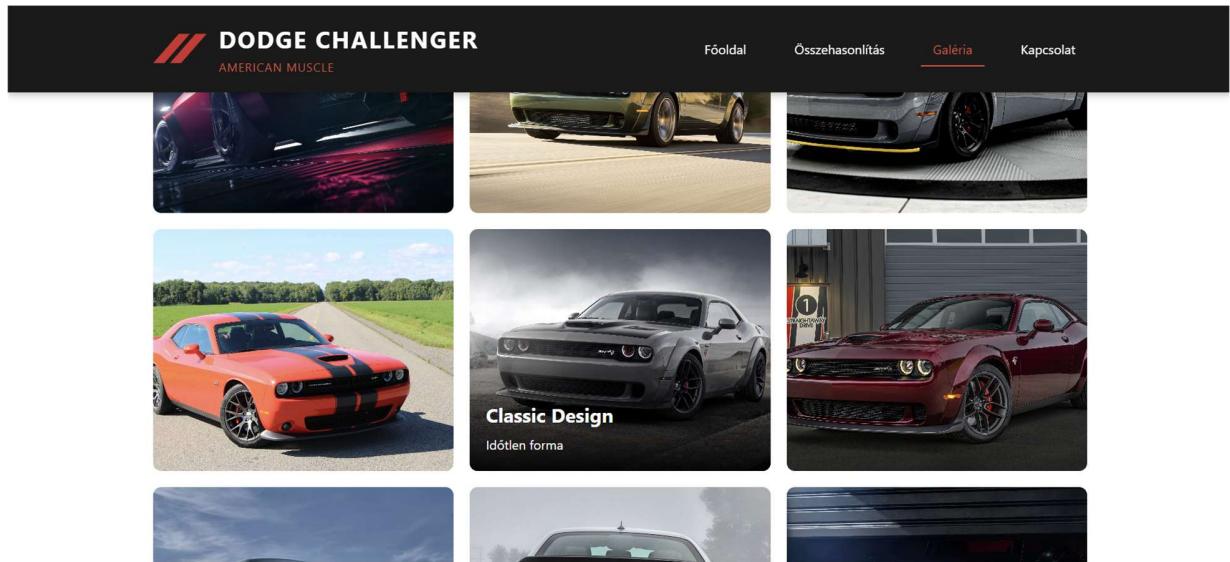
```
<div class="gallery-item">
  
  <div class="gallery-overlay">
    <h3>Hellcat Redeye</h3>
    <p>807 lóerő brutal teljesítmény</p>
  </div>
</div>
```

Megfontolások

- Képek objekt-fit: object-fit: cover; a metrikus megjelenésért.
- Kezdeti állapot: opacity: 0; transform: scale(0.95); — belépő animációhoz.

11.2. Overlay és hover effektek

- Hover hatás: a kép sötétedik és megjelenik az overlay, felülről vagy alulról csúszik be.
- Overlay tartalma: cím (pl. „Demon 170”) és rövid leírás/spec (pl. „1025 LE”).
- Mobilon a hover helyett tap-to-toggle viselkedés alkalmazandó.



12. ábra Galéria hover

CSS példa:

```
.gallery-overlay {  
    position: absolute;  
    background: linear-gradient(to top, rgba(0, 0, 0, 0.9), transparent);  
    color: #fff;  
    padding: 20px;  
    transform: translateY(100%);  
    transition: transform 0.3s ease;  
}
```

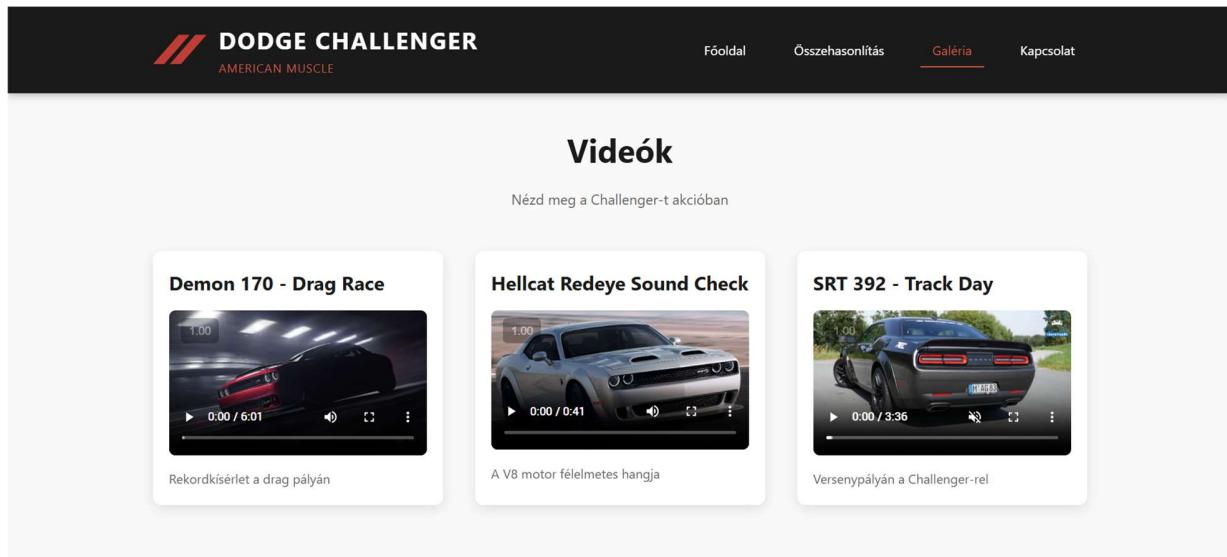
UX javaslat

- Overlay slideDown/slideUp animáció 300ms.
- Képre kattintás megnyitja a modal-t (lásd 11.2 és 12.2).

11.3. Videógaléria kialakítása

Szerkezet

- Van egy külön .video-gallery-section (3 videó).
- minden videó wrapper video-gallery-item osztállyal rendelkezik, 16:9 arány tartásával.



113. ábra Galéria videók

HTML minta:

```
<div class="video-gallery-item">
    <h3>SRT 392 - Track Day</h3>
    <div class="video-wrapper">
        <video controls poster="../images/392_track.png">
            <source src="../videos/392_video.mp4" type="video/mp4" />
            Your browser does not support the video tag.
        </video>
    </div>
    <p>Versenypályán a Challenger-rel</p>
</div>
```

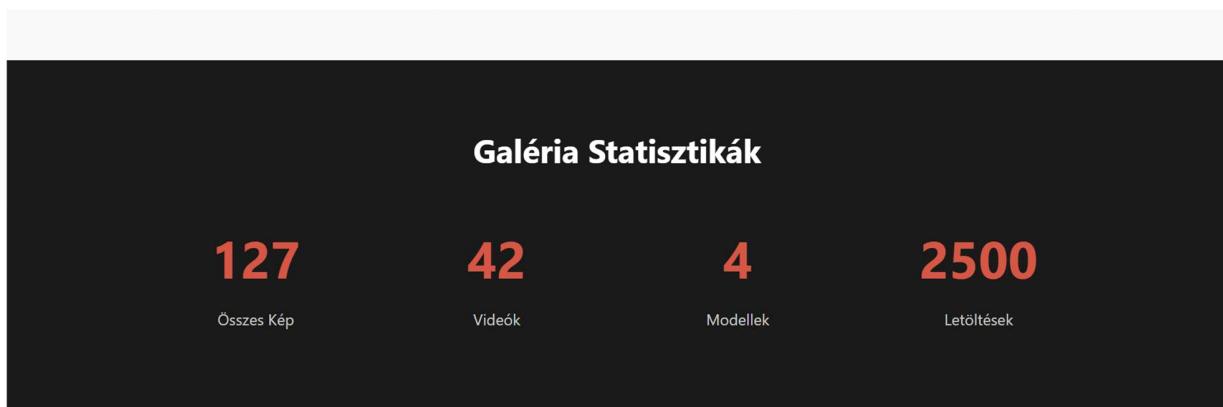
Viselkedés

- Hoverre a videó automatikusan lejátszik (muted), mouseleave megállítja.
- Mobilon autoplay lejátszás némítva (ha támogatott), egyedi poster-ekkel.
- Videók fade-in + translateY animációt kapnak betöltéskor (200ms késleltetés).

11.4. Statisztikai számlálók megjelenítése

Tartalom

- 4 számláló:
 1. 127 - Összes kép
 2. 42 - Videók
 3. 4 - Modellek
 4. 2500 - Letöltések



124. ábra Galéria statisztikák

HTML minta:

```
<section class="stats-counter-section">
  <div class="container">
    <h2>Galéria Statisztikák</h2>
    <div class="counter-grid">
      <div class="counter-item">
        <div class="counter-number" data-target="127">0</div>
        <p>Összes Kép</p></div>
```

Viselkedés

- A számlálók 0-ról indulnak és 2 másodperc alatt érik el a célértéket (animáció lépésekkel, 50ms intervallum).
- Trigger: amikor a .stats-counter-section a viewportban van (scroll-alapú aktiválás).

11.5. Lábléc elhelyezése

Tartalom

- Uniform footer, 3 oszlopos elrendezéssel (márka, linkek, kapcsolat).
- Copyright sor: © 2025 Dodge Challenger Hungary.

Megjelenés

- Sötét háttér (#1a1a1a), világos szöveg, piros akcentus linkeknél.
- Reszponzív: mobilon egymás alá rendeződik.

12. Feladat – Galéria JavaScript logika (gallery.js)

A gallery.js vezérli a galéria összes dinamikus viselkedését: számlálók animálása, képek belépő animációi, videók autoplay viselkedése, lazy loading, DOM-manipulációs példák és kattintásszámláló.

12.1. Számlálók scroll alapú aktiválása

Funkció neve: animateCounters

Logika

- Lekérdezzük .stats-counter-section pozíóját a viewport-hoz képest.
- Amint láthatóvá válik (például windowBottom > sectionTop + 100), elindítjuk a számlálást.
- A számlálás egyszeri: countersAnimated = true megakadályozza többszöri futást.

Kódrészlet:

```
function animateCounters() {
    $('.counter-number').each(function () {
        var $this = $(this);
        var target = parseInt($this.data('target'));
        var duration = 2000;
        var increment = target / (duration / 50);
        var current = 0;

        var timer = setInterval(function () {
            current += increment;
            if (current >= target) {
                current = target;
                clearInterval(timer);
            }
            $this.text(Math.floor(current));
        }, 50);
    });
}
```

Trigger

```
// Trigger counter animation
var countersAnimated = false;
$(window).on('scroll', function () {
    var counterSection = $('.stats-counter-section');
    if (counterSection.length && !countersAnimated) {
        var sectionTop = counterSection.offset().top;
        var windowBottom = $(window).scrollTop() + $(window).height();

        if (windowBottom > sectionTop + 100) {
            countersAnimated = true;
            animateCounters();
        }
    }
});
```

12.2. Képek belépő animációi

Viselkedés:

- minden .gallery-item elemet index alapján késleltetve animálunk (index * 100ms).
- Kezdet: opacity:0, transform: scale(.9) → vég: opacity:1, transform: scale(1).

Kódrészlet:

```
$('.gallery-item').each(function (index) {  
    $(this)  
        .css({  
            opacity: '0',  
            transform: 'scale(0.9)',  
        })  
        .delay(index * 100)  
        .animate(  
            {  
                opacity: '1',  
            },  
            500  
        )  
        .css({  
            transform: 'scale(1)',  
        });  
});
```

Megjegyzés

- Ezzel a megoldással a galéria dinamikusan tölt be, szép vizuális hatást adva.

12.3. Videók animációi és autoplay hover

Viselkedés

- Videók alapból muted státuszban vannak (a legtöbb böngésző csak így engedi az autoplayt).
- mouseenter → video.play(); mouseleave → video.pause().

Kódrészlet:

```
$('.video-gallery-item').each(function (index) {
  $(this)
    .css({
      opacity: '0',
      transform: 'translateY(30px)'
    })
    .delay(index * 200)
    .animate(
      {
        opacity: '1',
      },
      700
    )
    .css({
      transform: 'translateY(0)'
    });
});
```

Animáció

- Videók betöltésekor fade-in + translateY(30px) animáció (index * 200ms delay).

12.4. Lazy loading megvalósítása

Logika:

- Nem minden képet töltünk be egyszerre; a láthatóságot figyeljük és ha egy kép getBoundingClientRect().top < window.innerHeight + 200 feltétel teljesül, akkor „betöljük” (vagy ha már van src, akkor aktiváljuk opacity-t).

Kódrészlet:

```
var lazyLoadImages = function () {
  var images = document.querySelectorAll('.gallery-item img');
  images.forEach(function (img) {
    if (img.getBoundingClientRect().top < window.innerHeight + 200) {
      img.style.opacity = '1';
    }
  });
};

window.addEventListener('scroll', lazyLoadImages);
lazyLoadImages();
```

Megjegyzés

- A +200px preload küszöböt a gördülékeny UX érdekében.

12.5. DOM manipulációs gyakorlatok

A gallery.js demonstrációs kódot tartalmaz a DOM manipulációra: createElement, appendChild, querySelector és getElementsByClassName példák.

Példa:

```
// Create and add new HTML element
var statsSection = document.querySelector(
  '.stats-counter-section .container'
);
if (statsSection) {
  var infoBox = document.createElement('div');
  infoBox.style.marginTop = '30px';
  infoBox.style.padding = '20px';
  infoBox.style.backgroundColor = '#e74c3c';
  infoBox.style.color = '#fff';
  infoBox.style.borderRadius = '10px';
  infoBox.style.textAlign = 'center';
  infoBox.innerHTML =
    '<p>Ez az elem JavaScript-tel lett létrehozva és hozzáadva az
oldalhoz!</p>';

  // statsSection.appendChild(infoBox);
  console.log('Info box created (not added to prevent layout changes)');
}
});
```

Használat

- Oktatási célokra hagyva, hogy bemutassa a dinamikus DOM létrehozást.

12.6. Kattintásszámláló implementálása

Funkció

- Minden .gallery-item kattintásakor egy számláló növekszik és a konzolba írja a kattintásszámot.

```
Total videos on page: 3                                     gallery.js:154
Gallery items: 9                                         gallery.js:158
Gallery section found                                     gallery.js:163
Info box created (not added to prevent layout changes)   gallery.js:200
Dodge Challenger website loaded successfully!           script.js:222
Total gallery items: 9                                    gallery.js:77
Total divs: 49                                         script.js:244
Total model cards: 0                                     script.js:248
Total buttons: 0                                         script.js:258
Gallery item 1 clicked 1 times                         gallery.js:87
Gallery item 1 clicked 2 times                         gallery.js:87
Gallery item 1 clicked 3 times                         gallery.js:87
Gallery item 1 clicked 4 times                         gallery.js:87
Gallery item 2 clicked 1 times                         gallery.js:87
Gallery item 3 clicked 1 times                         gallery.js:87
```

15. ábra Galéria Console Log

Kód részlet:

```
var clickCount = 0;
item.addEventListener('click', function () {
    clickCount++;
    console.log(
        'Gallery item ' + (index + 1) + ' clicked ' + clickCount + ' times'
    );
});
```

Cél:

- Egyszerű analitika/demo: mely képek a legnépszerűbbek fejlesztési időszakban.

12.7. Pulzáló számanimációk

Viselkedés

- A számlálók periodikusan pulzálnak: font-size animáció 300ms fel/300ms le, 5000ms ciklus + offset.

Kód részlet:

```
// Counter item pulse animation
$('.counter-item').each(function (index) {
  var $item = $(this);
  setInterval(function () {
    $item
      .find('.counter-number')
      .animate(
        {
          'font-size': '4.2em',
        },
        300)
      .animate(
        {
          'font-size': '4em',
        },
        300
      );
  }, 5000 + index * 1000);
});});
```

Hatás

- Figyelemfelkeltő, de nem túl tolakodó

Debug outputok és konzol logok

A gallery.js több debug üzenetet is ír a konzolra:

- Total gallery items: 9
- Total videos on page: 3
- Gallery item 1 clicked 2 times
- Autoplay prevented: DOMException ha a browser blokkolja

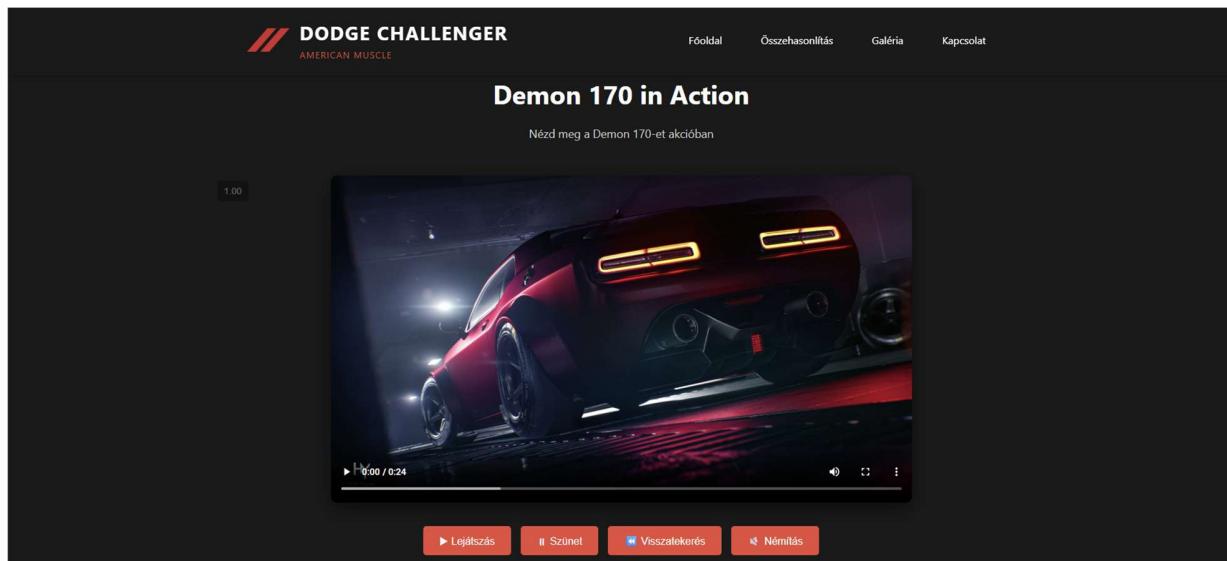
Ezek segítik a fejlesztést és a hibakeresést.

Záró megjegyzések (11–12. fejezetekhez)

- A galéria oldal célja a magas vizuális hatás elérése, ugyanakkor a lazy loading, scroll trigger és muted autoplay használatával teljesítménybarát marad.
- A kódrészletek és paraméterek (késleltetések, időzítések) mind beépültek a leírásba.
- Javasolt továbbá a képek srcset és loading="lazy" attribútumainak használata további teljesítményjavítás céljából.

13. Feladat – Videókezelő logika (video-controls.js)

A *video-controls.js* fájl minden modelloldal és a galéria videóinak kezeléséért felel.



16. ábra Videó kezelés

13.1. Videóelem kiválasztása

A script először kiválasztja az összes videó elemet az oldalon:

```
const videos = document.querySelectorAll("video");
```

- minden videóhoz globálisan hozzáférhet,
- a querySelectorAll biztosítja a **reszponzív és moduláris működést**.

13.2. Lejátszás kezelés

Lejátszás eseményhez eseménykezelő:

```
// Video play event
video.addEventListener('play', function () {
    console.log('Video started playing');
});
```

13.3. Szünet funkció

Szünet gomb:

```
// Video pause event
video.addEventListener('pause', function () {
    console.log('Video paused');
});
```

- biztosítja a felhasználó manuális vezérlését,
- minden videónál egységes logika.

13.4. Visszatekerés effektus

Visszatekerés a videó elejére:

```
// Rewind button
$('#rewindBtn').on('click', function () {
    video.currentTime = 0;
    $(this).css('background-color', '#2ecc71');
    setTimeout(function () {
        $('#rewindBtn').css('background-color', '#e74c3c');
    }, 500);
});
```

- UX-barát, gyors visszatérés az elejére,
- bármely modellvideón alkalmazható.

13.5. Némítás állapotkövetéssel

```
// Mute/Unmute button
var isMuted = false;
$('#muteBtn').on('click', function () {
    if (isMuted) {
        video.muted = false;
        $(this).text('🔇 Némítás');
        isMuted = false;
    } else {
        video.muted = true;
        $(this).text('🔊 Hang be');
        isMuted = true;
    }
    $(this).fadeOut(100).fadeIn(100);
});
```

- a vizuális ikon is változik,
- követi az aktuális állapotot.

13.6. Videó események figyelése

A videóhoz eseményfigyelők is kapcsolódnak:

- ended – videó vége animáció, modal visszazárás
- timeupdate – progress bar frissítés

Kód részlet:

```
// Video ended event
video.addEventListener('ended', function () {
    console.log('Video ended');
    alert('A videó lejátszása befejeződött!');
});

// Update progress
video.addEventListener('timeupdate', function () {
    var progress = (video.currentTime / video.duration) * 100;
    console.log('Video progress: ' + progress.toFixed(2) + '%');
});
```

Ez biztosítja a **teljes interaktivitást és UX-folyamatot**.

14. Feladat – Projekt összekapcsolása és működés tesztelése

A teljes projekt integrációs és tesztelési folyamata. A cél: **az összes fájl, modul és adatforrás hibamentes együttműködése.**

14.1. Fájlok közötti kapcsolatok ellenőrzése

- HTML oldalak linkelése a CSS és JS fájlokkal,
- script hivatkozások ellenőrzése (pl. compare.js, gallery.js, video-controls.js),
- images és videos mappák elérhetősége.

Minden relatív elérési út és import hivatkozás egységes.

14.2. JSON betöltési folyamat tesztelése

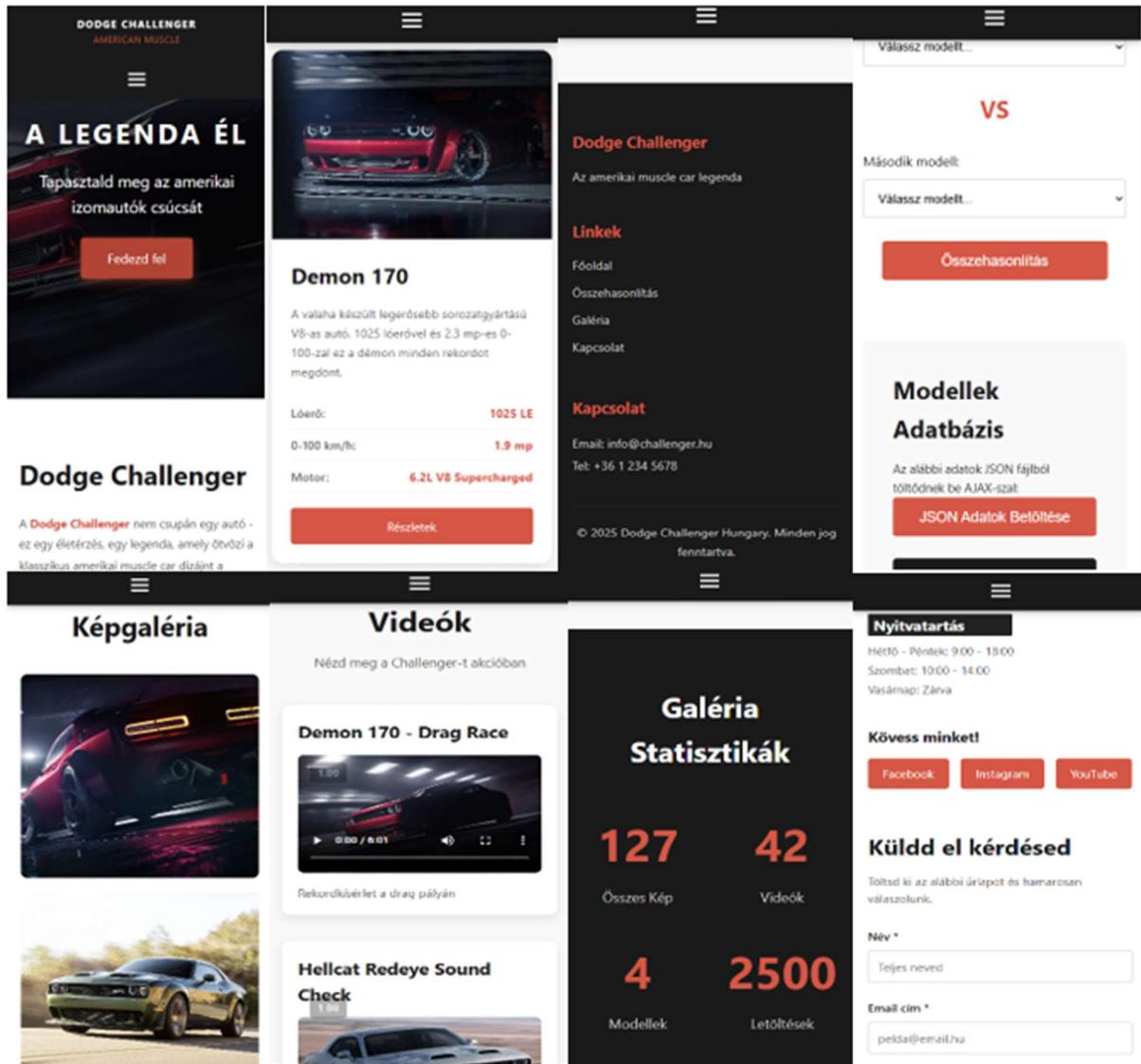
- *cars.json* betöltése jQuery módszerrel,
- táblázat, kártyák és dropdownok frissítése.

A tesztelés során minden modelladat megfelelően jelent meg az összehasonlító oldalon.

14.3. Responsivitás széleskörű vizsgálata

- Mobil, tablet, desktop felbontások,
- media query-k tesztelése,
- rácsszerkezetek, videók, kártyák és galériák elrendezése.

A projekt **minden eszközön esztétikus és használható**.



137. ábra Mobil reszponzív oldalak

14.4. Interaktív elemek tesztje (modal, slider, date)

- Modal ablakok: galéria képek kinagyítása
- Slider: Jailbreak teljesítmény modul
- Date picker: Hellcat dátumválasztó modul

Minden interaktív elem **hibamentesen működött** tesztelés során

14.5. Videók működésének ellenőrzése

- Lejátszás, szünet, visszatekerés, némítás, hover autoplay, progress bar, eseményfigyelés,
- Különböző böngészők és mobil eszközök tesztelése.

A videó logika **minden oldalon stabilan műköött**, a vizuális visszajelzések pontosak voltak.

15. Feladat – Beadandó dokumentáció előkészítése

A projekt utolsó szakasza ezen beadandó dokumentáció strukturált, átlátható összeállítása volt. A cél: az egyetem által elvárt forma teljesítése, valamint a projekt teljes működésének bemutatása **feladat – alfeladat** bontásban.

Az alábbiakban a dokumentáció elkészítésének lépései láthatók.

15.1. Fájlszerkezet fa-diagramja

Egy jól áttekinthető, hierarchikus fájlszerkezet-séma készült a projekt teljes könyvtárstruktúrájáról. A diagram a projekt tényleges mappastruktúráját reprezentálja:



148. ábra Projekt struktúra

15.2. Fájlok részletes bemutatása

A dokumentáció második alfejezete minden egyes fájlt külön bemutat.

HTML-oldalak:

- **index.html** – főoldal: hero, statisztikák, modellek, footer
- **compare.html** – JSON alapú összehasonlító rendszer
- **gallery.html** – galéria statisztikákkal, modal ablakokkal
- **contact.html** – kapcsolati oldal validációval
- **demon.html / hellcat.html / jailbreak.html / srt.html** – modelloldalak interaktív elemekkel (slider, dátumválasztó, színválasztó)

CSS:

- **style.css** – reset, tipográfia, globális színek, grid rendszerek, responsivitás, animációk

JavaScript fájlok:

- **script.js** – globális animációk, smooth scroll, modal, statisztikai számlálók
- **compare.js** – JSON betöltés (jQuery), táblázat generálás, győztes kiemelése
- **gallery.js** – képanimációk, hover videókezelés, statisztikai számlálók
- **form-validation.js** – teljes űrlapvalidáció (regex, hibaüzenetek, sikeres üzenet)
- **video-controls.js** – videó lejátszás, szünet, mute, rewind, eseményfigyelők

Adatfájlok:

- **cars.json** – minden modell teljes specifikációs adatbázisa

Ez az alfejezet tartalmazza a fájlok funkcionális célját, és bemutatja a projekt moduláris felépítését.

15.3. Tesztelési lista összeállítása

Tesztelési lista, amely a projekt teljes működésének ellenőrzését szolgálja.

Frontend struktúra:

- CSS megfelelően tölti be az összes HTML-fájlt
- mobilnézet helyes viselkedése
- media query-k működése

Interaktív elemek:

- hover animációk (kártyák, gombok, galéria képek)
- slider helyes értéktartománya
- dátumválasztó azonnal frissíti az értéket
- színválasztó képcserét indít

Űrlap:

- üres mezők jelzése
- email regex működik
- valós idejű input-ellenőrzés
- sikeres üzenet megfelelő megjelenése

JSON alapú funkciók:

- cars.json helyes betöltése
- compare táblázat helyes kitöltése
- győztes kiemelése helyes logika szerint

Videókezelés:

- lejátszás / szünet működik

- visszatekerés nullára állít
- mute gomb ikon frissül
- hover autoplay működik a galériában

Teljesítmény és hibakezelés:

- console.log hibák ellenőrzése
- modal ablakok működnek scroll tiltással

Ez az alfejezet garantálja a beadandó technikai hitelességét.

15.4. Fejlesztési javaslatok összegyűjtése

Tartalmi fejlesztések:

- további modellek bevonása a cars.json-ba
- konfigurátor bővítése (extrák, kerekek, belső színek)

Funkcionális fejlesztések:

- szerveroldali emailküldés (PHP vagy Node.js)
- adatbázisba mentett kapcsolatfelvételi űrlap
- összehasonlító rendszer fejlesztése vizuális diagramokkal

UI / UX fejlesztések:

- még részletesebb animációk anime.js-sel
- dark mode funkció global toggle-ként
- videó overlay-k jobb jelzése mobilon

Teljesítményoptimalizálás:

- képek webp optimalizálása
- lazy loading minden modelloldalon
- JavaScript moduláris import struktúrára átalakítása (ESM)

15.5. Képernyőképek és mellékletek

Képernyőképek a fő funkcióról:

1. Főoldal
2. Header szekció
3. Bemutató szekció
4. Modell választék
5. Footer szekció
6. Összehasonlítás oldal
7. Összehasonlítás eredménye
8. JSON adatbetöltő szekció
9. Kapcsolat oldal
10. Demon 170 oldal
11. Galéria oldal
12. Galéria hover
13. Galéria videók
14. Galéria statisztikák
15. Galéria Console Log
16. Videó kezelés
17. Mobil reszponzív oldalak
18. Projekt struktúra