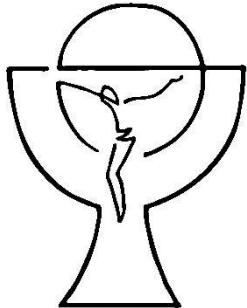




Car Merchants' Site

Dokumentáció



Gamauf Marcell Miklós

Git: /GamaufMarcell/

Back-end, adatbázis, saját területeinek
dokumentációja

Durányik Csaba

Git: /DuranyikCsaba/

Front-end, dokumentáció szervezése,
csapatvezetői feladatok

Szent István Katolikus Technikum és Gimnázium 2025



Tartalomjegyzék

Car Merchants' Site.....	1
Dokumentáció.....	1
Bevezetés:.....	3
Felhasználói dokumentáció:.....	3
Oldal és funkciók használatának részletes bemutatása (képekkel illusztrálva):.....	4
Fejlesztői dokumentáció:.....	11
Adatbázis kapcsolat.....	21
authController.....	22
Forum Controller.....	30
HirdetesekController.....	30
KommentController.....	34
authController.....	36
loadPosts().....	63
addPost().....	63
deletePost(postId: number).....	63
updatePost(postId: number).....	63
saveUpdatedPost(postId: number).....	63
addComment(postId: number).....	63
updateComment(commentId: number).....	64
saveUpdatedComment(commentId: number).....	64
deleteComment(commentId: number).....	64
toggleCommentInput(postId: number).....	64
toggleShowAllComments(postId: number).....	64
Tesztelés:.....	65
Továbbfejlesztési lehetőségek:.....	70
Irodalomjegyzék:.....	70



Bevezetés:

1. **Szoftver célja:** A célunk az volt, hogy laikusok számára egy helyen biztosítsa az autóvásárlást a fenntartáshoz szükséges információkat és irányt mutasson a bonyolult és olykor érhetetlen autóvásárlási és ügyintézési folyamatokban. Sok más használtautó weboldalt lehet találni az interneten de nem a leghatékonyabbat így átgondoltuk és bele vágtunk a mi saját használtautó oldalunkba ahol elhelyezkedik a hirdetések feltöltése mellett: **Fórum** (Regisztrációhoz majd utána bejelentkezéshez kötött) ahol autósok és nem autósok tudják megbeszélni a típus hibákat egyes autóknál ami nagyban megkönnyíti a jövendőbeli autó tulajokat a vásárlásban vagy estleges hiba feltárásban ,**Hirdetések** megtekintése nem kötött bejelentkezéshez azt az úgy nevezett „mezei” felhasználó is megtudja tekinteni de hirdetést feltölteni csak a bejelentkezett felhasználó tud, itt is megpróbáltunk arra törekedni hogy a lehető legegyszerűbben és legáttekinthetőbben lehessen hirdetést feltölteni hozzá nem értőknek is
2. **Tervezett funkciók:** Az alapvető (reg, log, quit) funkciókon kívül az oldal tartalmaz még hirdetésfeladást, profillal kapcsolatos funkciókat (pl másik felhasználó megtekintheti a profilt, pontozhatja, megbízhatósági ponszámot adhat, nézhet).

Hirdetésekre való szűrések paraméterek alapján.

Gyakori/elterjedt autók enciklopédiája, itt mi 9 „tuti tippet” mutatunk be a navigációs pánon lehet ezt kiválasztani majd ott lehet áttekinteni

Az oldal tartalmazni fog topikokkal ellátott fórumot, melyet bárki böngészhet, de csak a regisztrált felhasználók bővíthatik.

3. **Használt csoporthunka eszközök bemutatása:** Munkánk első lépéseként a **Trello**-t vettük használatba, egyszerű gyors és átlátható itt táblák és kártyák segítségével tudtuk követni egymás munkáját és mielőbbi haladást elérni a. Majd a közös munka el is kezdődött tervezünk majd használatba vettük a **Git**-et ezt a környezetet nagyon hamar magunkévá tettük és elsajátítottuk mint a **GitHub desktop**-ot és a webes felületét is ez nagyban könnyítette a munkát két kattintás után láttuk egymás munkáját, még nagy segítségünkre volt a **Discord** itt tartottuk a megbeszéléseket nem volt egy kitűzött rendszerünk hanem folyamatos haladás mellett pár perc alatt tudtunk beszálni egyszerűen képernyő megosztással

Felhasználói dokumentáció:

1. **Rendszerkövetelmények:** Mivel a szoftver böngészőben fut így nincs nagyobb rendszer követelménye mint egy Google Chrome–nak de itt ez minimum gépigény:

Windows:

- **Operációs rendszer:** Windows 7 vagy újabb
- **Processzor:** 1 GHz-es vagy gyorsabb processzor
- **RAM:** 2 GB
- **Szabad hely a merevlemezen:** Legalább 350 MB
- **Grafikai kártya:** Az alap grafikai funkciókhöz szükséges



macOS

- **Operációs rendszer:** macOS 10.11 (El Capitan) vagy újabb
- **Processzor:** Intel alapú processzor
- **RAM:** 4 GB
- **Szabad hely a merevlemezen:** Legalább 350 MB

Linux

- **Operációs rendszer:** A legfrissebb verziók valamelyikéből (pl. Ubuntu, Debian, Fedora)
- **Processzor:** 1 GHz-es vagy gyorsabb processzor
- **RAM:** 2 GB
- **Szabad hely a merevlemezen:** Legalább 350 MB

Android

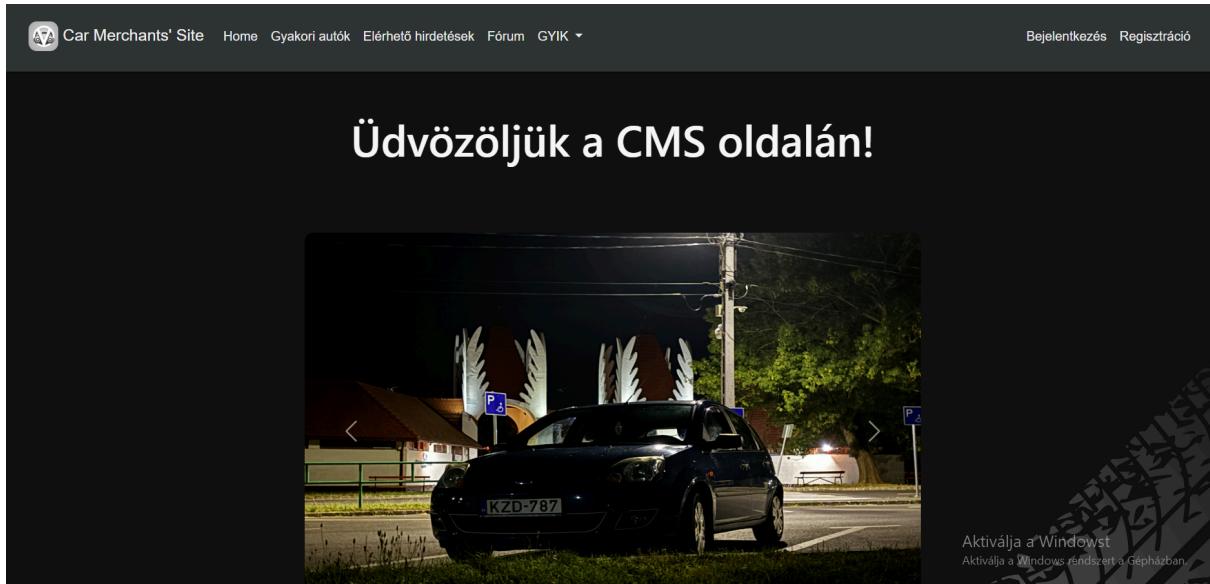
- **Operációs rendszer:** Android 5.0 (Lollipop) vagy újabb
- **Processzor:** ARM vagy x86 alapú
- **RAM:** Legalább 1 GB

iOS

- **Operációs rendszer:** iOS 12 vagy újabb
- **Eszköz:** iPhone 5s vagy újabb



Oldal és funkciók használatának részletes bemutatása (képekkel illusztrálva):



Amint belépünk az oldalra találkozunk egy üdvözlő szöveggel és egy carousel-el ez alatt található egy üdvözlő szöveg az oldalhoz:

A screenshot of the CMS homepage showing a text box containing promotional text. The text reads:

Üdvözöljük a CMS (Car Merchants' Site) weboldalon, ahol az autókereskedés világát egy új perspektívából ismerheti meg! Célunk, hogy fiatalok és laikusok számára könnyen érthető és hasznos információkat nyújtunk az autóvásárlás és -eladás folyamatáról, valamint segítséget kínálunk a megfelelő döntések meghozatalában.

Weboldalunk nem csupán egy egyszerű hirdetési platform. Fedezze fel az autópiac legújabb trendjeit, összehasonlítsa a különböző modellekét, és találja meg az ideális járművet, amely megfelel az igényeinek!

Továbbá, a CMS közösségi fóruma lehetőséget biztosít az autórajongók számára, hogy megosszák egymással tapasztalataikat, tanácsaikat és kérdéseiket. Legyen szó technikai részletekről, karbantartási tippekről vagy egyszerűen csak egy jó beszélgetésről, itt mindenki megtalálja a számítását.

Csatlakozzon hozzánk, és tapasztalja meg, milyen könnyű és elvezetés lehet az autókereskedés a CMS segítségével!

At the bottom right of the text box, there is a small watermark: "Aktiválja a Windows® rendszert a Gépházban".

Ez alatt pedig egy footer látható ahol ott vannak az elérhetőségeink illetve egy rólunk oldal:



Rólunk

Üdvözöljük a Car Merchants' Site oldalán! Célunk, hogy segítsünk a laikusoknak eligibilis információkat és iránymutatást nyújtani a bonyolult bürokratikus folyamatok lebonyolításához. Tudjuk, hogy a jogi és adminisztratív eljárások sokszor zavarosak és nehezen érhetőek lehetnek, ezért itt vagyunk, hogy megkönnyítsük az Ön számára a tájékozódást.

Weboldalunk lehetőséget biztosít autók meghirdetésére, így Ön könnyedén eladhatja vagy megvásárolhatja álmai járművét. Az autókereskedelem világában való eligibilis információk megosztása érdekében platformunkon egyszerűen létrehozhat hirdetéseket, amelyek elérhetik a potenciális vásárlókat. Célunk, hogy a járművásárlás és -eladás folyamata minél gördülékenyebb és átláthatóbb legyen.

Ezen kívül fórumot is biztosítunk, ahol a felhasználók megvitathatják egymással a különböző témaikat. Legyen szó autóvásárlásról, biztosításokról, vagy a legfrissebb jogszabályi változásokról, közösséggünk tagjai megosztják tapasztalataikat és tanácsaikat. Hiszünk abban, hogy a tudás megosztása és a közösségi támogatás segíthet mindenkinél a legjobb döntések meghozatalában.

Célunk, hogy egy olyan platformot hozunk létre, ahol a felhasználók könnyen hozzáférhetnek a szükséges információhoz, és ahol a közösség ereje segíti őket a bonyolult folyamatok során. Köszönjük, hogy velünk tart, és reméljük, hogy hasznosnak találja az oldalunkat!

Gamauf Marcell: [@](#) Durányik Csaba: [@](#) minden jog fenntartva © 2024 Aktiválja a Windows rendszert a Gépházban. Rólunk

illetve fent látunk egy navigációs sávot, itt egyből láthatóvá válik hogy nem vagyunk bejelentkezve ott egy egyszerű kattintással megtörténhet a regisztráció:

Regisztráció

Felhasználónév (5-15 karakter)

E-mail

Telefonszám (+36...)

Jelszó

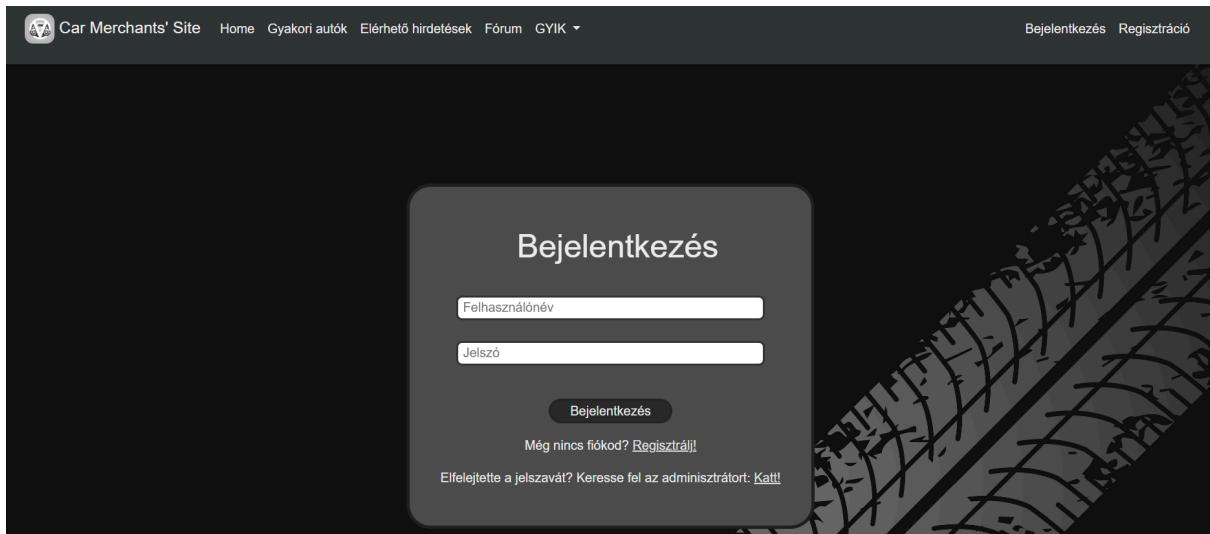
Jelszó megegyez

Regisztráció

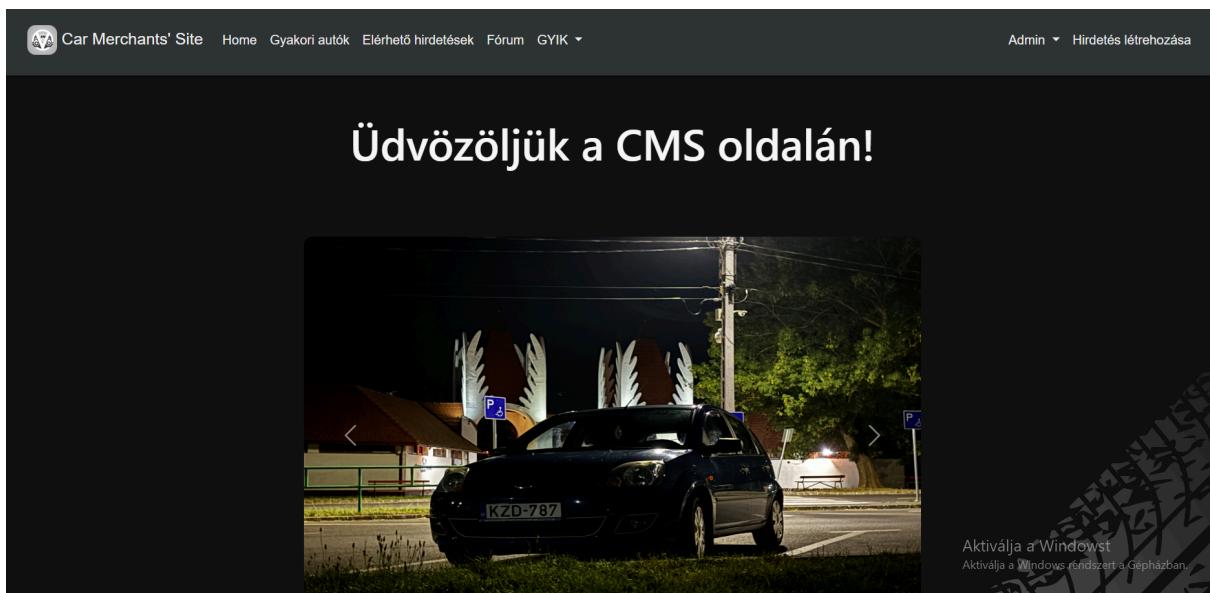
Már regisztráltál? [Jelentkezz bel!](#)

itt látható a regisztrációs felület minden mezőnek vannak kritériumai és ezeknek teljesülniük kell hogy végbemehessen a regisztráció ha sikeres a regisztráció átdob a bejelentkezés oldalra:





itt pedig már végbemehet a bejelentkezés sikeres bejelentkezés után pedig átdob a főoldalra azonban ha mégsem sikerülne mivel nincs fiókod vagy elfelejtetted a jelszavát akkor alatta 1 kattintással orvosolható mind a két probléma.



Sikeres bejelentkezés után a főoldalon a jobb felső sarokba látszik a bejelentkezett felhasználó neve illetve láthatóvá válik a hirdetések létrehozása gomb is ha a felhasználó nincs bejelentkezve akkor csak megtekinteni tudja az oldalakat azonban nincs hozzáférése.

Bejelentkezett felhasználóval lehet hirdetést létrehozni illetve a fórumon posztot létrehozni és másoké alá kommentelni de először az oldalakat mutatnám be sorba.



Suzuki Swift (2005–2010)

Előnyök:

- Megfizethető ár és alacsony fenntartási költségek.
- Megbízható motorok és egyszerű szerkezet.
- Kompakt méret, ideális városi közlekedéshez.

Hátrányok:

- Egyszerű belső tér és korlátozott felszereltség.
- Kisebb csomagtér, ami hosszabb utazásoknál lehet hátrányos.

Típushibák:

- A kipufogórendszer rozsdásodása, különösen a hátsó dob környékén.
- A gyűjtőtrafók meghibásodása, ami motorhibát eredményezhet.
- A futómű alkatrészeinek gyorsabb kopása rossz minőségű utakon.

Suzuki Swift (2005–2010)

Opel Astra H (2004–2014)

Volkswagen Golf V (2003–2008)

Toyota Yaris (XP10, 1999–2005)

Ford Focus II (2004–2011)

Renault Clio III (2005–2012)

Škoda Octavia II (2004–2012)

Hyundai i30 (2007–2012)

Kia Ceed (2006–2012)

A gyakori autók statikus oldallal kezdeném a bemutatást itt megkerestük a 9 leggyakoribb használtautó típusát a kínálatba és lenyíló menüs megoldással itt láthatóak az adott autó előnyei, hátrányai és típushibái.

Következő oldal az elérhető hirdetések:

Rover 45
Ár: 650000 ft
Hengerürtartalom: 1395 cm³
Üzemanyag: Benzin
Évjárat: 2004
Futott kilométer: 288000 km

[Részletek](#)

Ford Fiesta
Ár: 750000 ft
Hengerürtartalom: 1242 cm³
Üzemanyag: Benzin
Évjárat: 2007
Futott kilométer: 158000 km

[Részletek](#)

Volkswagen Passat b5.5
Ár: 1300000 ft
Hengerürtartalom: 1899 cm³
Üzemanyag: Dízel
Évjárat: 2003
Futott kilométer: 384600 km

[Részletek](#)

Aktiválja a Windowszt
Aktiválja Windows rendszert a Gépházban.



Ezt az oldalt egyből összekötném a jobb felső sarokba található hirdetés létrehozásával

Hirdetés Feladása

Márka
Modell
Hengerürtartalom
cm³-ben megadva
Ajtók száma
Üzemanyag
Válasszon üzemanyagon
Évjárat
Futott kilométer
km-ben
Szín
Sebességváltó típusa
Válasszon sebességváltó típust

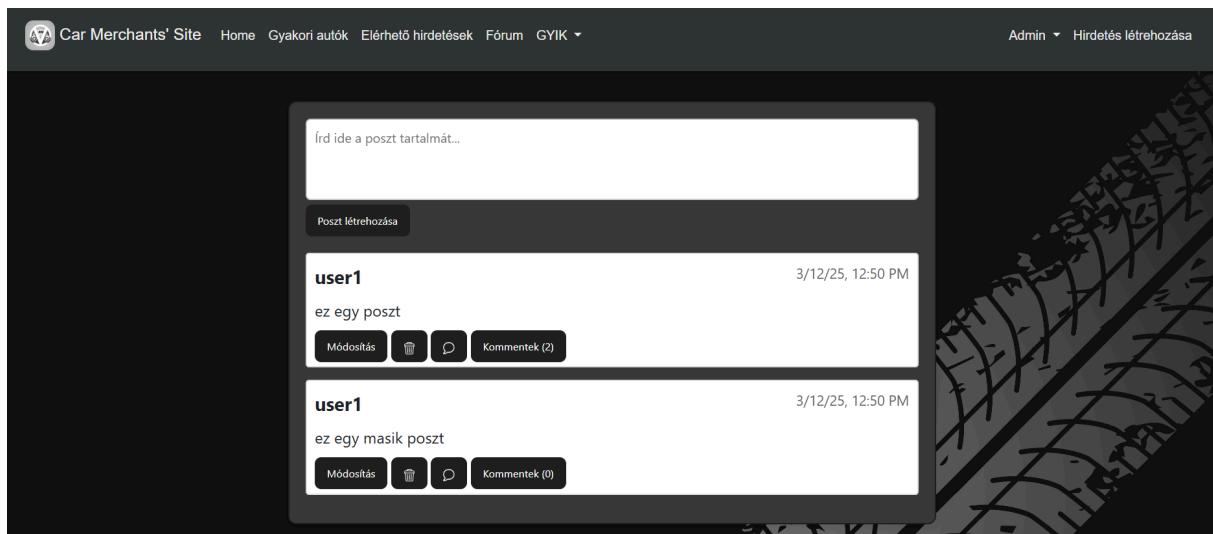
Leírás
(max 255 karakter)
Kiegészítők
Műszaki vizsga érvényessége
éééé. hh. nn.
Baleseti előzmények
Ár
Telefonszám
+36xxxxxxxx
Képek
Fájlok kiválasztása Nem lett kiválasztva fájl
Egy hirdetés időtartama 2 héti!
A hirdetés később csak annak törlésével majd újból létrehozásával módosítható!

Hirdetés Feladása

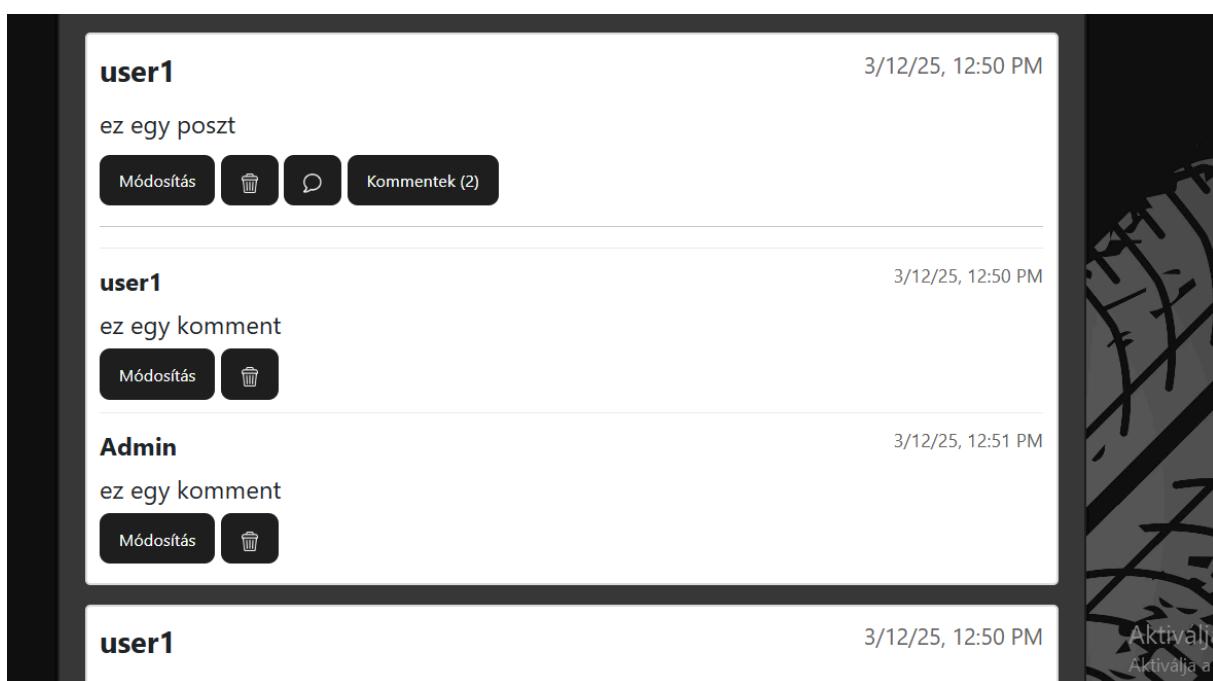
Itt lehet a bejelentkezett felhasználónak hirdetést létrehozni itt a csillagozott mező kitöltése kötelező és csak ezek után lehet feladni a hirdetést feladás után az elérhető hirdetesek oldalra kerül és ott válik láthatóvá

A navigációs sáv következő eleme a fórum itt is csak a bejelentkezett felhasználó tud posztot létrehozni és kommentelni





így néz ki az oldal felül tud a felhasználó posztot létrehozni a másik felhasználó ha kommentelni szeretne a komment ikonra nyom majd tudja is írni a kommentet a kommentek fül lenyitásával lehőszer látható a komment hogy letisztultan láthassuk a posztokat a mindenki a saját maga posztját tudja módosítani és kommentelni



A fórum mellett bedik egy GYIK oldal található ahol alapvető autós kérdésekre található válasz és segítségnyújtás a kezdő vagy éppen haladó autó vásárlóknak

The screenshot shows the homepage of the 'Car Merchants' Site'. At the top, there's a navigation bar with links: 'Car Merchants' Site', 'Home', 'Gyakori autók', 'Elérhető hirdetések', 'Fórum', and 'GYIK ▾'. On the right side, there's a sidebar with a red header 'Gépjármű átiratás' and a list of items: 'Adásvételi minta', 'Kötelező biztosítás', 'Casco tudnivalók', and 'Betétlap'. The main content area features a large white text 'Üdvözölj' and 'MS olda' on a dark background.

This screenshot shows the 'Gépjármű Átirása' (Bill of Lading) section of the GYIK page. It starts with a heading 'Gépjármű Átirása' and a sub-section 'A gépjármű átirás menete:'. Below this is a numbered list of requirements:

- Adásvételi szerződés megkötése:** Az eladó és a vevő négy példányban tölték ki és írják alá az adásvételi szerződést. Ebből kettő az eladónál, kettő a vevőnél marad.
- Eredetiségvizsgálat elvégzése:** A vevőnek kötelező elvégeztetnie az eredetiségvizsgálatot.
- Kötelező biztosítás megkötése:** Az átirás előtt a vevőnek rendelkeznie kell érvényes kötelező gépjármű-felelősségbiztosítással.
- Okmányirodai ügyintézés:** Az adásvételt követően 15 napon belül a vevőnek be kell jelentenie a tulajdonjog változást a kormányablaknál.

Below the list, there's a section titled 'Szükséges dokumentumok:' with three items: 'Adásvételi szerződés', 'Eredetiségvizsgálati bizonyítvány', and 'Kötelező biztosítás igazolása'.

This screenshot shows the 'Adásvételi Szerződés Minta' (Model of Purchase Agreement) section of the GYIK page. It includes a heading 'Adásvételi Szerződés Minta' and a sub-section 'Fontos tudnivalók az adásvételi szerződésről:'. Below this are several examples of required documents:

- Példányszám:** Négy eredeti példány szükséges; kettő az eladónál, kettő a vevőnél marad.
- Kitöltés:** minden adatot pontosan és olvashatóan kell kitölteni, különös tekintettel a jármű és a felek adataira.
- Aláírás:** minden példányt mindenki férne alá kell írnia.

Below these, there's a section titled 'Az adásvételi szerződés tartalma:' with several examples of document contents:

- Felek adatai:** Az eladó és a vevő neve, címe, telefonszáma, és személyazonosító adatai.
- Jármű adatai:** A jármű típusa, gyártmánya, évjára, rendszáma, alvázsáma és egyéb azonosító adatai.
- Ár:** A jármű vételára, valamint a fizetés módja és határideje.
- Átadás-átvétel:** A jármű átadásának időpontja és helyszíne.
- Nyilatkozatok:** Az eladó nyilatkozata a jármű állapotáról, terhekről, és arról, hogy a jármű nem lopott.



 Car Merchants' Site Home Gyakori autók Elérhető hirdetések Fórum GYIK ▾

user ▾ Hirdetés létrehozása

Kötelező Biztosítás

Lényeges információk a kötelező biztosításról:

Megkötés ideje: A tulajdonjog átruházását követően azonnal, de legkésőbb az átírás előtt meg kell kötni.

Díjfizetés: A biztosítás díját rendszeresen, a szerződésben meghatározott ütemezés szerint kell fizetni.

Igazolás: Az érvényes biztosítás meglétét igazolni kell az átírás során.

Kötelező biztosítás típusai:

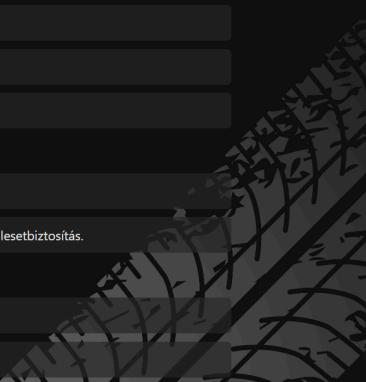
Alap kötelező biztosítás: A legkisebb fedezet, amely a harmadik félnek okozott kárra vonatkozik.

Extra fedezetek: Kiegészítő biztosítások, amelyek védelmet nyújtanak a saját járműben keletkezett károkra is, például balesetbiztosítás.

Kárbejelentés folyamata:

Első lépés: A baleset helyszínén rögzítsd a körülmenyeket, és készíts fényképeket.

Második lépés: Tölts ki a kárbejelentő lapot, amelyet a biztosítód biztosít.



 Car Merchants' Site Home Gyakori autók Elérhető hirdetések Fórum GYIK ▾

user ▾ Hirdetés létrehozása

Casco Tudnivalók

A Casco biztosítás előnyei:

Önkéntes biztosítás: Nem kötelező, de ajánlott a saját gépjármű védelme érdekében.

Káresemények fedezése: Védelmet nyújt többek között lopás, törés, elemi kár és vandalizmus esetén.

Díjazás: A díj mértéke függ a jármű típusától, értékétől és a választott fedezetektől.

Casco biztosítás típusai:

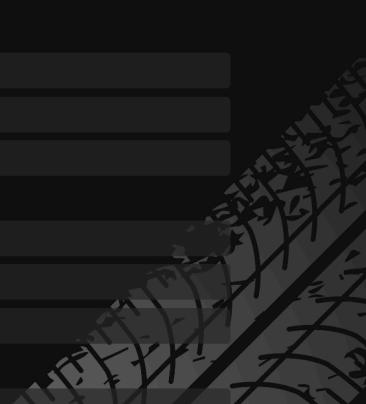
Teljes casco: minden káreseményre kiterjed, beleértve a saját hibás baleseteket is.

Részleges casco: Csak bizonyos káreseményekre terjed ki, például lopásra és tüzkárra.

Választható kiegészítők: Kiegészítő biztosítások, mint például balesetbiztosítás, jogi védelem, vagy bérkoci biztosítás.

Kárbejelentés folyamata:

Első lépés: A baleset helyszínén rögzítsd a körülmenyeket, és készíts fényképeket.



 Car Merchants' Site Home Gyakori autók Elérhető hirdetések Fórum GYIK ▾

user ▾ Hirdetés létrehozása

Betétlap

Betétlap használata baleset esetén:

Célja: A baleset körülmenyeinek rögzítése a felek által a helyszínen.

Kitöltés módja: Mindkét fél közösen tölti ki, ügyelve a pontos és részletes adatrögzítésre.

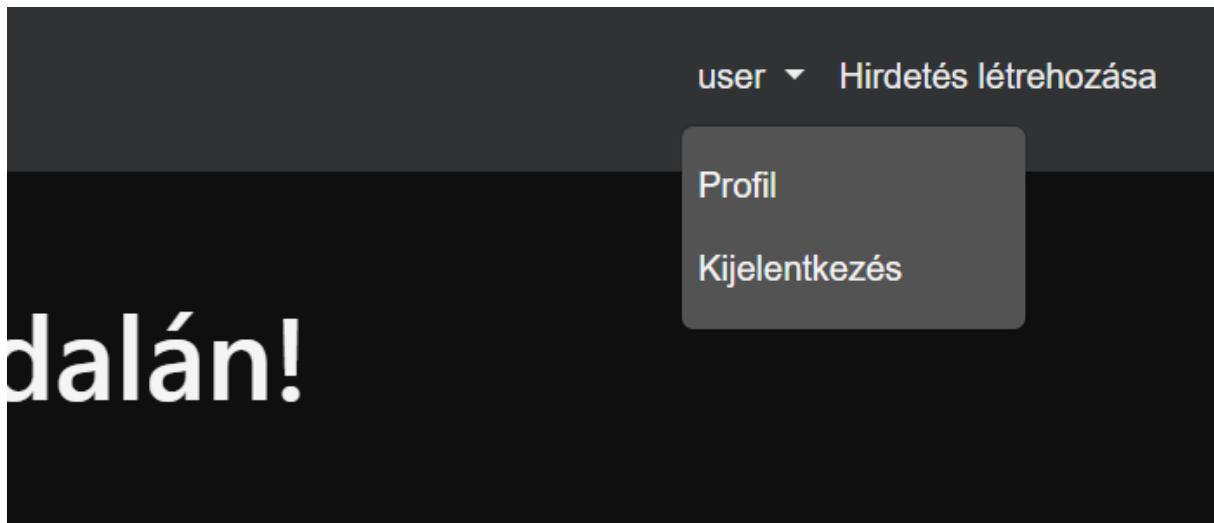
Jelentősége: Segíti a biztosítók számára a felelősségi megállapítását és a kár rendezését.

A betétlap kitöltésének lépései:

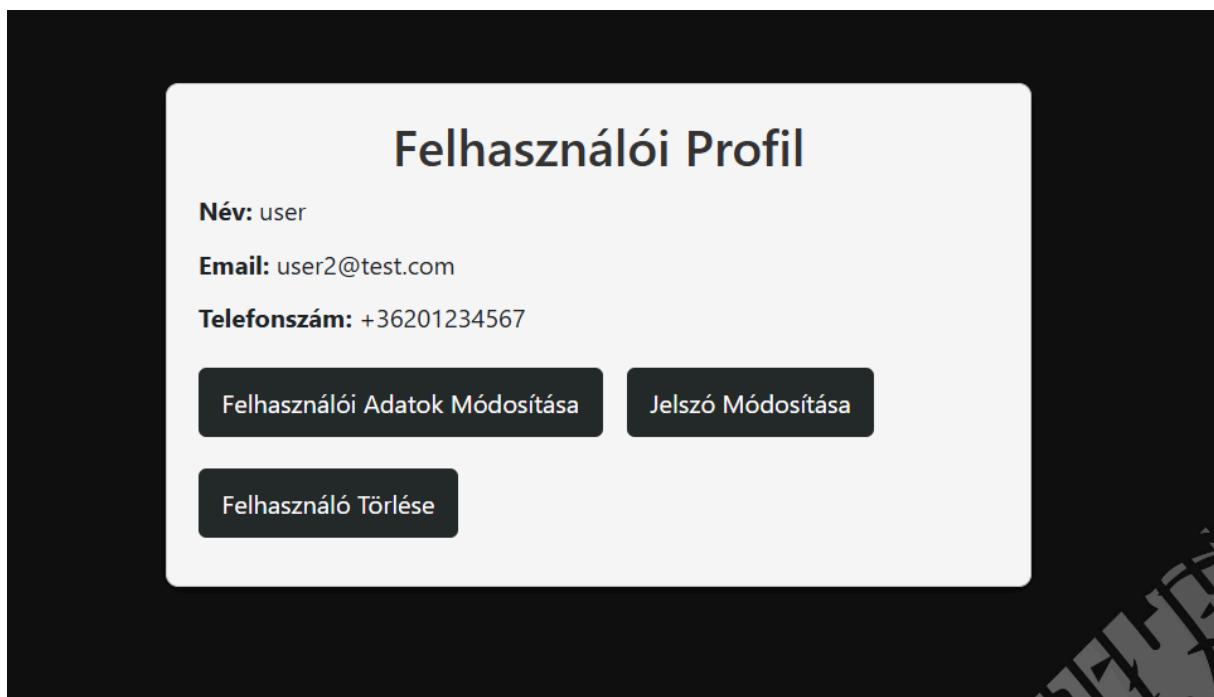
- Azonosítás:** A balesetben részt vevő felek azonosítása, név, cím, telefonszám, és biztosító adatok rögzítése.
- Baleset körülmenyei:** A baleset helyszínének, időpontjának és körülmenyeinek részletes leírása.
- Járművek adatai:** A balesetben érintett járművek adatai, mint például a rendszám, gyártmány, típus és évjárat.
- Tanúk:** Ha vannak tanúk, az ó nevük és elérhetőségük rögzítése.
- Fényképek:** A baleset helyszínéről és a járművekről készült fényképek készítése.



Következő amit bemutatnák a felhasználó kezelése



Az alap felhasználó a profil pontra kattintás után megnyílik:



itt pedig lehet adatot módosítani vagy jelszót illetve felhasználót törlni
most pedig az admin funkciók:

The screenshot shows a dark-themed admin dashboard. At the top right, there's a dropdown menu labeled "Admin" and the text "Hirdetés létrehozása". Below this is a vertical navigation bar with four items: "Profil" (highlighted in red), "Felhasználók", "Moderátorok", and "Kijelentkezés". On the left side of the screen, the text "Oldalán!" is displayed in large white letters.

itt egyből több funkció ugrik fel a felhasználók fül alatt:

The screenshot shows the "Felhasználók kezelése" (User Management) section. It displays a table with user information:

ID	Név	Email	Telefonszám	Akcíák
4	user	user2@test.com	+36201234567	<button>Törölés</button> <button>Promótálás</button>
5	user2	user3@test.com	+36201234567	<button>Törölés</button> <button>Promótálás</button>

látjuk az összes bejelentkezett felhasználót itt lehet törlni vagy pedig moderátorrá tenni
moderátor fül alatt pedig a moderátorokat látjuk és le tudjuk fokozni

The screenshot shows the "Moderátorok kezelése" (Moderator Management) section. It displays a table with moderator information:

ID	Név	Email	Telefonszám	Akcíák
2	user1	user1@test.com	+36201234567	<button>Lefokozás</button>



Fejlesztői dokumentáció:

1. *Telepítés, beindítás menete:*

1. A telepítéshez szükség van az alkalmazás GitHub repository-jára.
[\(<https://github.com/DuranyikCsaba/CMS-vizsgaremelek>\)](https://github.com/DuranyikCsaba/CMS-vizsgaremelek)
 2. A repository-t klónozzuk, vagy .zip állományként letöltsük.
 3. A webalkalmazás mappáját nyissuk meg.
 4. A futtatáshoz szükséges még telepíteni a XAMPP-ot lokális adatbázis hozzáféréshez, a NodeJs-t, illetve az Angular cli 16-os verzióját.
[\(<https://www.apachefriends.org/hu/download.html>\)](https://www.apachefriends.org/hu/download.html)
[\(<https://nodejs.org/en/download>\)](https://nodejs.org/en/download)
[\(<https://v16.angular.io/guide/setup-local>\)](https://v16.angular.io/guide/setup-local)
 5. Lokális kiszolgálón importáljuk a mellékelt adatbázist.
 6. Terminálban navigálunk a backend/ mappába majd futtassuk az “npm run start” parancsot.
 7. Terminálban navigálunk a frontend/cms/ mappába majd futtassuk az “ng serve -o” parancsot.
- Első indítás előtt szükséges node_modules telepítése minden komponensen.



2. Backend komponens dokumentációja:

a. Fejlesztéshez használt technológiák, szoftverek, fejlesztői környezetek:

Fejlesztői eszköz: Visual Studio Code 1.98

Adatbázis: MariaDB 3.4.0 (XAMPP-on futtatva)

ORM: Sequelize 6.37.5

Backend keretrendszer: Express.js 4.21.1

Autentikáció: JSON Web Token (JWT) 9.0.2, bcrypt 5.1.1

Környezeti változók kezelése: dotenv 16.4.7

Frontend kapcsolat: CORS engedélyezése 2.8.5

b. Adatbázis felépítése

Az oldalunk MariaDB 3.4.0 adatbázist használ Sequelize ORM-mel kezelve.

1. Felhasznalok (Users)

Oszlop	Típus	Leírás
id	INTEGER	Felhasználó egyedi azonosító (Primary Key)
nev	STRING	Felhasználó neve
jelszo	STRING	Felhasználó jelszava
email	STRING	Felhasználó email címe (unique)
tel	STRING	Felhasználó telefonos elérhetősége
tipus	INTEGER	Felhasználó típusa (pl. admin vagy sima)
createdAt	TIMESTAMP	Felhasználó regisztrációjának ideje
updatedAt	TIMESTAMP	Felhasználó adatainak módosítási ideje



2. Poszt (Post)

Oszlop	Típus	Leírás
id	INTEGER	Poszt egyedi azonosító (Primary Key)
felhasznaloId	INTEGER	Hozzá tartozó felhasználó ID-ja (Foreign Key)
felhasznaloNeve	TEXT	Felhasználó neve, aki létrehozta a posztot
tartalom	TEXT	Poszt szövege
letrehozas	TIMESTAMP	Poszt létrehozásának ideje
modositas	TIMESTAMP	Poszt utolsó módosításának ideje

3. Komment (Comment)

Oszlop	Típus	Leírás
id	INTEGER	Komment egyedi azonosító (Primary Key)
posztId	INTEGER	Hozzá tartozó poszt ID-ja (Foreign Key)
felhasznaloId	INTEGER	Hozzá tartozó felhasználó ID-ja (Foreign Key)
felhasznaloNeve	TEXT	Felhasználó neve, aki írta a commentet
kommentTartalom	TEXT	A komment tartalma
letrehozas	TIMESTAMP	Komment létrehozásának ideje
modositas	TIMESTAMP	Komment utolsó módosításának ideje



4. Kep (Image)

Oszlop	Típus	Leírás
id	INTEGER	Kép egyedi azonosító (Primary Key)
hirdetes_id	INTEGER	Hirdetés ID-ja (Foreign Key)
file_path	STRING	A kép fájl elérési útja
createdAt	TIMESTAMP	Kép feltöltésének ideje
updatedAt	TIMESTAMP	Kép utolsó módosításának ideje



5. Hirdetesek (Advertisements)

Oszlop	Típus	Leírás
id	INTEGER	Hirdetés egyedi azonosító (Primary Key)
modell	STRING	Jármű modellje
marka	STRING	Jármű márka
ajtok_szama	INTEGER	Jármű ajtóinak száma
hengerurtartalom	FLOAT	Jármű hengerűrtartalma
uzemanyag	STRING	Üzemanyag típusa
evjarat	INTEGER	Jármű évjárata
futott_kilometer	INTEGER	Jármű futott kilométerei
szin	STRING	Jármű színe
sebessegvalto_tipus	STRING	Sebességváltó típusa
kiegeszitok	STRING	Kiegészítők
muszaki_vizsga_ervenyes	DATE	Műszaki vizsga érvényessége
baleseti_elozmenyek	STRING	Baleseti előzmények
felhasznalo_id	INTEGER	Felhasználó ID-ja (Foreign Key)
adatok	STRING	Egyéb adatok
ar	INTEGER	Jármű ára
ert_telszam	STRING	Elérhetőség telefonszám
createdAt	TIMESTAMP	Hirdetés létrehozásának ideje
updatedAt	TIMESTAMP	Hirdetés utolsó módosításának ideje



6. associations (kapcsolótábla)

Oszlop	Típus	Leírás
felhasznalo_id	INTEGER	Felhasználó egyedi azonosító (Foreign Key)
hirdetes_id	INTEGER	Hirdetés egyedi azonosító (Foreign Key)
createdAt	TIMESTAMP	Létrehozás időpontja
updatedAt	TIMESTAMP	Módosítás időpontja



API Végpontok Dokumentációja

Az API végpontok Express.js alapon lettek megvalósítva, JWT alapú autentikációval.

A végpontokat minden esetben:

<http://localhost:5000/api/> - URL-el kezdjük

cms_vizsga forum_bejegyzes	cms_vizsga felhasznalok	cms_vizsga hirdetesek	cms_vizsga markak	cms_vizsga modellek
<ul style="list-style-type: none"># id : int(5)# topic : tinyint(1)# felhasznalo_id : int(5)@ tartalom : text@ letrehozas : datetime@ modositas : datetime# megtekintese : int(11)# torles : tinyint(1)	<ul style="list-style-type: none">v cms_vizsga felhasznalok# id : int(11)@ nev : varchar(255)@ jelszo : varchar(255)@ email : varchar(255)@ tel : varchar(255)# tipus : int(11)@ createdAt : datetime@ updatedAt : datetime	<ul style="list-style-type: none">v cms_vizsga hirdetesek# id : int(5)# felhasznalo_id : int(5)@ letrehozas : datetime@ modositas : datetime# megtekintese : int(11)# torles : tinyint(1)@ adatok : text@ modell : text@ marka : text# ajtolt_szama : int(11)# hengerurtartalom : float@ uzemanyag : text# evjarat : int(11)@ kep1 : text@ kep2 : text@ kep3 : text@ kep4 : text@ kep5 : text@ kep6 : text@ kep7 : text@ kep8 : text@ kep9 : text@ kep10 : text	<ul style="list-style-type: none">v cms_vizsga markak# id : int(5)@ nev : varchar(50)	<ul style="list-style-type: none">v cms_vizsga modellek# id : int(5)@ nev : varchar(50)# marka_id : int(5)

Aktiválja a Windowszt

Aktiválja a Windows rendszert a Gépházba



c. Végpontok dokumentációja

Felhasználók kezelése

POST /register - Felhasználó regisztrációja

POST /login - Felhasználó bejelentkezése

POST /logout - Felhasználó kijelentkezése

GET /user - Bejelentkezett felhasználó adatai

POST /user - Felhasználó adatainak frissítése

POST /user/password - Jelszó módosítása

DELETE /user - Felhasználó törlése

```
router.post("/register", authControllerJs.registerUser);

router.post("/login", authControllerJs.loginUser);

router.post('/logout', authControllerJs.logoutUser);

router.get('/user', authenticateToken, authControllerJs.getUser);
router.post('/user', authenticateToken,
authControllerJs.updateUserData);
router.post('/user/password', authenticateToken,
authControllerJs.updatePassword);
router.delete('/user', authenticateToken, authControllerJs.deleteUser);
router.delete('/user/:id', authenticateToken,
authControllerJs.adminDeleteUser);
router.get('/moderators', authControllerJs.getModerators);
router.post('/moderatorP/:id', authenticateToken,
authControllerJs.moderatorPromote);
router.post('/moderatorD/:id', authenticateToken,
authControllerJs.moderatorDemote);
```



Hirdetések kezelése

GET / - Összes hirdetés lekérése

GET /:id - Egy adott hirdetés lekérése

POST / - Új hirdetés létrehozása (képek feltöltésével)

PUT /:id - Hirdetés módosítása

DELETE /:id - Hirdetés törlése

```
const router = express.Router();
const storage = multer.diskStorage({
  destination: 'uploads/',
  filename: (req, file, cb) => {
    cb(null, `${Date.now()}-${file.originalname}`);
  }
});

const upload = multer({ storage: storage });

router.get("/", getAllHirdetesek);
router.get("/:id", getHirdetesById);
router.post("/", authenticateToken, upload.array('kepek', 10),
createHirdetes);
router.put("/:id", authenticateToken, updateHirdetes);
router.delete("/:id", authenticateToken, deleteHirdetes);
```



Fórumbejegyzések kezelése

POST / - Új bejegyzés létrehozása
GET / - Összes bejegyzés lekérése
GET /:id - Egy adott bejegyzés lekérése
DELETE /:id - Bejegyzés törlése
PATCH /:id - Bejegyzés módosítása

```
const PosztRouter = Router();

PosztRouter.post("/", authenticateToken, posztController.PosztPost);
PosztRouter.get("/", posztController.PosztGet);
PosztRouter.delete("/:id", authenticateToken,
posztController.PosztIdDelete);
PosztRouter.patch("/:id", authenticateToken,
posztController.PosztIdPatch);
PosztRouter.get("/:id", posztController.PosztIdGet);
```



Fórum kommentek kezelése

POST / - Új comment hozzáadása

GET /:posztId - Bejegyzéshez tartozó commentek lekérése

DELETE /:id - Komment törlése

PATCH /:id - Komment módosítása

```
KommentRouter.post("/", authenticateToken,
kommentController.KommentPost);

KommentRouter.get("/:posztId", kommentController.KommentGet);

KommentRouter.delete("/:id", authenticateToken,
kommentController.KommentIdDelete);

KommentRouter.patch("/:id", authenticateToken,
kommentController.KommentIdPatch);
```

d. Funkciók működésének, megvalósításának magyarázata

Adatbázis kapcsolat

A kódrészletben az adatbázishoz való csatlakozás látható sequelize leírása szerint.

```
import {Sequelize} from 'sequelize';

const sequelize = new Sequelize('cms_vizsga', 'admin', 'pwd', {
  host: 'localhost',
  dialect:'mariadb',
  logging: console.log,
});

export default sequelize;
```



authController

Ez a kódrészlet a regisztráció Controller-é ami regisztrációt valósít meg egy Node.js backendben, amely Sequelize-t használ az adatbázis kezeléséhez.
Ellenőrzi név alapján a felhasználót, ha van ilyen névű felhasználó hibaüzenet dob vissza és nem tud végrehajtani a regisztrációt

```
// Regisztráció
export const registerUser = async (req, res) => {
    const { nev, jelszo, email, tel } = req.body;

    try {
        const userExists = await Felhasznalok.findOne({ where: { nev } });
        if (userExists) {
            return res.status(400).json({ message: 'A felhasználó ezzel a felhasználónévvel már létezik.' });
        }

        const hashedPassword = await bcrypt.hash(jelszo, 10);

        const newUser = await Felhasznalok.create({
            nev,
            jelszo: hashedPassword,
            email,
            tel,
            tipus: 1,
        });

        res.status(201).json({ message: 'A felhasználó sikeresen regisztrálva lett!', user: newUser });
    } catch (error) {
        console.error('Regisztrációs hiba:', error);
        res.status(500).json({ message: 'Hiba történt a regisztráció során.' });
    }
};
```



Ez a bejelentkezésnek a Controller-e itt a frontend küldi a bejelentkezési adatokat, amelyet a backend megkap majd az adatbázis megkeresi név alapján itt titkosított formában van eltárolva és a bcrypt segítségével összehasonlíta és ha sikeres az egyezés akkor folytatódhat a bejelentkezés és kap egy JWT(JSON Web Token) Token-t ami egy óráig él.
Ha minden sikeres akkor ezt megkapja a frontend és 200-as kóddal lefut.

```
// Bejelentkezés
export const loginUser = async (req, res) => {
    const { nev, jelszo } = req.body;

    try {
        const user = await Felhasznalok.findOne({ where: { nev } });
        if (!user) {
            return res.status(400).json({ message: 'Helytelen felhasználónév vagy jelszó.' });
        }

        const isPasswordValid = await bcrypt.compare(jelszo, user.jelszo);
        if (!isPasswordValid) {
            return res.status(400).json({ message: 'Helytelen felhasználónév vagy jelszó.' });
        }

        const token = jwt.sign({ id: user.id, nev: user.nev, email: user.email, tipus: user.tipus }, 'titkoskulcs123', { expiresIn: '1h' });

        res.status(200).json({ token });
    } catch (error) {
        console.error('Bejelentkezési hiba:', error);
        res.status(500).json({ message: 'Hiba történt a bejelentkezés során.' });
    }
};
```



Ez a végpont csak egy üzenetet küld vissza, hogy a kilépés sikeres volt. Nincs tényleges művelet, mert a JWT-alapú autentikációnál a frontend kezeli a bejelentkezési állapotot.

```
// Kilépés
export const logoutUser = async (req, res) => {
  try {
    res.status(200).json({ message: 'Kilépés sikeres' });
  } catch (error) {
    console.error('Kilépési hiba:', error);
    res.status(500).json({ message: 'Belső szerver hiba' });
  }
};
```



Itt látható 3 fajta lekérdezés: átlag felhasználó, moderátor

Ez a kódrészlet kettő felhasználó típusról mutat be lekérdezést a legfőbb különbség felhasználó és más jogkörrel szereplő felhasználó között a

Felhasznalok.findAll({ where: { tipus: '!?' } }); a kérdőjeles részen szerepel egy szám amely meghatározza a felhasználó típusát 1-átlag felhasználó, 2- moderátor

```
// Felhasználó lekérdezése
export const getUser = async (req, res) => {
  const id = req.user.id;
  try {
    const user = await Felhasznalok.findByPk(id);
    if (user) {
      res.status(200).json({
        error: false,
        message: "A felhasználó lekérdezése sikeres",
        user
      });
    } else {
      res.status(404).json({
        error: true,
        message: "Nincs ilyen felhasználó",
      });
    }
  } catch (err) {
    console.error("A felhasználó lekérdezése sikertelen", err);
    res.status(500).json({
      error: true,
      message: "A felhasználó lekérdezése során adatbázishiba történt"
    });
  }
};

// minden átlagos felhasználó lekérdezése

export const getAllUsers = async (req, res) => {
  try {
    const users = await Felhasznalok.findAll({
      where: {
        tipus: 1 // Csak a nem admin felhasználók lekérése
      }
    });
    res.status(200).json({
  
```



```

        error: false,
        message: "Felhasználók lekérdezése sikeres",
        users
    });
} catch (error) {
    console.error("Hiba történt a felhasználók lekérdezésékor:", error);
    res.status(500).json({
        error: true,
        message: "Hiba történt a felhasználók lekérdezése során."
    });
}
};

// Moderátorok lekérdezése

export const getModerators = async (req, res) => {
try {
    const users = await Felhasznalok.findAll({
        where: {
            tipus: 2
        }
    });
    res.status(200).json({
        error: false,
        message: "Felhasználók lekérdezése sikeres",
        users
    });
} catch (error) {
    console.error("Hiba történt a felhasználók lekérdezésékor:", error);
    res.status(500).json({
        error: true,
        message: "Hiba történt a felhasználók lekérdezése során."
    });
}
};

```



Ez a függvény lehetővé teszi a bejelentkezett felhasználó számára a jelszava módosítását. Először ellenőrzi, hogy a megadott jelenlegi jelszó helyes-e, majd megnézi, hogy az új jelszó és annak megerősítése megegyezik-e. Ha minden rendben van, az új jelszót bcrypt segítségével titkosítja, és elmenti az adatbázisba. Végül egy sikeres válasszal jelzi, hogy a jelszó frissítése megtörtént.

```
// Jelszó módosítása
export const updatePassword = async (req, res) => {
    const id = req.user.id; // A felhasználó azonosítója a JWT-ból
    const { jelszo, ujjelszo, ujjelszoMegint } = req.body; // Jelszó
mezők

    try {
        const user = await Felhasznalok.findByPk(id);
        if (!user) {
            return res.status(404).json({ message: 'Nincs ilyen felhasználó.' });
        }
    }

    // Jelenlegi jelszó ellenőrzése
    const isPasswordValid = await bcrypt.compare(jelszo, user.jelszo);
    if (!isPasswordValid) {
        return res.status(400).json({ message: 'Helytelen előző jelszó.' });
    }

    // Új jelszavak ellenőrzése
    if (ujjelszo && ujjelszo !== ujjelszoMegint) {
        return res.status(400).json({ message: 'Az új jelszavak nem
egyeznek.' });
    }

    // Jelszó frissítése
    user.jelszo = await bcrypt.hash(ujjelszo, 10); // Új jelszó
hash-elése
    await user.save(); // Változások mentése

    res.status(200).json({ message: 'A jelszó sikeresen frissítve
lett!' });
} catch (error) {
    console.error('Jelszó módosítása hiba:', error);
    res.status(500).json({ message: 'Hiba történt a jelszó módosítása
során.' });
}
```



Ez a függvény lehetővé teszi a bejelentkezett felhasználó számára a saját fiókjának törlését. Először ellenőrzi, hogy a megadott jelszó helyes-e, majd azt is, hogy a jelszó és annak megerősítése megegyezik-e. Ha minden feltétel teljesül, törli a felhasználót az adatbázisból.

```
// Felhasználó törlése
export const deleteUser = async (req, res) => {
    const id = req.user.id; // A felhasználó azonosítója a JWT-ból
    const { jelszo, jelszoMegint } = req.body; // Jelszó és megerősítés

    try {
        const user = await Felhasznalok.findByPk(id);
        if (!user) {
            return res.status(404).json({ message: 'Nincs ilyen felhasználó.' });
        }

        // Jelszó ellenőrzése
        const isPasswordValid = await bcrypt.compare(jelszo, user.jelszo);
        if (!isPasswordValid) {
            return res.status(400).json({ message: 'Helytelen jelszó.' });
        }

        // Megerősítés ellenőrzése
        if (jelszo !== jelszoMegint) {
            return res.status(400).json({ message: 'A jelszavak nem egyeznek.' });
        }

        await Felhasznalok.destroy({ where: { id } }); // Felhasználó törlése

        res.status(200).json({ message: 'A felhasználó sikeresen törölve lett.' });
    } catch (error) {
        console.error('Felhasználó törlése hiba:', error);
        res.status(500).json({ message: 'Hiba történt a felhasználó törlése során.' });
    }
};
```



Ebben a kódrészletben a felhasználói fiókok jogainak változtatása látható, a 0-ás az admin és Ő adhat moderátor jogot a sima felhasználónak 2-est és a sima felhasználók pedig az 1-est használják

```
// Felhasználó típusának módosítása 2-re
user.tipus = 2;
await user.save(); // Mentjük a módosított felhasználót

res.status(200).json({ message: 'A felhasználó típusa sikeresen
módosítva lett.' });
} catch (error) {
  console.error('Felhasználó típusa módosítása hiba:', error);
  res.status(500).json({ message: 'Hiba történt a felhasználó típusa
módosítása során.' });
}

export const moderatorDemote = async (req, res) => {
  const { id } = req.params; // A felhasználó azonosítója

  try {
    // Ellenőrizzük, hogy a kérés indítója admin-e
    if (req.user.tipus !== 0) {
      return res.status(403).json({ message: 'Nincs jogosultságod a
felhasználó típusának módosításához.' });
    }

    const user = await Felhasznalok.findByPk(id);
    if (!user) {
      return res.status(404).json({ message: 'Nincs ilyen felhasználó.' });
    }
  }

  // Felhasználó típusának módosítása 1-re
  user.tipus = 1;
  await user.save(); // Mentjük a módosított felhasználót

  res.status(200).json({ message: 'A felhasználó típusa sikeresen
módosítva lett.' });
} catch (error) {
  console.error('Felhasználó típusa módosítása hiba:', error);
  res.status(500).json({ message: 'Hiba történt a felhasználó típusa
módosítása során.' });
}
```



Forum Controller

Ez a kód egy új poszt létrehozását kezeli. Ellenörzi, hogy a poszt tartalma nem üres-e, majd létrehoz egy új poszt objektumot a beérkező adatokkal (tartalom, felhasználó azonosítója és neve). Az adatbázisba történő mentés után sikeres válasz üzenetet küld.

```
export default {

  PosztPost: (req, res) => {

    if (!req.body.tartalom) {
      return res.status(400).json({
        error: true,
        message: "A poszt tartalma üres"
      });
    }

    const ujPoszt = PosztModel.build();
    ujPoszt.tartalom = req.body.tartalom;
    ujPoszt.felhasznaloId = req.user.id;
    ujPoszt.felhasznaloNeve = req.user.nev;

    ujPoszt.save()
      .then(() => {
        res.status(201).json({
          error: false,
          message: "A poszt létrehozása sikeres!",
          data: ujPoszt
        })
      })
      .catch((err) => {
        console.error("Hiba történt!");
        console.error(err);
        return res.status(500).json({
          error: true,
          message: "A poszt létrehozása sikertelen! Adatbázis hiba!"
        });
      });
  },
}
```



HirdetesekController

A fenti kód egy hirdetéskezelő API-t tartalmaz, amely négy fő funkciót valósít meg: hirdetések lekérése, létrehozás, frissítés és törlés. A getAllHirdetesek és getHirdetesById metódusok lekérdezik a hirdetéseket, beleértve a kapcsolódó képeket, és azok elérési útját a base URL hozzáadásával módosítják. A createHirdetes metódus új hirdetést hoz létre, és a feltöltött képeket is elmenti. Az updateHirdetes lehetővé teszi a meglévő hirdetés adatainak frissítését, megőrzve azokat az adatokat, amelyek nem változnak. A deleteHirdetes metódus lehetővé teszi egy hirdetés törlését, de csak akkor, ha a felhasználónak van megfelelő jogosultsága

```
export const getAllHirdetesek = async (req, res) => {

  try {

    const hirdetesek = await Hirdetesek.findAll({
      include: [{ model: Kep, as: 'kepek' }]
    });

    const baseUrl = "http://localhost:5000/";

    const modifiedHirdetesek = hirdetesek.map(hirdetes => ({
      ...hirdetes.toJSON(),
      kepek: hirdetes.kepek.map(kep => ({
        ...kep.toJSON(),
        file_path: baseUrl + kep.file_path
      }))
    }));
  }

  res.status(200).json({ hirdetesek: modifiedHirdetesek });
} catch (error) {
  console.error('Hiba a hirdetések lekérése során:', error);
  res.status(500).json({ message: 'Hiba a hirdetések lekérése során.' });
}
```



```

        }

    } ;



export const getHirdetesById = async (req, res) => {
    const { id } = req.params;
    try {
        const hirdetes = await Hirdetesek.findById(id, {
            include: [{ model: Kep, as: 'kepek' }]
        });
        if (!hirdetes) {
            return res.status(404).json({ message: 'A hirdetés nem található.' });
        }
        const baseUrl = "http://localhost:5000/";
        hirdetes.kepek = hirdetes.kepek.map(kep => ({
            ...kep.toJSON(),
            file_path: baseUrl + kep.file_path
        }));
        res.status(200).json(hirdetes);
    } catch (error) {
        console.error('Hiba a hirdetés lekérdezése során:', error);
        res.status(500).json({ message: 'Hiba a hirdetés lekérdezése során.' });
    }
}

export const createHirdetes = async (req, res) => {

```



```

try {

    const { adatok, modell, marka, ajtok_szama, hengerurtartalom,
uzemanyag, evjarat, futott_kilometer, szin, sebessegvalto_tipus,
kiegeszitok, muszaki_vizsga_ervenyes, baleseti_elozmenyek, ar,
ert_telszam} = req.body;

    const felhasznalo_id = req.user.id;

    const kepek = req.files;

    if (!modell || !marka || !ajtok_szama || !hengerurtartalom ||
!uzemanyag || !evjarat || !felhasznalo_id || !ar || !ert_telszam) {

        return res.status(400).json({
            error: true,
            message: "Minden kötelező mezőt ki kell tölteni!",
        });
    }

    const newHirdetes = await Hirdetesek.create({
        adatok,
        modell,
        marka,
        ajtok_szama,
        hengerurtartalom,
        uzemanyag,
        evjarat,
        futott_kilometer,
        szin,
        sebessegvalto_tipus,
        kiegeszitok,
        muszaki_vizsga_ervenyes,
    })
}

```



```

    baleseti_elozmenyek,
    felhasznalo_id,
    ar,
    ert_telszam
} );

```

KommentController

Ez a kódrészlet a kommentek létrehozásáért felelős API végpontot valósít meg. Az első lépés, hogy ellenőrzi, van-e tartalom a kommentben. Ha nincs, hibát ad vissza. Ezután ellenőrzi, hogy a megadott poszt létezik-e az adatbázisban. Ha a poszt nem található, szintén hibát jelez. Ha a poszt létezik, akkor egy új kommentet hoz létre, amelyet a KommentModel segítségével épít fel, hozzárendelve a felhasználó adatait, a poszt ID-ját és a komment tartalmát. A kommentet végül elmenthetik az adatbázisba.

```

import KommentModel from "../models/Komment.js";
import PosztModel from "../models/Poszt.js";

export default {

  KommentPost: async (req, res) => {

    const { posztId, kommentTartalom } = req.body;

    if (!kommentTartalom) {
      return res.status(400).json({
        error: true,
        message: "A komment tartalma üres"
      });
    }

    const poszt = await PosztModel.findById(posztId);
    if (!poszt) {
      return res.status(404).json({
        error: true,
        message: "A poszt nem található"
      });
    }

    const newKomment = new KommentModel({
      posztId: poszt._id,
      tartalom: kommentTartalom,
      felhasznalo_id: req.user._id,
      ar: 0,
      ert_telszam: 0
    });

    await newKomment.save();
    res.json(newKomment);
  }
}

```



```
    message: "A megadott poszt nem található!"  
  } );  
  
}  
  
const ujKomment = KommentModel.build({  
  posztId,  
  felhasznaloId: req.user.id,  
  felhasznaloNev: req.user.nev,  
  kommentTartalom  
});
```



authController

ez a kód ellenőrzi, hogy a kérés tartalmaz-e érvényes JWT tokent. Ha nincs token vagy érvénytelen, 403-as hibaüzenetet küld, egyébként hozzáadja a felhasználó adatokat a kéréshez és továbbengedi a következő lépéshez.

```
import jwt from 'jsonwebtoken';

export const authenticateToken = (req, res, next) => {

  const authHeader = req.headers['authorization'];

  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {

    return res.status(403).json({ message: 'Token szükséges a hozzáféréshez.' });

  }

  jwt.verify(token, 'titkoskulcs123', (err, user) => {

    if (err) {

      return res.status(403).json({ message: 'Érvénytelen vagy lejárt token!' });

    }

    req.user = user;

    next();

  });

};
```



3. Frontend komponens dokumentációja:

a. Elkészített komponensek, és működésük:

A frontenden található statikus oldalak komponensei (fooldal, gyakori-autok, gyik-adasveteli, gyik-atiratas, gyik-betetlap, gyik-casco, gyik-kotelezo, rolunk, footer) csupán adatot jelenítenek meg, nem rendelkeznek önálló funkciókkal.

app komponens:

Ebben a komponensben csupán az oldal keretét képző komponensek mint a nav és a footer találhatóak, a router-outlet ami megjeleníti az aktuálisan meghívott komponenseket illetve az “oldal tetejére” gomb melyek minden oldalon megjelennek.

A komponens ts állományában az “oldal tetejére” funkció foglal helyet ami HostListener-rel működik és a scrollToTop() eljárás esetén a window.scrollTo() parancsot hajtja végre.

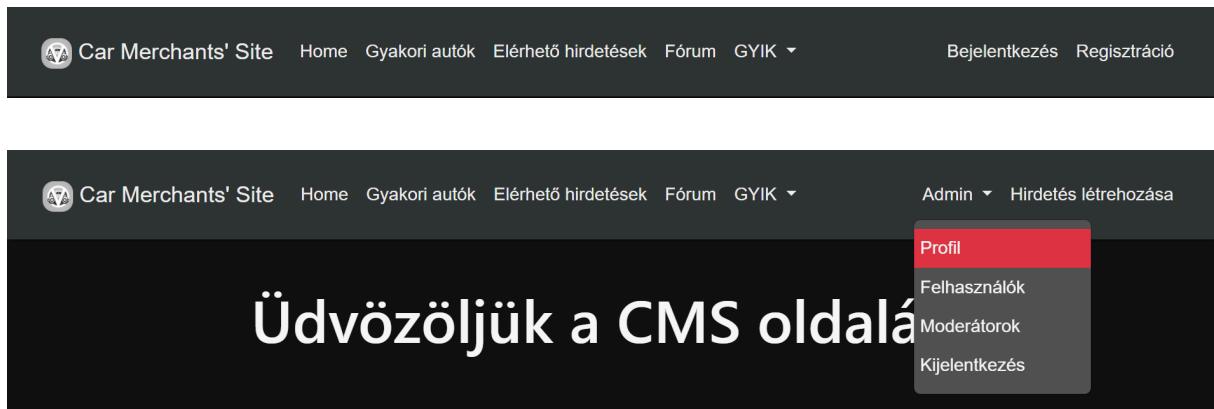
```
@HostListener('window:scroll', [])
onWindowScroll() {
    this.showButton = window.scrollY > 350;
}

scrollToTop() {
    window.scrollTo({ top: 0, behavior: 'smooth' });
}
```



nav komponens:

A nav komponens az oldalon végbemenő navigációt bonyolítja le. Az első képen a vendég felhasználó szemszögét szemlélteti, míg a második a bejelentkezett Admin felhasználót mutatja be. Az általános és moderátor felhasználók csupán a Profil és Kijelentkezés opciók érhetőek el.



A NavComponent az OnInit és OnDestroy lehetőségeket implementálja ezzel lehetővé téve a komponens inicializálását és az erőforrások felaszabadítását. A komponens az AuthService segítségével figyeli a felhasználó bejelentkezési állapotát majd a bejelentkezett felhasználó tipusának azonosításával a megfelelő formátumot tölti be. Megkülönböztetünk vendég, regisztrált felhasználó és admin felhasználókat a navigációs sáv betöltésekor. A loadUserData metódus gondoskodik arról, hogy a felhasználó neve jelenjen meg a navigációs sávon. A komponens a Subscription osztályt használja a megfigyelők kezelésére.



```

export class NavComponent implements OnInit, OnDestroy {
  isLoggedIn = false;
  userName: string | null = null;
  userType: number | null = null;
  private subscription: Subscription = new Subscription();

  constructor(private authService: AuthService, private userService: UserService) {}

  ngOnInit(): void {
    this.subscription.add(
      this.authService.isLoggedIn().subscribe(
        (loggedIn: boolean) => {
          this.isLoggedIn = loggedIn;
          if (loggedIn) {
            this.loadUserData();
          } else {
            this.userName = null;
            this.userType = null;
          }
        }
      )
    );
  }

  this.subscription.add(
    this.userService.userUpdated$.subscribe(() => {
      this.loadUserData();
    })
  );
}

ngOnDestroy(): void {
  this.subscription.unsubscribe();
}

loadUserData(): void {
  const user = this.authService.getCurrentUser();
  this.userName = user ? user.nev : null;
  this.userType = user ? user.tipus : null;
}

```



```
logout(): void {
  this.authService.logout();
}
```



login komponens:

Ez a komponens a felhasználó bejelentkeztetésére szolgál amelyet felhasználónév és jelszó párossal végzünk.

Az onSubmit metódus először a hibaüzeneteket törli majd futtatja a validateForm metódust amely az űrlapba bevitt adatok hitelességét ellenőrzi.

A validateForm metódus azt ellenőrzi, hogy a felhasználónév és a jelszó meg van-e adva illetve megfelelnek-e azok a kritériumoknak. Amennyiben az űrlap adatai érvényesek, elküldi azokat a backend szerver számára a login metódussal egy HTTP POST kérés formájában.

Amennyiben a bejelentkezés sikeresen lezajlott, a kapott token az AuthService-ben kerül tárolásra, a felhasználó átirányítódik a főoldalra. Ellenkező esetben hibaüzenet jelenik meg.

Ha a felhasználónak nincs fiókja, egy kattintással a regisztrációs felületre kerül ahol elvégezheti a szükséges műveleteket. Elfelejtett jelszó esetén az Adminisztrátorral veheti fel a kapcsolatot.



```

export class LoginComponent {
    username: string = '';
    password: string = '';
    successMessage: string = '';
    serverErrorMessage: string = '';
    usernameError: string = '';
    passwordError: string = '';

    constructor(
        private http: HttpClient,
        private router: Router,
        private authService: AuthService
    ) {}

    onSubmit() {
        this.serverErrorMessage = ''; // Korábbi hibaüzenet törlése
        if (this.validateForm()) {
            this.login();
        }
    }

    validateForm(): boolean {
        this.usernameError = '';
        this.passwordError = '';

        // Üres felhasználónév ellenőrzése
        if (!this.username.trim()) {
            this.usernameError = 'Felhasználónév megadása kötelező!';
        } else if (!/^[a-zA-Z0-9áéíóöűÁÉÍÓÖŰ]+$/ .test(this.username)) {
            this.usernameError = 'A felhasználónév csak betűket, számokat és ékezetes karaktereket tartalmazhat!';
        }

        // Üres vagy túl rövid jelszó ellenőrzése
        if (!this.password.trim()) {
            this.passwordError = 'Jelszó megadása kötelező!';
        } else if (this.password.length < 8) {
            this.passwordError = 'A jelszónak legalább 8 karakter hosszúnak kell lennie!';
        }

        return !this.usernameError && !this.passwordError;
    }
}

```



```

}

login() {
  const payload = { nev: this.username, jelszo: this.password };
  console.log('Login payload:', payload);

  this.http.post('http://localhost:5000/api/auth/login', payload, {
    headers: { 'Content-Type': 'application/json' }
  }).subscribe(
    (response: any) => {
      this.successMessage = 'Sikeres bejelentkezés!';
      this.authService.login(response.token);
      this.router.navigate(['/']);
    },
    (error) => {
      console.error('Login error:', error);

      if (error.status === 401) {
        this.serverErrorMessage = 'Hibás felhasználónév vagy
jelszó!';
      } else if (error.status === 500) {
        this.serverErrorMessage = 'Szerverhiba! Kérjük, próbálja újra
később.';
      } else {
        this.serverErrorMessage = 'Hiba történt a bejelentkezés
során.';
      }
    }
  );
}

```



register komponens:

A register komponens egy űrlapot tartalmaz amelynek mezői bizonyos kritériumoknak megfelelő adatokat fogadnak el:

Felhasználónév: 5-15 karakter terjedelem, nem lehet üres.

E-mail: "szöveg"@"szöveg". "szöveg", nem lehet üres.

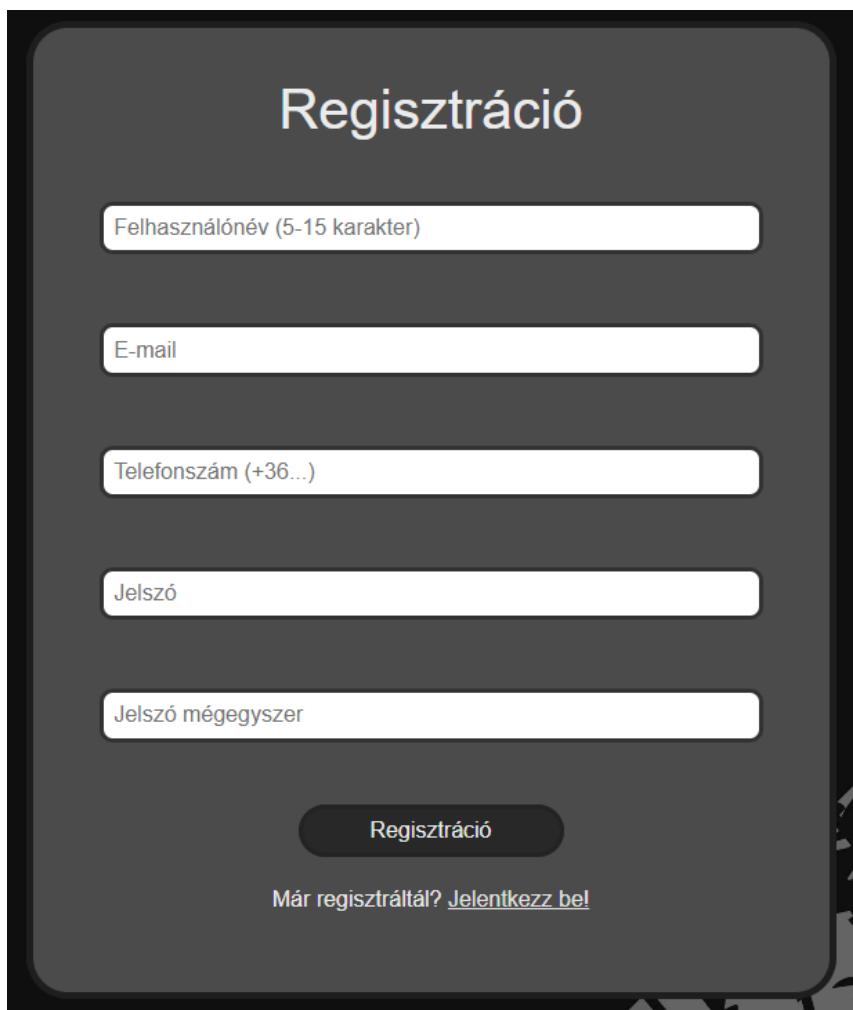
Telefonszám: +36xxxxxxxxx , nem lehet üres, csak magyar nemzetközi előhívóval rendelkező telefonszámok elfogadottak.

Jelszó: minimum 8 karakter, maximum 32 karakter.

Jelszó megegyezés: meg kell egyeznie a Jelszóval.

Hibás adat megadása esetén az űrlap az adott hibás beviteli mezőhöz hibaüzenetet ír.

Regisztráció gomb megnyomásával az onSubmit() metódus fut le amely a validateForm() eljárással hitelesíti az adatokat, ezután egy formData objektumot hoz létre amelyet HTTP POST metódussal továbbít a backend szerver megfelelő regisztrációs végpontjára. Ha sikeres a küldés, a login felületre irányítja a felhasználót. Sikertelen küldés esetén hibaüzenetet jelenít meg.



```

export class RegisterComponent {
  username: string = '';
  email: string = '';
  phone: string = '';
  password: string = '';
  confirmPassword: string = '';

  usernameError: string = '';
  emailError: string = '';
  phoneError: string = '';
  passwordError: string = '';
  confirmPasswordError: string = '';

  successMessage: string = '';
  serverErrorMessage: string = '';

  constructor(private http: HttpClient, private router: Router) {}

  validateForm() {
    let isValid = true;

    // Felhasználónév ellenőrzése
    if (!this.username) {
      this.usernameError = 'A felhasználónév megadása kötelező!';
      isValid = false;
    } else {
      this.usernameError = '';
    }

    // E-mail ellenőrzése
    const emailPattern =
      /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
    if (!this.email) {
      this.emailError = 'Az e-mail megadása kötelező!';
      isValid = false;
    } else if (!emailPattern.test(this.email)) {
      this.emailError = 'Helytelen e-mail formátum!';
      isValid = false;
    } else {
      this.emailError = '';
    }

    // Telefonszám ellenőrzése
  }
}

```



```

const phonePattern = /^+36[0-9]{9}$/;
if (!this.phone) {
    this.phoneError = 'A telefonszám megadása kötelező!';
    isValid = false;
} else if (!phonePattern.test(this.phone)) {
    this.phoneError = 'Helytelen telefonszám formátum! A helyes
formátum: +36nnnnnnnn';
    isValid = false;
} else {
    this.phoneError = '';
}

// Jelszó ellenőrzése
if (!this.password) {
    this.passwordError = 'A jelszó megadása kötelező!';
    isValid = false;
} else if (this.password.length < 8 && this.password.length > 32) {
    this.passwordError = 'Min. karakterhossz: 8 karakter. Max.
karakterhossz: 32 karakter';
    isValid = false;
} else {
    this.passwordError = '';
}

// Jelszó megerősítésének ellenőrzése
if (this.password !== this.confirmPassword) {
    this.confirmPasswordError = 'A jelszavak nem egyeznek!';
    isValid = false;
} else {
    this.confirmPasswordError = '';
}

return isValid;
}

onSubmit() {
    if (this.validateForm()) {
        this.successMessage = ''; // Reset success üzenet
        this.serverErrorMessage = ''; // Reset server hibaüzenet

        console.log('Felhasználónév:', this.username);
        console.log('E-mail:', this.email);
        console.log('Telefonszám:', this.phone);
    }
}

```



```

        console.log('Jelszó:', this.password);
        console.log('Jelszó mégegyszer:', this.confirmPassword);

        // Backendnek küldött adatok objektuma
        const formData = {
            nev: this.username, // A backend `nev` mezőt vár
            email: this.email,
            tel: this.phone, // Telefonszám mező: `tel`
            jelszo: this.password, // Jelszó mező: `jelszo`
        };

        // HTTP POST kérés küldése
        this.http.post('http://localhost:5000/api/auth/register',
        formData)
        .subscribe({
            next: (response) => {
                console.log('Sikeres regisztráció:', response);
                this.successMessage = 'Sikeres regisztráció!';
                this.router.navigate(['/login'])
            },
            error: (error) => {
                console.error('Hiba történt:', error);
                this.serverErrorMessage = 'Hiba történt a regisztráció során! Kérjük próbálja újra.';
            }
        });
    } else {
        console.error('A form kitöltése hibás.');
    }
}

```



create-hirdetes komponens:

Ez egy hirdetés létrehozásáért felelős Angular komponens, amely lehetővé teszi a felhasználók számára, hogy hirdetéseket adjanak fel. Csak regisztrált felhasználók számára érhető el.

FormGroup használatával létrehoz egy űrlapot amely mezőket tartalmaz. A mezők egy használt autó hirdetésnek a legfontosabb adatait rögzítik.

Az onFileSelected metódus a kép fájlok előnézetét hivatott megjeleníteni.

Az onSubmit metódus ellenőrzi a megadott beviteli mezők helyes kitöltöttségét, illetve, hogy a felhasználó jogosult-e a hirdetés feladására token alapján. Amennyiben jogosult, az adatokból létrejön egy FormData objektum ami képekkel együtt a backend szerverre kerül továbbításra.

Sikerességtől esetén a hirdetes-list komponensre továbbít a metódus, ahol fellelhető a létrehozott hirdetés.

Ellenkező esetben hibaüzenetet kapunk.



```

export class CreateHirdetesComponent implements OnInit {
  hirdetesForm: FormGroup;
  isLoggedIn$: Observable<boolean>;
  selectedFiles: File[] = [];
  imageUrl: (string | ArrayBuffer)[] = [];
  currentYear: number;

  constructor(private fb: FormBuilder, private http: HttpClient,
  private authService: AuthService, private router: Router) {
    this.isLoggedIn$ = this.authService.isLoggedIn();
    this.currentYear = new Date().getFullYear(); // Az aktuális év
    inicializálása

    // Úrlap inicializálása
    this.hirdetesForm = this.fb.group({
      adatok: ['', Validators.required],
      modell: ['', Validators.required],
      marka: ['', Validators.required],
      ajtok_szama: ['', [Validators.required, Validators.min(1)]],
      hengerurtartalom: ['', [Validators.required, Validators.min(0)]],
      uzemanyag: ['', Validators.required],
      evjarat: ['', [Validators.required, Validators.min(1900),
      Validators.max(this.currentYear)]],
      futott_kilometer: ['', [Validators.required, Validators.min(0)]],
      szin: ['', Validators.required],
      sebessegvalto_tipus: ['', Validators.required],
      kiegeszitok: ['', Validators.required],
      muszaki_vizsga_ervenyes: [''],
      baleseti_elozmenyek: [''],
      ert_telszam: ['', Validators.required],
      ar: ['', [Validators.required, Validators.min(0)]]
    });
  }

  ngOnInit(): void {}

  // Fájlok kiválasztásának kezelése
  onFileSelected(event: Event): void {
    const input = event.target as HTMLInputElement;
    if (input.files && input.files.length > 0) {
      this.selectedFiles = Array.from(input.files); // Fájlok
      hozzáadása a tömbhöz
    }
  }
}

```



```

this.imageUrls = [];< // Tömb ürítése új fájlokhoz

        // minden kiválasztott fájl beolvasása és előnézetének
lététrehozása
for (let i = 0; i < this.selectedFiles.length; i++) {
    const file = this.selectedFiles[i];
    const reader = new FileReader();

    reader.onload = (e) => {
        this.imageUrls.push(reader.result as string); // URL
hozzáadása a tömbhöz
    };

    reader.readAsDataURL(file); // Fájl olvasása Data URL-ként
}
}

// Űrlap elküldése
onSubmit(): void {
if (this.hirdetesForm.valid && this.selectedFiles.length > 0) {
    // Felhasználó ID lekérése a tokenból
    const currentUser = this.authService.getCurrentUser();
    if (!currentUser) {
        console.error('Felhasználó nincs bejelentkezve!');
        return;
    }

    // FormData létrehozása
    const formData = new FormData();

    // Űrlap adatainak hozzáadása a FormData-hoz
    Object.keys(this.hirdetesForm.controls).forEach(key => {
        formData.append(key, this.hirdetesForm.get(key)?.value);
    });

    // Felhasználó ID hozzáadása
    formData.append('felhasznalo_id', currentUser.id.toString());

    // Képek hozzáadása a FormData-hoz
    this.selectedFiles.forEach((file) => {
        formData.append('kepek', file, file.name);
    });
}
}

```



```

// Feltöltés a szerverre FormData formátumban
this.http.post('http://localhost:5000/hirdetesek', formData, {
  headers: {
    'Authorization': `Bearer ${localStorage.getItem('token')}`
  }
}) .subscribe(
  response => {
    console.log('Hirdetés sikeresen feladva!', response);
    // Úrlap resetelése
    this.hirdetesForm.reset();
    this.selectedFiles = [];
    this.imageUrls = [];
    this.router.navigate(['/hirdetes-list']);

  },
  error => {
    console.error('Hiba a hirdetés feladása során:', error);
    if (error.error) {
      console.error('Szerver válasza:', error.error);
    }
  }
);
} else {
  console.error('Kérém, töltse ki az összes mezőt és válasszon ki legalább egy képet!');
}
}
}

```



hirdetes-list komponens:

A komponens inicializáláskor lekéri az adatbázisban található 2 hétnél nem régebbi hirdetéseket.

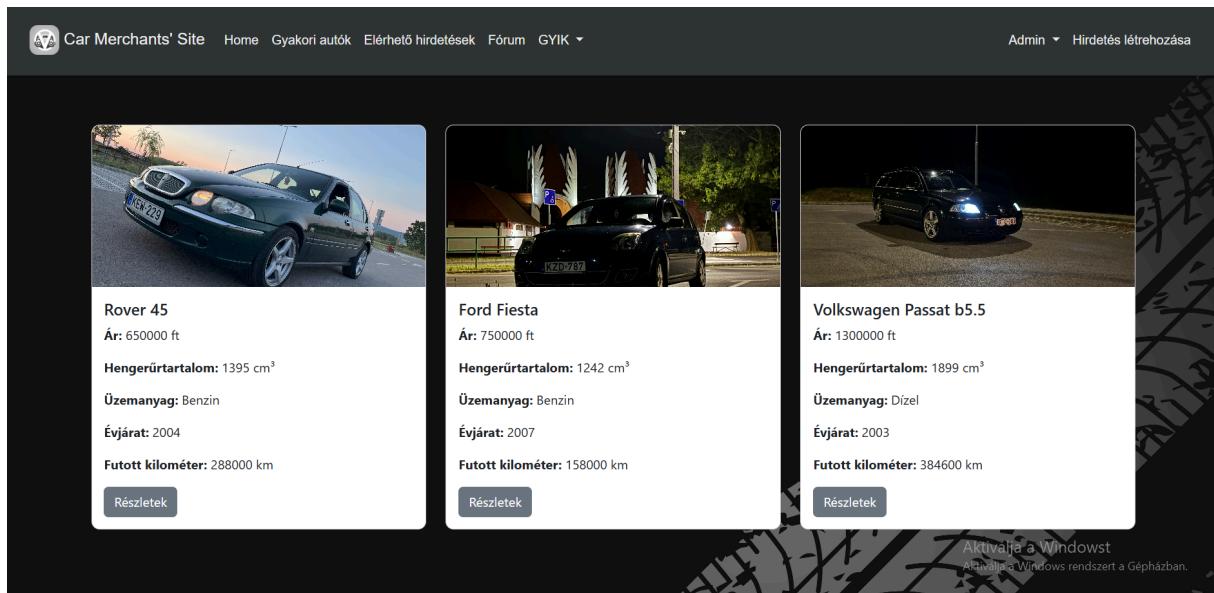
```
ngOnInit(): void {
    this.fetchHirdetesek();
}

fetchHirdetesek(): void {
    this.loading = true;
    this.http.get<{ hirdetesek: Hirdetes[] }>('http://localhost:5000/hirdetesek')
        .subscribe({
            next: (response) => {
                const ketHetElott = new Date();
                ketHetElott.setDate(ketHetElott.getDate() - 14); // 14 nappal
                korábbi dátum

                this.hirdetesek = response.hirdetesek
                    .filter(hirdetes => new Date(hirdetes.createdAt) >=
                ketHetElott) // Szűrés
                    .map(hirdetes => ({
                        ...hirdetes,
                        kepek: hirdetes.kepek ? hirdetes.kepek.map(kep => ({
                            file_path: `${kep.file_path}`
                        })) : []
                    }));
                this.loading = false;
            },
            error: (error) => {
                console.error('Hiba a hirdetések lekérése során:', error);
                this.errorMessage = 'Hiba történt a hirdetések lekérése
                során.';
                this.loading = false;
            }
        });
}
```



Ezek egy kártyán jelennek meg, az első feltöltött képpel és azokkal az adatokkal amelyek relevánsak egy hirdetés rövid ismertetéséhez



A "Részletek" gombra kattintva egy NgbModal nyílik meg ami a többi adatot és képet is listázza, a modal alján szerepel egy "Bezárás" gomb továbbá admin vagy moderátor jogkört betöltő felhasználók illetve a posztot létrehozó felhasználó esetén egy törlés gombot is tartalmaz. A hirdetés törlése egy http kérést indít a backend szerver számára melynek sikeres lefutása esetén a hirdetés törlődik, eltűnik az oldalról, visszajelzést küld. ellenkező esetben error folyamat fut le.

```
deleteHirdetes(id: number): void {
    if (confirm('Biztosan törölni szeretnéd ezt a hirdetést?')) {
        const token = this.authService.getToken(); // Token lekérése az AuthService-ból
        this.http.delete(`http://localhost:5000/hirdetesek/${id}`, {
            headers: {
                'Authorization': `Bearer ${token}` // Token hozzáadása a kéréshez
            }
        })
        .subscribe({
            next: () => {
                this.hirdetesek = this.hirdetesek.filter(hirdetes =>
                    hirdetes.id !== id);
                alert('Hirdetés törlve.');
                this.closeModal();
            },
            error: (error) => {
                console.error('Hiba a hirdetés törlése során:', error);
            }
        });
    }
}
```



```
        alert('Hiba történt a hirdetés törlése során.');
    }
})
}
}

canEditOrDelete(hirdetes: Hirdetes): boolean {
  const user = this.authService.getCurrentUser ();
  return user !== null && (hirdetes.felhasznalo_id === user.id ||
user.tipus === 0 || user.tipus === 2);
}
}
```



profil komponens:

Ezen a komponensen az inicializáció során a getUserData() metódus fut le amely egy HTTP GET kérés során lekéri a felhasználó adatait (név, email, telefonszám).

```
getUserData(): void {
    const token = localStorage.getItem('token');
    const headers = new HttpHeaders().set('Authorization', `Bearer
${token}`);

    this.http.get('http://localhost:5000/api/auth/user', { headers })
        .subscribe({
            next: (response: any) => {
                this.user = response.user;
                this.userDataForm.patchValue({
                    nev: this.user.nev,
                    email: this.user.email,
                    tel: this.user.tel
                });
            },
            error: (error) => {
                console.error('Hiba a felhasználói adatok lekérdezésekor:', error);
            }
        });
}
```

Ezeket illetve a jelszót módosíthatja a felhasználó. Egyszerű adatok módosítása esetén az oldal újból betöltésre kerül és a friss adatok jelennek meg. Jelszó módosítás esetén szükséges megadni a jelenlegi jelszót majd az újat illetve annak megerősítését. Ezek a metódusok a megfelelő végpontokra küldenek HTTP POST kéréseket.

```
updateUserData(): void {
    if (this.userDataForm.invalid) {
        return;
    }

    const token = localStorage.getItem('token');
    const headers = new HttpHeaders().set('Authorization', `Bearer
${token}`);
```



```

        this.http.post(`http://localhost:5000/api/auth/user`,
this.userDataForm.value, { headers })
    .subscribe({
      next: (response) => {
        console.log('Felhasználói adatok sikeresen frissítve:', response);
        this.feedbackMessage = 'A felhasználói adatok sikeresen frissítve!';
        this.userService.notifyUserUpdate();
        setTimeout(() => {
          window.location.reload();
        }, 2000);
      },
      error: (error) => {
        console.error('Hiba a felhasználói adatok frissítése során:', error);
        this.feedbackMessage = 'Hiba történt a felhasználói adatok frissítése során.';
      }
    });
  }

updatePassword(): void {
  if (this.passwordForm.invalid) {
    return;
  }

  const { previousPassword, newPassword, confirmPassword } =
this.passwordForm.value;

  if (newPassword !== confirmPassword) {
    this.feedbackMessage = 'Az új jelszavak nem egyeznek.';
    return;
  }

  const token = localStorage.getItem('token');
  const headers = new HttpHeaders().set('Authorization', `Bearer ${token}`);
}

this.http.post(`http://localhost:5000/api/auth/user/password`, {
  jelszo: previousPassword,
  ujJelszo: newPassword,
  ujJelszoMegint: confirmPassword
}

```



```

}, { headers })
.subscribe({
  next: (response) => {
    console.log('Jelszó sikeresen frissítve:', response);
    this.feedbackMessage = 'A jelszó sikeresen módosítva!';
    this.passwordForm.reset();
    this.showPasswordForm = false;
  },
  error: (error) => {
    console.error('Hiba a jelszó módosítása során:', error);
    this.feedbackMessage = 'Hiba történt a jelszó módosítása
során.';
  }
});
}

```

A felhasználónak még lehetősége van törlni profilját a jelszava kétszeri megadásával. Ebben az esetben a felhasználó kijelentkeztetésre kerül.

Minden HTTP kérésnél siker illetve hibaüzenetek jelennek meg a felhasználó számára

```

deleteUser (): void {
  if (this.deleteForm.invalid) {
    return;
  }

  const { password, confirmPassword } = this.deleteForm.value;

  if (password !== confirmPassword) {
    this.feedbackMessage = 'A jelszavak nem egyeznek.';
    return;
  }

  const token = localStorage.getItem('token');
  const headers = new HttpHeaders().set('Authorization', `Bearer
${token}`);
}

this.http.delete('http://localhost:5000/api/auth/user', {
  headers,
  body: {
    jelszo: password,
    jelszoMegint: confirmPassword
  }
})

```



```
.subscribe({
  next: (response) => {
    console.log('Felhasználó sikeresen törölve:', response);
    this.feedbackMessage = 'A profil sikeresen törölve!';
    this.authService.logout();
  },
  error: (error) => {
    console.error('Hiba a felhasználó törlésekor:', error);
    this.feedbackMessage = 'Hiba történt a profil törlése során.';
  }
});
```



admin komponens:

Az admin komponens csak adminok számára használható funkciókat tartalmaz melyek a felhasználó:

- törlése
- moderátorrá tétele
- adatainak módosítása

```
editUser (user: User): void {
    this.selectedUser = user; // Kiválasztott felhasználó beállítása
        this.userDataForm.patchValue(user); // Űrlap kitöltése a kiválasztott felhasználó adataival
    this.showUserDataForm = true; // Űrlap megjelenítése

    // Gördülés a Űrlaphoz
    setTimeout(() => {
        const element = document.getElementById('userDataForm');
        if (element) {
            element.scrollIntoView({ behavior: 'smooth' });
        }
    }, 0);
}

updateUserData(): void {
    const token = this.authService.getToken(); // Token lekérése az AuthService-ból
    if (this.selectedUser) {
        const updatedUser = { ...this.selectedUser,
...this.userDataForm.value }; // Frissített felhasználói adatok

        this.http.post(`http://localhost:5000/api/auth/aUpdate`, updatedUser, {
            headers: {
                'Authorization': `Bearer ${token}` // Token hozzáadása a kéréshez
            }
        }).subscribe({
            next: (response) => {
                console.log('Felhasználói adatok sikeresen frissítve:', response);
                this.fetchUsers(); // Frissítjük a felhasználók listáját
                this.showUserDataForm = false; // Űrlap elrejtése
                alert('Felhasználói adatok sikeresen frissítve!');
            },
        });
    }
}
```



```

        error: (error) => {
            console.error('Hiba a felhasználói adatok frissítése során:', error);
            alert('Hiba történt a felhasználói adatok frissítése során.');
        }
    );
}
}

deleteUser (id: number): void {
    if (confirm('Biztosan törölni szeretnéd ezt a felhasználót?')) {
        const token = this.authService.getToken(); // Token lekérése az AuthService-ból
        this.http.delete(`http://localhost:5000/api/auth/user/${id}`, {
            headers: {
                'Authorization': `Bearer ${token}` // Token hozzáadása a kéréshez
            }
        }).subscribe({
            next: () => {
                this.users = this.users.filter(user => user.id !== id); // Felhasználó eltávolítása a listából
                alert('Felhasználó törlőltve.');
            },
            error: (error) => {
                console.error('Hiba a felhasználó törlése során:', error);
                alert('Hiba történt a felhasználó törlése során.');
            }
        });
    }
}

promoteModerator(userId: number): void {
    const token = this.authService.getToken(); // Token lekérése az AuthService-ból

    this.http.post<PromoteResponse>(`http://localhost:5000/api/auth/moderator/${userId}`, {}, {
        headers: {
            'Authorization': `Bearer ${token}` // Token hozzáadása a kéréshez
        }
    });
}

```



```

        }) .subscribe({
          next: (response) => {
            console.log('Sikeres promóció:', response.message);
            this.fetchUsers();
          },
          error: (error) => {
            console.error('Hiba a felhasználó promóciója során:', error);
          }
        ) );
      }
    }
}

```

A listázás az egyszerű (1-es típusú) felhasználókat listázza amely a backenden történik kiválasztásra.

A moderátorrá promotált felhasználók a “Moderátorok” aloldalra kerülnek ahol lefokozások lehetséges.

Egy moderátor típusú felhasználó ugyanúgy rendelkezik az admin poszt, komment és hirdetés törlő jogaival, a fennmaradókat nem kapja meg.

Minden funkció rendelkezik egy önálló HTTP kéréssel a megfelelő végpontra. Siker és hibakezeléssel is.

Az adatok módosítása egy panelt használ amelyen az admin felhasználónévet, emailt, telefonszámot és jelszót módosíthat. Módosítás után az adatok frissülnek.

moderátor komponens az admin komponenstől annyiban tér el, hogy kizárolag a promoteModerator() metódus ellenetétjét tatraalmazza. A fetchelés megegyezik.

```

demoteModerator(userId: number): void {
  const token = this.authService.getToken(); // Token lekérése az AuthService-ból

  this.http.post<DemoteResponse>(`http://localhost:5000/api/auth/moderato
rD/${userId}`, {}, {
    headers: {
      'Authorization': `Bearer ${token}` // Token hozzáadása a
    kéréshez
    }
  }) .subscribe({
    next: (response) => {
      console.log('Sikeres demótálás:', response.message);
      this.fetchUsers();
    }
  })
}

```



```
    } ,
    error: (error) => {
      console.error('Hiba a felhasználó demótálása során:', error);
    }
  );
}

}
```



forum komponens:

ngOnInit()

- Leírás: Az Angular életciklus metódusa, amely a komponens inicializálásakor hívódik meg. Itt hívja meg a loadPosts() metódust, hogy betöltsse a fórum posztokat.

loadPosts()

- Leírás: Lekéri a fórum posztokat a ForumService-től. A válasz alapján frissíti a posts tömböt, amely tartalmazza a posztokat és azok kommentjeit. Hiba esetén a konzolra írja a hibaüzenetet.

addPost()

- Leírás: Új posztot hoz létre a felhasználó által megadott tartalom alapján. Ellenőrzi, hogy a tartalom nem üres-e, majd a ForumService-t hívja meg az új poszt létrehozásához. Sikeres létrehozás esetén a posztot hozzáadja a posts tömb elejéhez.

deletePost(postId: number)

- Leírás: Törli a megadott azonosítójú posztot a ForumService-en keresztül. Ha a törlés sikeres, a poszt eltávolításra kerül a posts tömbből.

updatePost(postId: number)

- Leírás: Beállítja a posztot szerkesztésre a megadott azonosító alapján. A poszt tartalmát a newPostContent változóba másolja, hogy a felhasználó szerkeszthesse.

saveUpdatedPost(postId: number)

- Leírás: Menteni próbálja a módosított posztot a ForumService-en keresztül. Ha a frissítés sikeres, a posztot frissíti a posts tömbben, és visszaállítja a szerkesztési állapotot.

addComment(postId: number)

- Leírás: Új kommentet ad hozzá a megadott poszthoz. Ellenőrzi, hogy a komment tartalma nem üres-e, majd a ForumService-t hívja meg az új komment létrehozásához. A sikeres létrehozás után a kommentet hozzáadja a megfelelő poszt kommentjeihez.



`updateComment(commentId: number)`

- Leírás: Beállítja a commentet szerkesztésre a megadott azonosító alapján. A comment tartalmát a newCommentContent változóba másolja, hogy a felhasználó szerkeszthesse.

`saveUpdatedComment(commentId: number)`

- Leírás: Menteni próbálja a módosított commentet a ForumService-en keresztül. Ha a frissítés sikeres, a commentet frissíti a megfelelő poszt commentjei között, és visszaállítja a szerkesztési állapotot.

`deleteComment(commentId: number)`

- Leírás: Törli a megadott azonosítójú commentet a ForumService-en keresztül. Ha a törlés sikeres, a comment eltávolításra kerül a megfelelő poszt commentjei közül.

`toggleCommentInput(postId: number)`

- Leírás: Váltja a comment írásához szükséges űrlap megjelenítését a megadott poszt azonosítója alapján. Ha a poszt azonosítója megegyezik a kiválasztott poszt azonosítójával, akkor elrejti az űrlapot, különben megjeleníti.

`toggleShowAllComments(postId: number)`

- Leírás: Váltja a megadott poszthoz tartozó összes comment megjelenítését. Ha a poszt azonosítója már szerepel a showAllComments objektumban, akkor az értékét megfordítja, így megjeleníti vagy elrejti a commenteket.



b. Routing működése, frontend végpontok felépítése

Angular Routing Modul Dokumentáció

Áttekintés

Ez a dokumentum egy Angular alkalmazás routing konfigurációját mutatja be. A routing modul meghatározza a navigációs útvonalakat és a hozzájuk tartozó komponenseket, lehetővé téve a felhasználók számára, hogy navigáljanak az alkalmazás különböző nézetei között.

Importok

A következő Angular modulok és komponensek vannak importálva:

- **NgModule:** Dekorátor, amely megjelöli a klasszist Angular modulként.
- **RouterModule:** Modul, amely biztosítja a routinghoz szükséges szolgáltatásokat és direktívákat.
- **Routes:** Típus, amely meghatározza az útvonal konfiguráció szerkezetét.
- **Komponensek:** Különböző komponensek, amelyek az alkalmazás különböző nézeteit képviselik.

Komponensek

A routing konfigurációban a következő komponensek találhatók:



- *RegisterComponent*: **Felhasználói regisztrációs komponens.**
- *LoginComponent*: **Felhasználói bejelentkezési komponens.**
- *GyakoriAutokComponent*: **Gyakori autókat megjelenítő komponens.**
- *NavComponent*: **Navigációs komponens.**
- *FooldalComponent*: **Főoldali komponens.**
- *CreateHirdetesComponent*: **Hirdetés létrehozására szolgáló komponens.**
- *HirdetesComponent*: **Egyetlen hirdetés megtekintésére szolgáló komponens.**
- *HirdetesListComponent*: **Hirdetések listázására szolgáló komponens.**
- *ForumComponent*: **Fórum szekciót megjelenítő komponens.**
- *ProfilComponent*: **Felhasználói profil komponens.**
- *GyikAtiratasComponent*: **GYIK a járműregisztrációról.**
- *GyikAdasveteliComponent*: **GYIK a járművásárlásról.**
- *GyikKotelezoComponent*: **GYIK a kötelező biztosításról.**
- *GyikCascoComponent*: **GYIK a casco biztosításról.**
- *GyikBetetlapComponent*: **GYIK a biztosítási kötvényekről.**
- *RolunkComponent*: **Az egyesületről szóló információkat tartalmazó komponens.**
- *AdminComponent*: **Adminisztrátori funkciókat megjelenítő komponens.**
- *ModeratorsComponent*: **Moderátori funkciókat megjelenítő komponens.**

Útvonalak Konfigurációja

Az útvonal konfiguráció a következőképpen van meghatározva:

- **Alapértelmezett Útvonal:** A főoldalra (*/fooldal*) irányít át, amikor az alkalmazás a gyökér URL-en érkezik.
- **Felhasználói Hitelesítés:**
 - */register*: A regisztrációs oldalra navigál.
 - */login*: A bejelentkezési oldalra navigál.
- **Gyakori Autók:**
 - */gyakori-autok*: A gyakori autók oldalra navigál.
- **Navigáció:**
 - */nav*: A navigációs komponensre navigál.
- **Főoldal:**
 - */fooldal*: A főoldalra navigál.
- **Hirdetések:**
 - */create-hirdetes*: A hirdetés létrehozására szolgáló oldalra navigál.
 - */hirdetes*: Egyetlen hirdetés megtekintésére navigál.
 - */hirdetes-list*: A hirdetések listájára navigál.
- **Fórum:**
 - */forum*: A fórum komponensre navigál.



- **Felhasználói Profil:**
 - */profil*: A felhasználói profil oldalra navigál.
- **Gyakran Ismételt Kérdések (GYIK):**
 - */gyik-atiratas*: Járműregisztrációval kapcsolatos GYIK oldalra navigál.
 - */gyik-adasveteli*: Járművásárlással kapcsolatos GYIK oldalra navigál.
 - */gyik-kotelezo*: Kötelező biztosítással kapcsolatos GYIK oldalra navigál.
 - */gyik-casco*: Casco biztosítással kapcsolatos GYIK oldalra navigál.
 - */gyik-betetlap*: Biztosítási kötvényekkel



Tesztelés:

1. Backend:

```
// Login
POST http://localhost:5000/api/auth/login
Content-Type: application/json

{

    "nev" : "Admin",
    "jelszo" : "Adminpwd"
}

### 2. user számára bejelentkezés (POST)

POST http://localhost:5000/api/auth/login
Content-Type: application/json

{

    "nev" : "test",
    "jelszo" : "test"
}

###

// Register Admin
POST http://localhost:5000/api/auth/register
Content-Type: application/json

{

    "nev": "Admin",
    "jelszo": "Adminpwd",
    "email": "admin@admin.com",
    "tel": "+36201234456",
    "tipus": 0
}

### 2. user létrehozása (POST)
```



```

POST http://localhost:5000/api/auth/register
Content-Type: application/json

{
    "nev": "user1",
    "jelszo": "user12345",
    "email": "user1@test.com",
    "tel": "+36201234567",
    "tipus": 1
}

### Saját felhasználó lekérése

GET http://localhost:5000/api/auth/user
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsInRpCHVzIjoxLCJpYXQiOjE3NDE2MTkxNDUsImV4cCI6MTc0MTYyMjc0NX0.MRR77JtMaYQ5xLtRhqSZDjpDcV2Jhtdd9G-_toRrlrw

### minden felhasználó lekérése

GET http://localhost:5000/api/auth/users
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsInRpCHVzIjowLCJpYXQiOjE3NDE2MTk0OTIsImV4cCI6MTc0MTYyMzA5Mn0.1z7_EjPUTZVJ5c9ZZzf4G7msH6AHDRKhZLDmjXTnCwA

### Felhasználói adatok frissítése
POST http://localhost:5000/api/auth/user
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsImlhdcI6MTc0MTUzNjEyNywiZXhwIjoxNzQxNTM5NzI3fQ.o9AipgkiBJrTe6CrK7cOkmJHuiWrxd_NMUFhsElir4c
Content-Type: application/json

{
    "nev": "AdminUpdated",
    "email": "adminupdated@admin.com",
    "tel": "+36201234457"
}

### Felhasználó törlése

```



```

DELETE http://localhost:5000/api/auth/user
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSISImlhdcI6MTc0MTUzMID1MywiZXhwIjoxNzQxNTMzODYzfQ.H71IiEUBybnm3U26IausBKu2s1XC9FTvrXrpaym71lQ

### Hirdetesek Endpoints ###

GET http://localhost:5000/hirdetesek

###

GET http://localhost:5000/hirdetesek/6

### Create a new hirdetes ###
POST http://localhost:5000/hirdetesek
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSISInRpCHVzIjowLCJpYXQiOjE3NDE2MzIxMDcsImV4cCI6MTc0MTYzNTcwN30.EZ8n__b5qCLV5bpkmhGUmpICCTqm0PNLNHIXrH9iyRM

{
    "modell": "Golf",
    "marka": "Volkswagen",
    "ajtok_szama": 5,
    "hengerurtartalom": 1.6,
    "uzemanyag": "benzin",
    "evjarat": 2018,
    "futott_kilometer": 85000,
    "szin": "fekete",
    "sebessegvalto_tipus": "manuális",
    "kiegeszitok": "klíma, ülésfűtés",
    "muszaki_vizsga_ervenyes": "2026-06-30",
    "baleseti_elozmenyek": "nincs",
    "felhasznalo_id": 3,
    "adatok": "Megkímélt állapot",
    "ar": 4500000,
    "ert_telszam": "+36202939772"
}

### Hirdetés törlése (DELETE)

```



```

DELETE http://localhost:5000/hirdetesek/3
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsInRpCHVzIjowLCJpYXQiOjE3NDE2MTYyNDIsImV4cCI6MTc0MTYxOTg0Mn0.8MJfDrjY1dnM3PWA1yY6PTx6uk3CXJygB3nIOKs0g8I

### Poszt létrehozása (POST)

POST http://localhost:5000/api/poszt
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwibmV2IjoidGVzdCIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsInRpCHVzIjoxLCJpYXQiOjE3NDE2MTcwNjksImV4cCI6MTc0MTYyMDY2OX0.Usb-nObK9b_FlleOm kzQ5i9SzhIHTfvUna-c4HZXr8Y

{
    "tartalom": "Ez egy másik poszt"
}

### Poszt törlése (DELETE)

DELETE http://localhost:5000/api/poszt/4
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsInRpCHVzIjowLCJpYXQiOjE3NDE2MTcxMTAsImV4cCI6MTc0MTYyMDcxMH0.RUXv6v_3ap7JB8a5i2DwPaDdBmzuQi6jo3n4smx4SyQ

### Poszt módosítása (PATCH)

PATCH http://localhost:5000/api/poszt/1
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwibmV2IjoidGVzdCIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsImlhdcI6MTczOTczODk2MywiZXhwIjoxNzM5NzQyNTYzfQ.jKMyEaTqW4jOvSN4WZcZbO7YKCTqMwKcc6I37p8cpPc

{
    "tartalom": "Ez egy poszt"
}

### Posztok lekérése (GET)

```



```

GET http://localhost:5000/api/poszt/

### Posztok lekérése id szerint (GET)

GET http://localhost:5000/api/poszt/1

### Komment létrehozása (PUT)

POST http://localhost:5000/api/komment
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsImhdCI6MTczOTg5NzE3MSwiZXhwIjoxNzM5OTAwNzcxfQ.Ch0-fDedku3cZNelIdGVTyCqMitly_eGvukFto7uNrns

{
    "posztId": 2,
    "kommentTartalom": "Ez egy szuper poszt!"
}

### Komment lekérdezése (GET)

GET http://localhost:5000/api/komment/1
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsImhdCI6MTczOTg4MzE0MiwiZXhwIjoxNzM5ODg2NzQyfQ.mNG5RUF9CiCCPpFoyXyyNoV4iJnxNChryyxP5M7vrlo

### Komment törlése (DELETE)

DELETE http://localhost:5000/api/komment/1
Authorization: Bearer <token>

### Komment módosítása (PATCH)

PATCH http://localhost:5000/api/komment/1
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsImhdCI6MTczOTg4MzE0MiwiZXhwIjoxNzM5ODg2NzQyfQ.mNG5RUF9CiCCPpFoyXyyNoV4iJnxNChryyxP5M7vrlo

```



```
{
    "kommentTartalom": "Ez egy frissített komment!"
}

###

GET http://localhost:5000/api/poszt/1
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmV2IjoiQWRtaW4iLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsImlhdcI6MTczOTg4MzE0MiwiZXhwIjoxNzM5ODg2NzQyfQ.mNG5RUF9CiCCPpFoyXyyNoV4iJnxNChryyxP5M7vrlo
```

2. *Frontend:*

A frontend tesztelés célja a webalkalmazás felhasználói felületének és funkcionálisának megbízhatóságának, teljesítményének és felhasználói élményének biztosítása. A tesztelési folyamat során különböző tesztelési típusokat alkalmazunk, beleértve az egység- és integrációs teszteket, valamint a felhasználói teszteket. A tesztelés folyamata a fejlesztés során folyamatosan zajlik, és a felhasználói visszajelzések alapján módosításokat végzünk.

Tesztelési Típusok

1. Egység Tesztelés

- Cél: Az egység tesztelés célja a legkisebb kód egységek (pl. funkciók, komponensek) önálló tesztelése, hogy biztosítsuk azok helyes működését.
- Folyamat:
 - minden új funkció vagy komponens fejlesztésekor egység teszteket írunk.
 - A teszteket automatikusan futtatjuk a CI/CD folyamat részeként, hogy azonnali visszajelzést kapjunk a kód működéséről.
 - A tesztek lefedik a különböző bemeneti értékeket és a várt kimeneteket.



2. Integrációs Tesztelés

- Cél: Az integrációs tesztelés célja a különböző modulok és komponensek közötti interakciók tesztelése, hogy biztosítsuk a rendszer összhangját.
- Folyamat:
 - Az integrációs teszteket a fejlesztési ciklus során, a különböző modulok összekapcsolása után végezzük.
 - A tesztek ellenőrzik, hogy a különböző komponensek megfelelően kommunikálnak-e egymással, és hogy a rendszer a várt módon működik-e.
 - A tesztelés során figyelembe vesszük a felhasználói interakciókat is, például űrlapok kitöltését és gombok megnyomását.

3. Felhasználói Tesztelés

- Cél: A felhasználói tesztelés célja a végfelhasználók által végzett tesztelés, amely során a felhasználók visszajelzéseket adnak a felhasználói élményről és a funkcionalitásról.
- Folyamat:
 - A tesztelés során a felhasználókat meghívjuk, hogy használják az alkalmazást, és figyeljük a viselkedésüket.
 - Kérdőíveket és interjúkat használunk a felhasználói élmény és a funkcionalitás értékelésére.
 - A felhasználói visszajelzések alapján azonosítjuk a problémákat és a fejlesztési lehetőségeket.
 - A visszajelzések alapján módosításokat végezzünk az alkalmazásban, hogy javítsuk a felhasználói élményt.

Folyamat

1. Fejlesztés: A fejlesztési ciklus során folyamatosan írunk egység- és integrációs teszteket.
2. Felhasználói Tesztelés: A fejlesztés végén felhasználói teszteket végezzünk, és a visszajelzések alapján módosításokat hajtunk végre.
3. Iteráció: A tesztelési folyamat iteratív, folyamatosan javítjuk az alkalmazást a felhasználói visszajelzések és a tesztelési eredmények alapján.



Továbbfejlesztési lehetőségek:

1. Vállalati Felhasználó Jogkör

- Leírás: Bevezetésre kerülne egy vállalati felhasználói fiók, amely lehetővé tenné a cégek számára, hogy saját profiljukat kezeljék.
- Funkcionalitás:
 - Vállalkozások regisztrálhatnak és jogköröket kaphatnak, például hirdetések létrehozására, módosítására és törlésére.
 - A vállalati felhasználók számára külön adminisztrációs felület készülne, ahol nyomon követhetik a hirdetések teljesítményét és statisztikákat kaphatnak.

2. Interaktív Térkép

- Leírás: Az alkalmazásba integrálható egy interaktív térkép, amely lehetővé tenné a felhasználók számára, hogy vizuálisan elhelyezzék vállalkozásukat.
- Funkcionalitás:
 - A felhasználók a térképen megjelölhetik a vállalkozásuk helyét, és információkat adhatnak meg, például nyitvatartási időt, elérhetőséget és szolgáltatásokat.
 - A térkép lehetővé tenné a felhasználók számára, hogy könnyen megtalálják a közelben lévő vállalkozásokat.

3. Autós Esemény Szervezése

- Leírás: Az alkalmazás lehetőséget biztosítana autós események, mint például találkozók, versenyek vagy kiállítások szervezésére.
- Funkcionalitás:
 - Felhasználók eseményeket hozhatnak létre, amelyeket mások megtekinthetnek és csatlakozhatnak hozzájuk.
 - Az eseményekhez kapcsolódó információk, mint például helyszín, időpont és részvételi díj, könnyen elérhetők lennének.
 - Értesítések küldése a résztvevőknek az események közeledtével.

4. Mobil Applikáció



- Leírás: A webalkalmazás mobil verziójának kifejlesztése, amely lehetővé tenné a felhasználók számára, hogy okostelefonjukon is hozzáférjenek a szolgáltatásokhoz.
- Funkcionalitás:
 - Az applikáció tartalmazná a webalkalmazás összes funkcióját, optimalizálva a mobil felhasználói élményre.
 - Push értesítések küldése a felhasználóknak új hirdetésekről, eseményekről vagy ajánlatokról.
 - Offline hozzáférés bizonyos funkciókhoz, például a korábban megtekintett hirdetésekhez.

5. Több Hirdetés Összehasonlítása

- Leírás: A felhasználók számára lehetőséget biztosítana, hogy több hirdetést összehasonlítsanak egymással.
- Funkcionalitás:
 - A felhasználók kiválaszthatják a hirdetéseket, amelyeket össze szeretnének hasonlítani, és megtekinthetik a főbb jellemzőiket egy összehasonlító táblázatban.
 - Az összehasonlítás során figyelembe vehetők a hirdetések ára, állapota, jellemzői és egyéb releváns információk.
 - A felhasználók könnyebben hozhatnak döntéseket a hirdetések közötti választás során.



Irodalomjegyzék:

Backendhez használt dokumentációk:

2024.10.05. 8:00

- <https://expressjs.com/>
- <https://nodemon.io/>
- <https://www.npmjs.com/package/package>

2024. 10.28. 16:25:

- <https://sequelize.org/docs/v6/getting-started/>
- <https://jwt.io/>

2025.02.17 19:35:

- <https://blog.logrocket.com/multer-nodejs-express-upload-file/>

A frontendhez használt dokumentációk:

2024. 10.10. 16:20

- <https://angular.dev/>
- <https://getbootstrap.com/>

Gyakori autók funkcióhoz felhasznált források:

2025. jan. 8. 7:50 - 12:50

- https://nepitelet.hu/autok/peugeot/308_2007/alicsony_a_fogyasztasa/
- https://nepitelet.hu/autok/dacia/duster_2010/igazan_szeretheto_joszag/
- <https://forum.index.hu/Article/showArticle?t=9161039>
- <https://gepjarmuadat.hu/renault-clio-3-tipushibak-a-franciak-regi-kedvencenek-hibai/>
- https://tipushiba.hu/auto/skoda/octavia_20042012
- https://nepitelet.hu/autok/renault/clio_2005/kenyelem_sebesseg_megbizhatosag_dizajn_clio_i_ii/
- <https://www.autonavigator.hu/cikkek/milyen-egy-200-ezret-futott-159-millios-octavia-hasznaltteszt/>
- <https://alapjarat.hu/el-mondo/13-auto-elerheto-aron-azoknak-akik-most-kostolnak-bele-vezetes-oromebe>
- <https://autopovblog.hu/top-10-legjobb-auto-1-millio-forint-alatt-a-hasznalt-piacrol/>

Logo:

A logo készítéséhez elsősorban ötleteket gyűjtünk amelyeket később személyre szabunk a saját igényeinknek megfelelően, ehhez képgenerátort, majd képszerkesztő programokat használunk.

A logónk egy C betűt ábrázol melyben egy M betű helyezkedik el egy VW Beetle formatervéből kiemelve.



Színpaletta az oldalon:

- <https://colorhunt.co/>

Háttér:

<https://www.freepik.com/free-photos-vectors/car-seamless/7#uuid=e7d990c5-e2d3-4626-9e43-5ce85a745ebd>

