győri szakképzési centrum

Jedlik Ányos
Gépipari és Informatikai
Technikum és Kollégium

9021 Győr, Szent István út 7.
+36 (96) 529-480
+36 (96) 529-448

OM: 203037/003
jedlik@jedlik.eu
www.jedlik.eu

Záródolgozat feladatkiírás

Tanulók nevei: Pusoma Gergő, Székely Áron, Takács Marcell

Képzés: nappali

Szak: 5 0613 12 03 Szoftverfejlesztő és -tesztelő technikus

A záródolgozat címe:

CarScope

Konzulens: Sándor László

Beadási határidő: 2022. 04. 29.

Győr, 2022. 04. 29.

Módos Gábor
igazgató

Konzultációs lap

	A konzultáció		Konzulens aláírása
	ideje	témája	
1.	2022.02.15.	Témaválasztás és specifikáció	
2.	2022.03.14.	Záródolgozat készültségi fokának értékelése	
3.	2022.04.17.	Dokumentáció véglegesítése	

Tulajdonosi nyilatkozat

Ez a dolgozat a saját munkám eredménye. Dolgozatom azon részeit, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem.
Ha kiderülne, hogy ez a nyilatkozat valótlan, tudomásul veszem, hogy a szakmai vizsgabizottság a szakmai vizsgáról kizár és szakmai vizsgát csak új záródolgozat készítése után tehetek.

Győr, 2022. április 29.

Pusoma Gergő	Székely Áron	Takács Marcell

Tartalomjegyzék

1.1 Az oldal célja 5
2 CarScope felhasználói kézikönyv 6
2.1 Bejelentkezés/Regisztráció 7
2.2 Vendégoldal 8
2.3 Tájékoztató és kapcsolat oldalak 9
2.4 Főoldal/Autó lekérdezés 10
2.5 Sikeres lekérdezés 12
2.6 Termékek 13
2.7 Kosár 14
2.8 Rendelés véglegesítése 15
2.9 Kerékmagasság kalkulátor 16
3 CarScope-MyAdmin Felhasználói kézikönyv 17
3.1 A program használata 17
3.2 Az űrlapok helyes kitöltése 18
4 Backend dokumentáció 21
4.1 Fejlesztői környezet ismertetése: 21
4.1.1 Használt technológiák: 21
4.1.2 Adatbázis, kapcsolati diagram: 21
4.1.3 Könyvtár struktúra: 22
4.2 A weboldal működése: 23
4.2.1 Nyitó oldal 24
4.2.2 Kezdő oldal megjelenítése: 24
4.2.3 Paraméterek: 25
4.2.4 Regisztráció 26
4.2.5 Bejelentkezés 29

4.3 Honnan tudjuk, hogy tényleg be van-e jelentkezve a felhasználó? 29
4.4 Főoldal működését bemutató ábra 30
4.4.1 Jármű lekérdezés 31
4.4.2 Kosár 34
4.4.3 Termékek kilistázása 35
4.4.4 Termékek kosárhoz adása 35
4.5 Tesztelés 36
4.5.1 Regisztráció tesztelése 37
5 Frontend/Design dokumentáció 39
5.1 Mobilnézet 39
5.2 Főoldal/.js fájlok 41
5.3 Termékek/rendelés 42
6 CarScope-MyAdmin Dokumentáció 44
6.1 A program célja 44
6.2 A program felépítése 44
6.3 Indítás után 48
6.4 Keresés 50
6.5 Módosítás és törlés 51
6.6 Új adat felvitele 52
6.7 Az űrlapok visszaállítása 54
7 Konklúzió 55
8 Telepítés 56
9 Fejlesztők 57
10 Források 58

1.1 Az oldal célja

A **CarScope** oldal megálmodásakor a célunk egy Magyarországon könnyen átlátható használt és/vagy behozott autó lekérdező weboldal volt, ami nemcsak információt tud biztosítani a felhasználónak, hanem különböző termékeket is.

A magyar állami rendszám lekérdezés korlátozott adattartalommal bír, csak a hazai forgalomba helyezésű járművek adatait tartalmazza attól a pillanattól kezdve, hogy itthon forgalomba helyezték őket. Oldalunknak ezzel szemben hazai és külföldi szervízek is szolgáltatnak adatokat melyek aztán felkerülnek az adatbázisba.

Egy, a felhasználó számára egyértelműen kezelhető weboldalt szerettünk volna biztosítani, valamint az esetleges kérdések megválaszolására egy kapcsolat oldalt is létrehoztunk.

Az oldal megalkotásakor szerettük volna ösztönözni a felhasználót, hogy hozzon létre egy saját profilt a weboldalon, ezzel gyorsítva a vásárlás folyamatát, valamint, ha a későbbiekben további funkciókkal bővül a weboldal, akkor azok könnyedén személyre szabhatóak legyenek. Éppen ezért a legtöbb funkció csak regisztráció után érhető el.

Az oldalt szerettük volna egyéb hasznos funkciókkal ellátni, mint például egy kerékmagasság kalkulátor, ahol ki lehet számolni, hogy a megvásárolni tervezett gumival milyen magas lenne a kerék.

Az autókról megjelenített információk nagyban segítenek összehasonlítani az autó tényleges állapotát a hirdetésben látottakkal, ezzel megkönnyítve a böngészést, vásárlást.

2 CarScope felhasználói kézikönyv

1. kép

A weboldal a localhost:2001-es porton fut, melyet megnyitva a kezdőlap fogadja a felhasználót.

1. Ha a felhasználó a „Bejelentkezés most” vagy a navigáció sávon a „Bejelentkezés” gombra kattint, a bejelentkezési felület jelenik meg a képernyőn (3. kép).

2. A navigációs sávon a „Regisztráció” gombra kattintva pedig a regisztráció felület lesz látható (4. kép).

3. A „Folytatás vendégként” gombra kattintva a vendégoldalra lesz a felhasználó irányítva (5. kép).

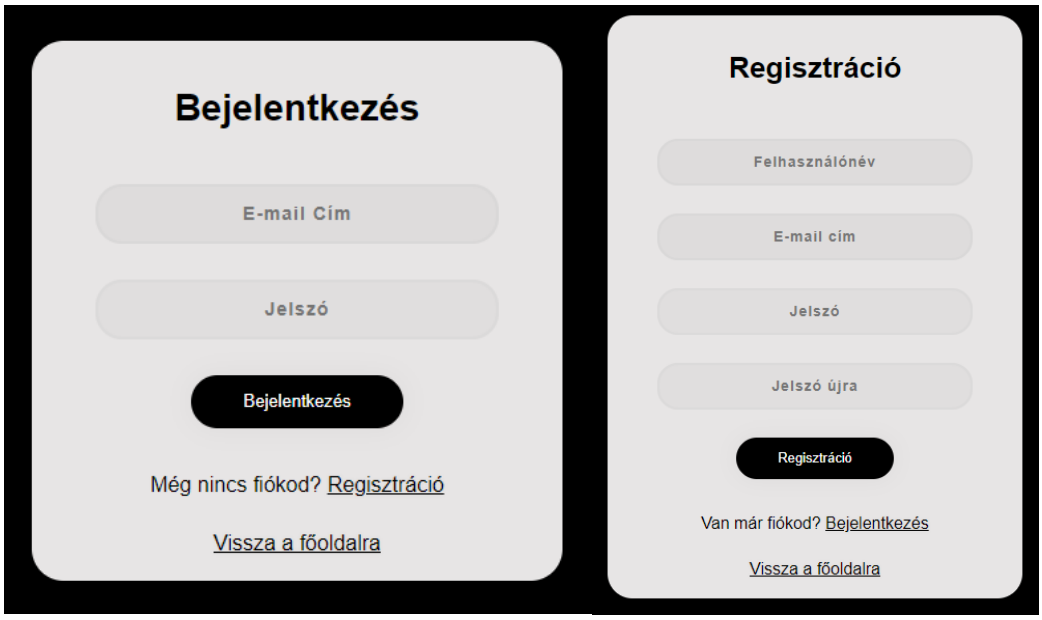
4. A „Tudj meg többet” menüpontra kattintva a súgó felületre kerül a felhasználó, ahol további információkat tudhat meg az oldalról (6. kép).

5. A „Kapcsolat” menüpontra kattintva a kapcsolat felület nyílik meg. Itt a felhasználó üzenni tud a fejlesztőknek, ha bármi észrevétele van (7. kép).

2. kép

6. A kezdőlap alsó részén található ikonok a CarScope közösségi média oldalaira mutatnak (2. kép).

2.1 Bejelentkezés/Regisztráció



3-4. kép

A regisztráció/bejelentkezés fontos az oldal használatának szempontjából.
Bizonyos funkciók az oldalon csak akkor elérhetőek, ha a felhasználó már bejelentkezett. Ehhez először is regisztráció szükséges. Ha a felhasználó helyesen tölti ki a beviteli mezőket, akkor sikeres lesz a

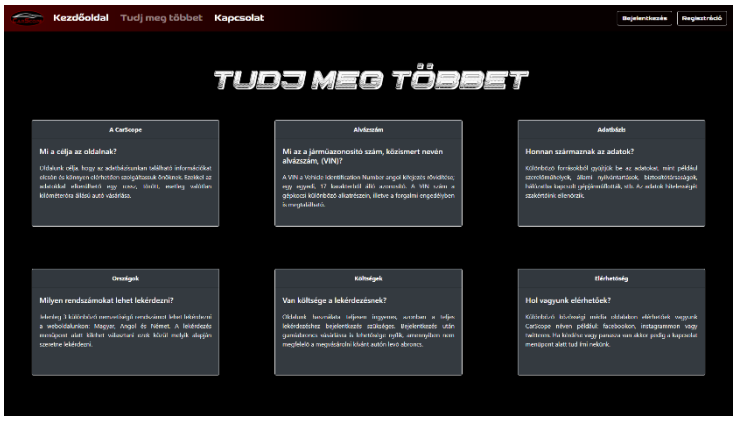
bejelentkezés/regisztráció. Ha valamilyen feltételnek nem felelnek meg a bevitt karakterek, arról megfelelő hibaüzenet tájékoztatja a felhasználót.
A sikeres regisztrációról a felhasználó email értesítést kap.

2.2 Vendégoldal

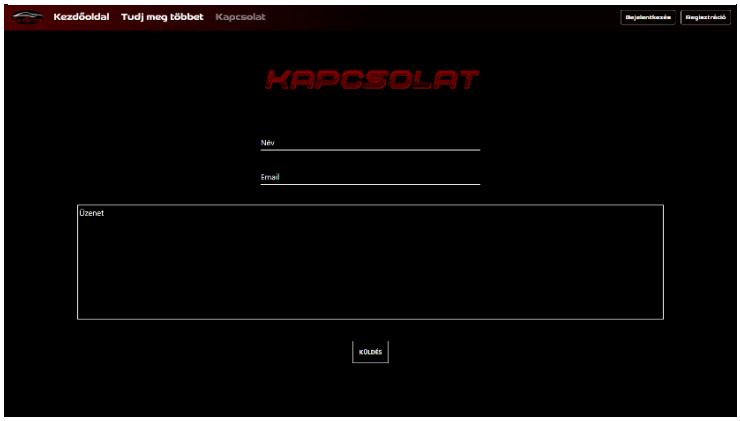
5. kép

A vendégoldalon a felhasználó számára csak a lekérdezés menüpont érhető el, a további funkciók használatához be kell jelentkeznie. Itt csak magyar rendszám alapú lekérdezés lehetséges. Ha a felhasználó nem létező vagy általunk fel nem jegyzett rendszámot ad meg, az oldal erről értesíti. Amennyiben az adatbázisban eltárolt rendszámot ad meg, abban az esetben az oldal megjelenít egy a gépjárműről készült képet és pár alapvető információt az autóról. Részletesebb lekérdezés azonban csak a bejelentkezés után lehetséges.

2.3 Tájékoztató és kapcsolat oldalak



6. kép



7. kép

A ,,Tudj meg többet” menüpont alatt a felhasználó gyakran feltett kérdésekre kaphat választ, valamint az oldallal kapcsolatos további információkat tudhat meg.

A ,,Kapcsolat” menüpontnál a felhasználó üzenetet küldhet a fejlesztőknek, ha észrevétele, kérdése vagy problémája van az oldallal kapcsolatban. A név és az e-mail cím mezők kitöltése kötelező. Ezt az üzenetet az adminok emailben fogják megkapni.

Ha a felhasználó sikeresen bejelentkezett, abban az esetben a Főoldalra lesz átirányítva, ami egyben a jármű lekérdezési felület is. (8. kép).

2.4 Főoldal/Autó lekérdezés

8. kép

1. A felhasználó a CarScope logóra kattintva a térhet vissza a kezdőlapra, természetesen bejelentkezve marad (9. kép).

2. A navigációs sávon a „Kijelentkezés” gombra kattintva pedig a felhasználó kijelentkezik a fiókjából, majd szintén a kezdőlapra lesz irányítva.

3. A „Termékek” menüpontra kattintva a felhasználó a termékek oldalra jut, ahol az oldal által árusított gumiabroncsokból tud vásárolni (15. kép).

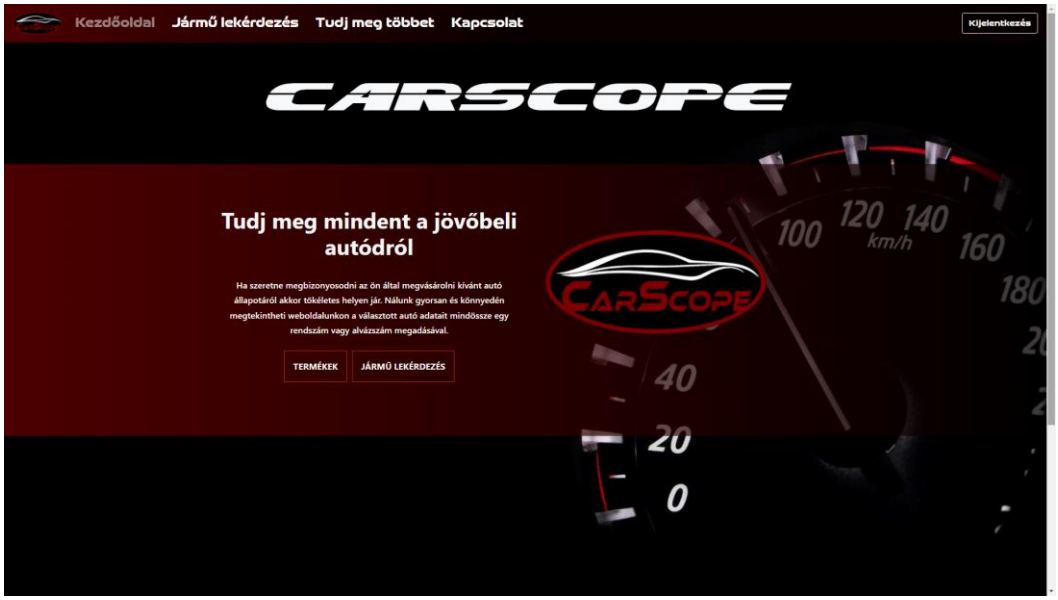
4. A „Kosár” menüpontra kattintva a felhasználó megtekintheti, hogy milyen termékek vannak eddig a kosarában (17. kép).

5. A „Magasság kalkulátor” menüpontra kattintva egy kalkulátor nyílik meg. Itt a felhasználó kitudja számoltatni az oldallal a várható kerékmagasságot a megvásárolni kívánt gumi méretei alapján (19. kép).

6. Az oldal közepén egy lenyíló lista látható, melyen a felhasználó kitudja választani, hogy alvázszám vagy rendszám alapján szeretne lekérdezni. Az első lekérdezés előtt a felhasználó tájékoztatást kap arról, hogy milyen formában kell megadnia a rendszámot vagy alvázszámot, az információ a weboldal alján jelenik meg (11-12. kép). Amennyiben az alvázszámot választja a felhasználó, ennek

megadásához egy beviteli mező jelenik meg, majd a ,,Keresés” gombra kattintva indíthat lekérdezést az adott alvázszámra.

Ha a felhasználó a rendszám alapú lekérdezést választja, akkor megjelenik még egy lenyíló lista, ahol kiválaszthatja, hogy milyen nemzetiségű rendszámot szeretne megadni. A választott országnak megfelelő beviteli mező jelenik meg. (10. kép).



9. kép

Ha a felhasználó rossz vagy általunk nem tárolt alvázszámot vagy rendszámot ad meg, abban az esetben a ,,Keresés” gomb megnyomása után hibaüzenet jelenik meg.

10. kép

11-12. kép

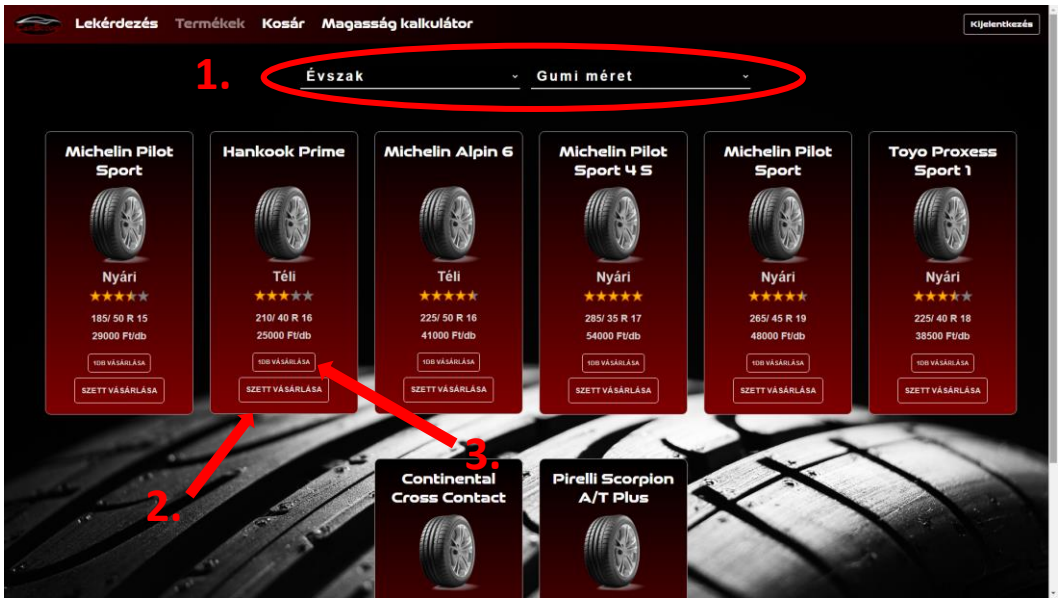
2.5 Sikeres lekérdezés

13. kép

A sikeres lekérdezés után megjelenik az autó képe, hasonlóan ahhoz, amit a vendégoldalon is láthattunk, itt azonban már az autó értékelése is látható, valamint a ,,Részletek" gombra kattintva az autó részletesebb leírását is megtekinthetjük (14. kép)

14. kép

2.6 Termékek



15. kép

A ,,Termékek” oldalon látható az összes termék, ami elérhető a weboldalon.

1. Az oldal tetején látható 2 darab lenyíló listán szűrni lehet a találatokat évszak és/vagy gumiméret alapján.

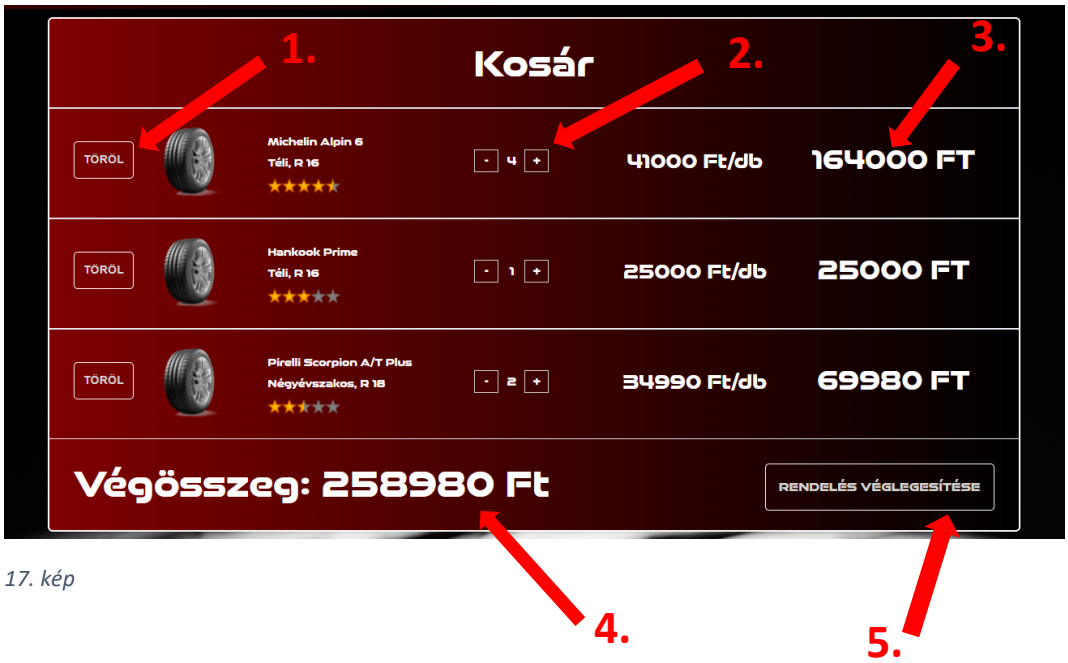
2. Minden termék egy kártyában van elhelyezve. A kártya alján 2 gomb található. a felső gombot akkor válasszuk, ha az adott gumiból csak 1db-ot szeretnénk hozzáadni a kosárhoz.

3. Az alsó gombbal az adott gumiból 4 darabot, azaz egy teljes szettet tud a felhasználó hozzáadni a kosárhoz.

16. kép

Ha a felhasználó még nem adott hozzá terméket a kosárhoz, akkor a „Jelenleg üres a kosár” felirat jelenik meg. Ha vannak hozzáadott termékek, akkor azok megjelennek a kosárban (16. kép).

2.7 Kosár



17. kép

1. A ,,Töröl” gombbal az adott termék minden darabját el lehet távolítani a kosárból.

2. A termékinformációktól jobbra az adott termékből a kosárban található mennyiség látható. A plusz és mínusz gombokkal pedig a terméknek a darabszámát egyesével tudjuk változtatni. Ha 1 alá csökkentjük a darabszámot a termék eltűnik a kosárból.

3. A kosár jobb oldalán látható összegek az adott termékből vásárolt mennyiségnek az összára. Ez az ár dinamikusan változik, ahogy változtatjuk a darabszámot.

4. A kosár bal alsó sarkában a végösszeg látható mely az összes hozzáadott guminak az ára, ez szintén dinamikusan változik.

5. A kosár jobb alsó sarkában a rendelés véglegesítése gombra kattintva egy felugró ablakon megadhatjuk a hiányzó adatokat a rendelés leadásához. (18. kép)

2.8 Rendelés véglegesítése

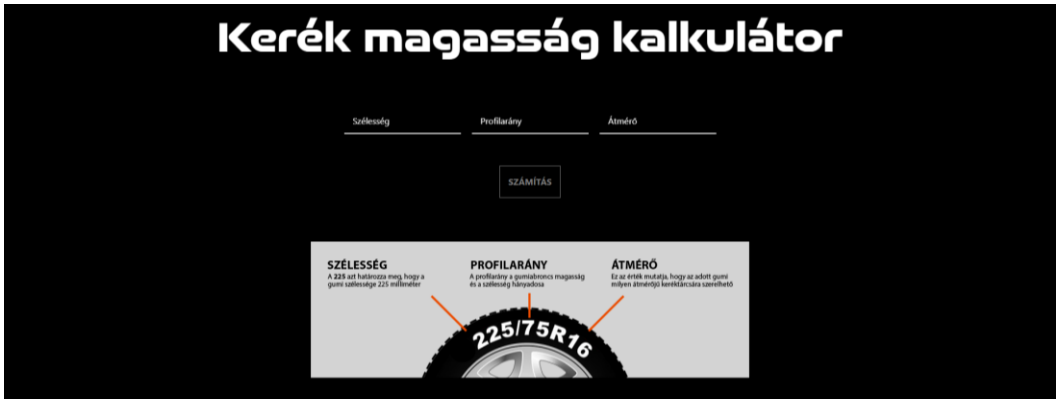
18. kép

A kért mezők kitöltése kötelező, valamint ha helytelen adatot ad meg a felhasználó valamelyik mezőben pl.(számot a névhez) akkor piros

lesz a beviteli mező, míg azt ki nem javítja a felhasználó. Ha minden mezőt helyesen tölt ki a felhasználó az addig letiltott gomb kattinthatóvá válik és a felhasználó letudja adni a rendelését. A sikeres rendelésről a felhasználó az megerősítő emailt kap.

Az általános szerződési feltételeket kötelező elfogadni a rendeléshez, amit meg lehet tekinteni, ha a nevében található linkre rákattintunk. A szerződés egy újabb felugró ablakban jelenik meg.

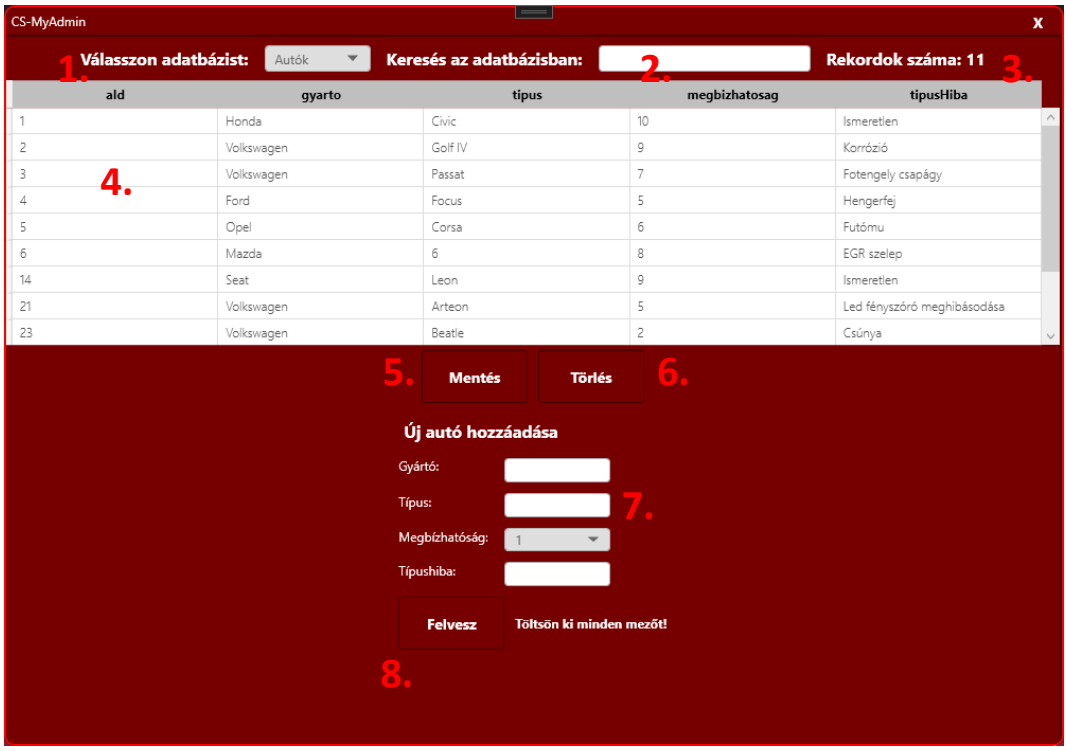
2.9 Kerékmagasság kalkulátor



19. kép

A ,,Magasság kalkulátor” oldalon a vásárló megtudhatja, hogy a megvásárolni kívánt gumival mekkora lesz a kerékmagasság. Ehhez 3 adatot kell a gumiról megadnia, melyeket mind megtalál az oldalunkon. Az első a gumi szélessége, a második a gumi a profilaránya, végül pedig az átmérője. Ezt követően a ,,Számítás” gombra kattintva megjelenik a számított kerékmagasság. Ha az információkat közlő képre visszük az egeret, akkor az 180° fokkal megfordul a kép és további információkat tudhatunk meg.

3 CarScope-MyAdmin Felhasználói kézikönyv



A program felépítése

3.1 A program használata

Indítás után a felhasználót egy táblázat és egy űrlap fogadja. A táblázatban lehetőségünk van az adatbázisban már szereplő rekordok módosítására, az alatta található űrlap segítségével pedig új rekordot csatolhatunk az adatbázishoz. Az adatbázis utf8mb4_hungarian_ci karakterkódolást használ.

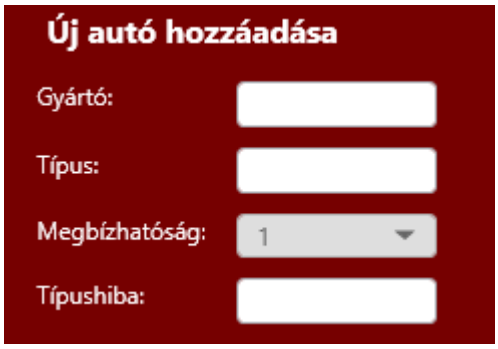
1. Az alkalmazás az adatbázis három táblájának szerkesztésére készült, a lenyíló lista segítségével választhatunk a megjeleníteni kívánt táblák között

2. A beviteli mezőbe beírt szöveg alapján kereshetünk egy adott rekordot vagy rekordokat az adatbázisban:
   - autót gyártó és típus szerint,
   - autó infó-t rendszám és alvázszám szerint,
   - gumiabroncsot pedig gyártó és évszak alapján.

Volkswagen gyártmányú járművek

3. az éppen megjelenített rekordok számát az automatikusan frissülő felirat mutatja
4. az adatok megjelenítésére szolgáló táblázat, amelynek tartalma táblák közti váltásnál folyamatosan változik. Ha egy mezőre duplán kattintunk, az abban szereplő érték módosíthatóvá válik
5. a gombra kattintva véglegesíthetjük a táblázatban végrehajtott módosításokat
6. ha a táblázat egy sorát kiválasztjuk, erre a gombra kattintva törölhetjük az adatbázisból
7. az új rekord felvételére szolgáló űrlap
8. az űrlap kitöltése után a gomb segítségével adhatjuk hozzá az új rekordot az adatbázishoz

3.2 Az űrlapok helyes kitöltése



Új autó

Gyártó: az autó márkajelzése; betű és szám is szerepelhet
Típus: az autó típusa; betű és szám
Megbízhatóság: az autó megbízhatósága; 1: kevésbé megbízható, 10: nagyon megbízható
Típushiba: a gépjárműről általánosságban elmondható egy vagy több hátrányos jellemző; betű és szám

Új info

Rendszám: országtól függően 7-8 karaktert hosszúságú, betűt és számot is tartalmazhat

Alvázszám: az autó nemzetközi, egyedi azonosítója, 17 karakter hosszúságú betű-szám sor

Futott km: az autóval összesen megtett távolság kilométerben, 10 milliónál kisebb szám lehet

Évjárat: az autó gyártási éve, 1901 és 2155 közötti szám lehet

Vezetett szervizkönyv: Az autó vezetett szervizkönyvvel rendelkezik: Igen/Nem

Műszaki érvényes: az autó műszaki vizsgájának érvényességi határideje

Autó azon.: az adatok mely, az autó táblában szereplő autóhoz tartoznak (lenyíló lista)

Képcím: a járműről készült kép URL-címe

Állapot: az autó általános állapota (pl. frissen felújított)

Okmányok: az autó milyen nemzetiségű és érvényességű okmányokkal rendelkezik

Gumiabroncs: az autón található gumi típusa

Sérülés: Az autón található sérülések helye/mértéke

Új gumi

Gyártó: az gumi márkajelzése; betű és szám is szerepelhet
Évszak: a gumi típusa
Kategória: a gumi értékelése: 1 – nagyon rossz, 10 – nagyon jó
Ár: gumi ára forintban; csak szám
Átmérő: a gumihoz megfelelő felni átmérője collban; csak szám
Oldalfal: a gumi oldalprofiljának aránya a futófelület szélességéhez képest; csak szám
Szélesség: a gumi futófelületének szélessége mm-ben; csak szám

4 Backend dokumentáció

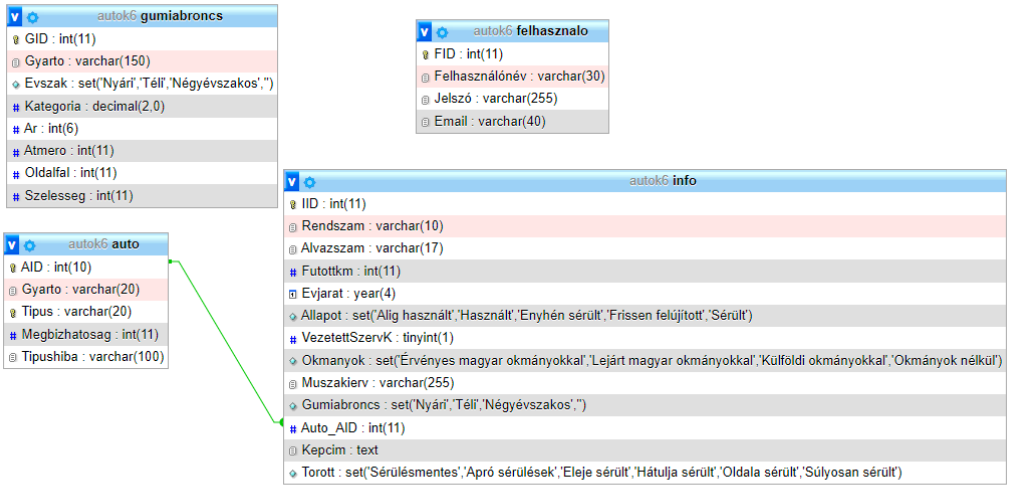
4.1 Fejlesztői környezet ismertetése:

Fejlesztő program: Visual Studio Code 2019

4.1.1 Használt technológiák:

Node.js, Vue.js, MySql

4.1.2 Adatbázis, kapcsolati diagram:

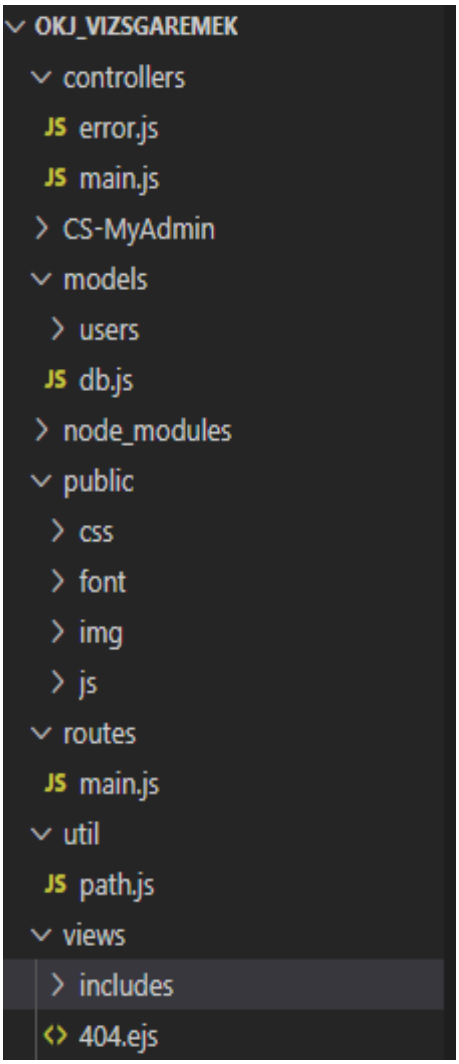


Három részre bontható az adatbázis, az első a személygépjármű lekérdezésére szolgáló adatokat tartalmazza két táblában, a második a gumiabroncsok értékesítésével kapcsolatos. A harmadik pedig a regisztrált felhasználók adatait tartalmazza.

Az **„auto”** táblában találhatók egy autó általános adatai, így egy adott típusú és évjáratú járműből egy rekordot tartalmaz. Az **„AID”** az **„info”** táblában idegenkulcsként tartalmazza az egyedi információkat az adott járműről.

A **„felhasznalo”** táblában kerül rögzítésre minden felhasználó sikeres regisztrációja, **md5** titkosított jelszava, emailcíme, és felhasználóneve. A **„gumiabroncs”** tábla pedig tartalmazza a webshopunkban elérhető gumiabroncsok minden adatát.

4.1.3 Könyvtár struktúra:



controllers – a **routes** mappából meghívott metódusok és eljárások találhatóak ebben a mappában

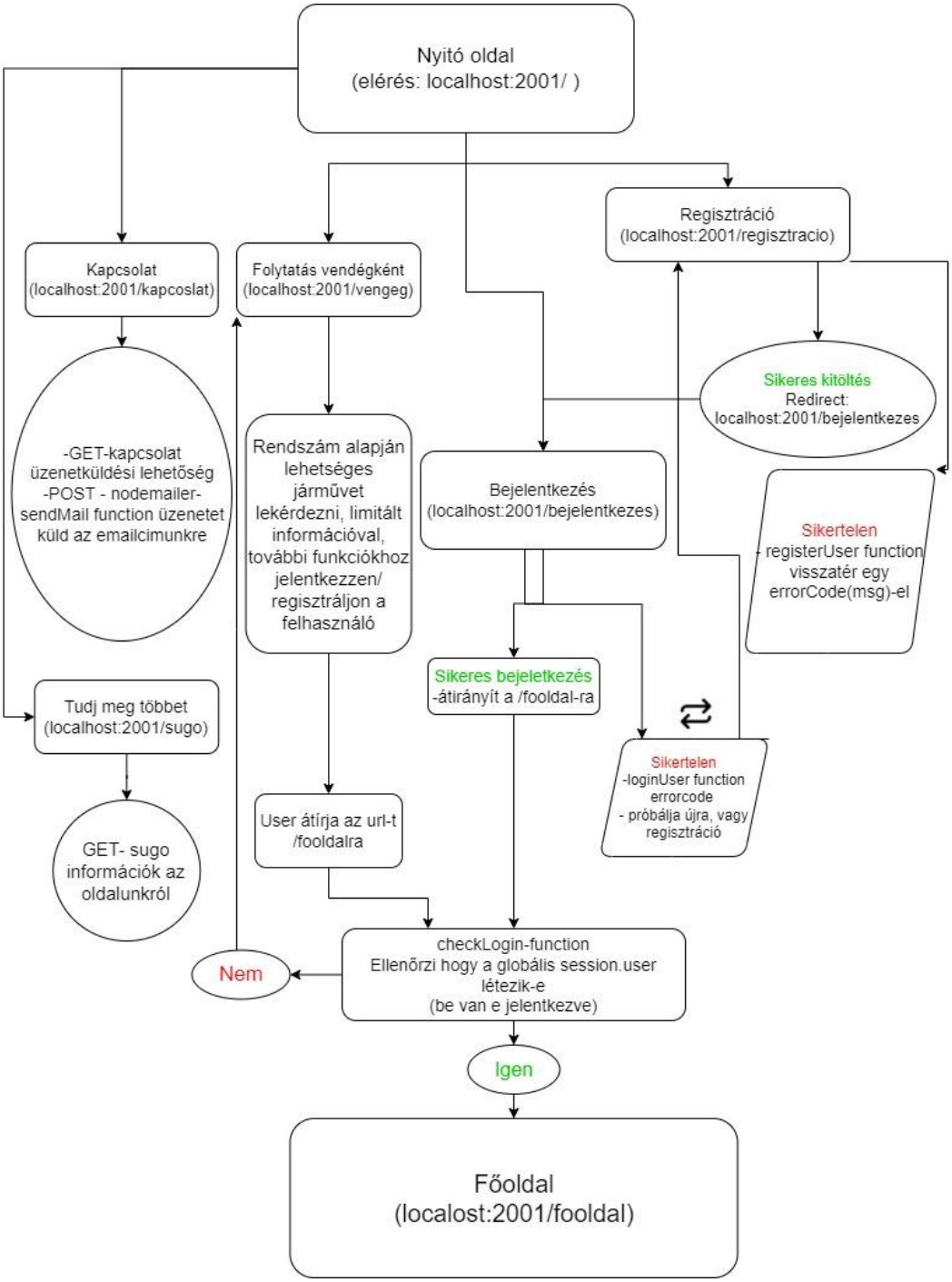
models – a felhasználók regisztrációját, autentikációját, és az adatbázisból való lekérdezések kódjait tartalmazza melyeket exportálás után a **/controllers/main.js** fájlba importálunk

public – a frontendhez szükséges fájlokat tartalmazza a megfelelő mappákban

routes – a szerver útvonalait tartalmazza

views – tartalmazza a renderelésre kész .ejs fájlokat, a **/include** a views .ejs fájlaiba beépített kódrészleteket ezzel elkerülve az ismétlődő kódokat

4.2 A weboldal működése:



4.2.1 Nyitó oldal

A szerver a **localhost:2001**-es porton fut, ha megkapja böngészőből a „**/**” kérést akkor a megfelelő oldalt legenerálja a felhasználónak egy .ejs állományban, a megadott paraméterekkel.

A **/routes/main.js** állomány felelős a kérések irányításáért:

(kódrészlet:)

```
router.get('/bejelentkezes',
mainController.getBejelentkezes);
router.post('/bejelentkezes',
mainController.postLogin);
router.get('/bejelentkezes',
mainController.checkLogin);
router.get('/regisztracio',
mainController.getRegisztracio);
router.post('/regisztracio',        validationForm.form
,mainController.validateRegistration);
router.get('/sugo', mainController.getSugo);
router.get('/kapcsolat', mainController.getContactus);
router.post('/contactuskuld',
mainController.postContactus);
```

A fent látható módon kerülnek meghívásra az adott útvonalakhoz rendelt metódusok a **/controllers/main.js** fájlból.

A POST kéréseket mindig egy **post** típusú **form** küldi a megfelelő **.ejs** állományból és ennek megfelelően kerülnek meghívásra a POST metódusok.

4.2.2 Kezdő oldal megjelenítése:

```
router.get('/', mainController.getIndex);
```

Ezzel meghívásra kerül a **/controllers/main.js** fájlban található függvény, ami lerendereli a **/views/kezdolap.ejs** fájlt a user paraméterrel:

```
//Kezdő oldal
exports.getIndex = (req, res, next) => {
    res.render("kezdolap", {
        pageTitle: "CarScope",
        path: "/",
        user: req.session.user,
```

4.2.3 Paraméterek:

A kezdő oldal megjelenítésekor az .ejs fájlnak átadásra kerül a fenti példában látható módon egy „user” nevű paraméter ami egy globális változótól kapja értékét (tartalmazza, hogy a felhasználó be van-e jelentkezve)

Az .ejs kódrészletben egy if() elágazásban ennek a változónak az értéke határozza meg a navigációs sáv kinézetét a következő módon:

```
     <% if(user == undefined) { %>
          <form class="form-inline my-2 my-lg-0">

                    <a  class="btn  btn-outline-light"
href="bejelentkezes"
            type="submit">Bejelentkezés</a>
              
                <a  href="regisztracio"  class="btn  btn-
outline-light"
            type="submit">Regisztráció</a>
          </form>
      <% }else{ %>
              <form  class="form-inline  my-2  my-lg-0"
method="post"
          action="/kijelentkezes">
            <button class="btn btn-outline-light"
            type="submit">Kijelentkezés</button>
          </form>
      <% } %>
```

Amennyiben a felhasználó nincs bejelentkezve (a user paraméter üres), a **„Bejelentkezés”**, **„Regisztráció”** gombok lesznek láthatóak, ellenkező esetben a **”Kijelentkezés”** gomb.

4.2.4 Regisztráció

A regisztrációs űrlap kitöltése után a felhasználó rákattint a **„Regisztráció”** gombra és lefut a **/regisztracio** form, egy post kérés. A **/controllers/main.js** - meghívja a **validateRegistration** metódust:

```
exports.validateRegistration = (req, res, next) => {
    let msg = [];
    const error = validationResult(req);
    const errorList = error.array();
    errorList.forEach((element) => {
        msg.push({ param: element.param, msg: element.msg
});
    });
    if (!error.isEmpty()) {
        res.render("regisztracio", {
            pageTitle: "CarScope - Regisztráció",
            path: "/regisztracio",
            errorCode: msg,
            user: req.session.user,
        });
    } else {
          register.registerUser(req, res, function(err,
data) {
            if (err) {
                res.json({ error: true });
            } else {
                res.redirect("/");
            }
        });
    }
```

Az **express-validator** importálása után feliratkozunk a csomagokra és egy **„error”** változóban tároljuk a **/models/users/validate.js** által visszaadott hibaüzenetet.

Ha a felhasználó valamit rosszul töltött ki, a hibaüzenet kiírásra kerül a regisztrációs oldalon.

Ellenkező esetben lefut a **/models/users/register.js – registerUser** metódusa

(kódrészlet:)

```
register.prototype.registerUser = function(req, res,
next) {
        secret = "!%/=._"+req.body.jelszo+"!%/._";
        jelszo = req.body.jelszo;
        md5Hasher = crypto.createHmac("md5", secret);
        jelszo = md5Hasher.update(jelszo).digest("hex");
                params = [req.body.felhasznalonev,
req.body.email,jelszo,
     req.body.password_again,0],
        checkAvailabilityQuery = 'SELECT Felhasználónév,
Email FROM felhasznalo WHERE
Felhasználónév = ? OR Email = ?',
                registerUserQuery = 'INSERT INTO
felhasznalo(Felhasználónév, Email, Jelszó) VALUES
(?,?,?)',
```

A felhasználó által megadott jelszó először titkosításra kerül az **md5** egyirányú kódolási szabvány szerint.

A **checkAvailabilityQuery** mysql lekérdezés ellenőrzi, hogy a felhasználó már szerepel-e az adatbázisban, ha nem, a **registerUserQuery** insert into beszúrja az adatokat az adatbázis **„felhasznalo”** táblájába.

Ha sikeres a regisztráció a **Nodemailer** használatával emailt küldünk a felhasználónak a sikeres regisztrációról:

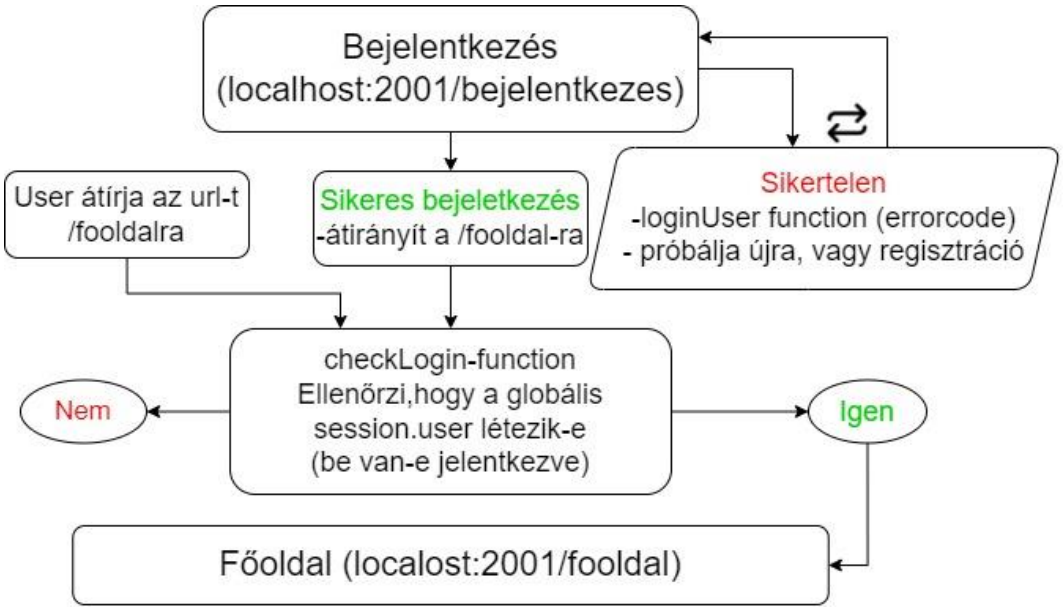
```
var nodemailer = require('nodemailer');
var transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
        user: 'carscope.site@gmail.com',
        pass: 'Carscope2022'
    }
});
```

A nodemailer levelező szerverként használja a gmail fiókunkat. A működése érdekben azonban szükséges volt a CarScope gmail fiók biztonsági beállításait átállítani, hogy engedélyezze külső alkalmazások számára a levelező használatát.

```
var mailOptions = {
                        to: req.body.email,
                        subject: 'SIKER! - CarScope',
                        text: 'Sikeresen regisztrál a
                               CarScopeoldalon!'
                    };
              transporter.sendMail(mailOptions,function(
             error, info){
               if (error) {console.log(error);
                    } else {
                        connection.release();
                        callback(null,null);
                        res.redirect('/bejelentkezes')
                    }
          });
```

Ha sikerült a felhasználó regisztrálása és az email elküldése, a **/bejelentkezes** oldal jelenik meg a felhasználó számára.

4.2.5 Bejelentkezés



4.3 Honnan tudjuk, hogy tényleg be van-e jelentkezve a felhasználó?

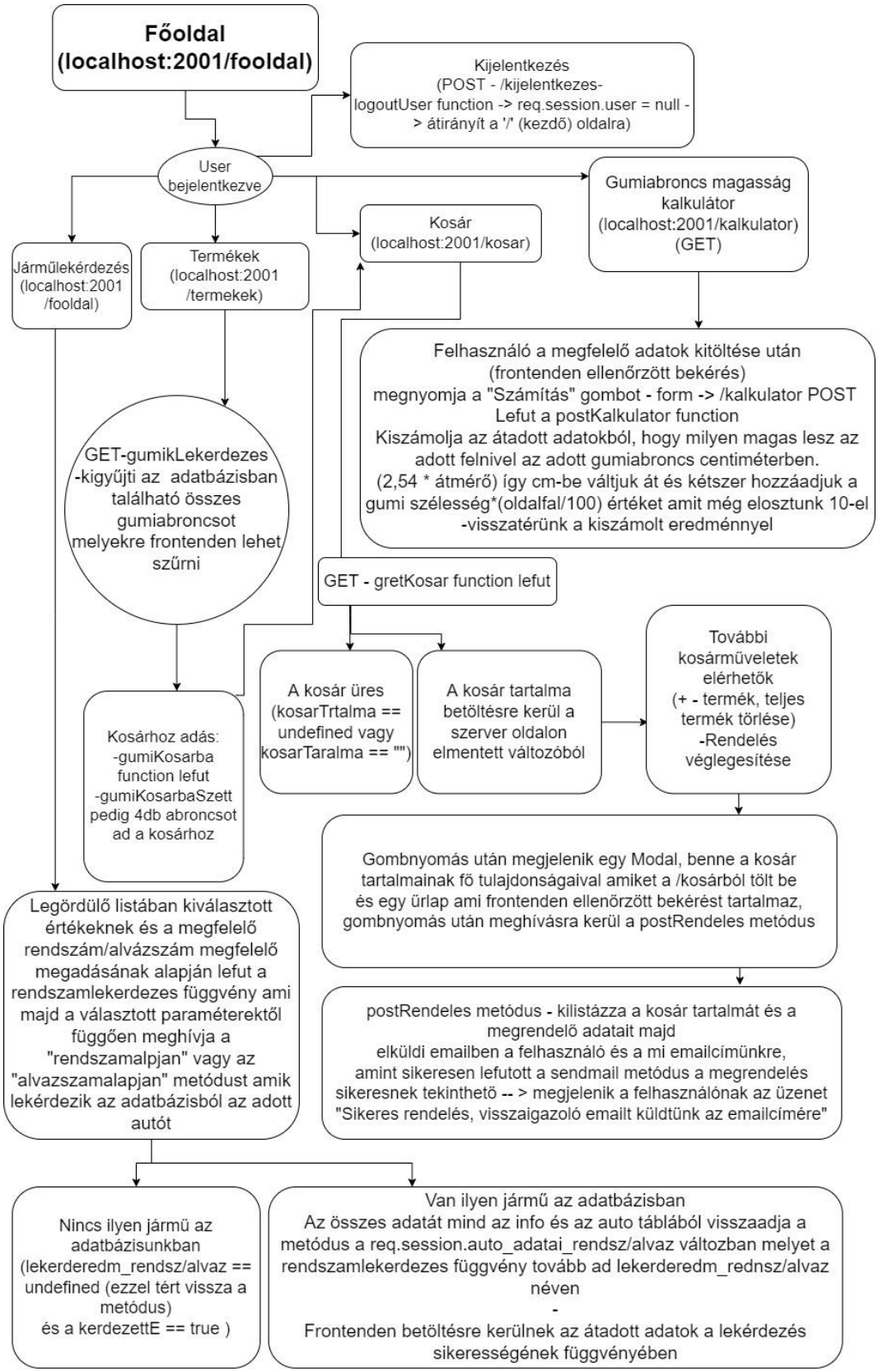
Amennyiben sikeres a bejelentkezés, vagy a felhasználó olyan url-t ír be a keresőbe melynek eléréséhez szükséges a bejelentkezés, az oldal betöltése előtt a szerver ellenőrzi, hogy az adott helyről érkező kérést végrehajtó felhasználó be van-e jelentkezve.

/controllers/main.js (122.sor)

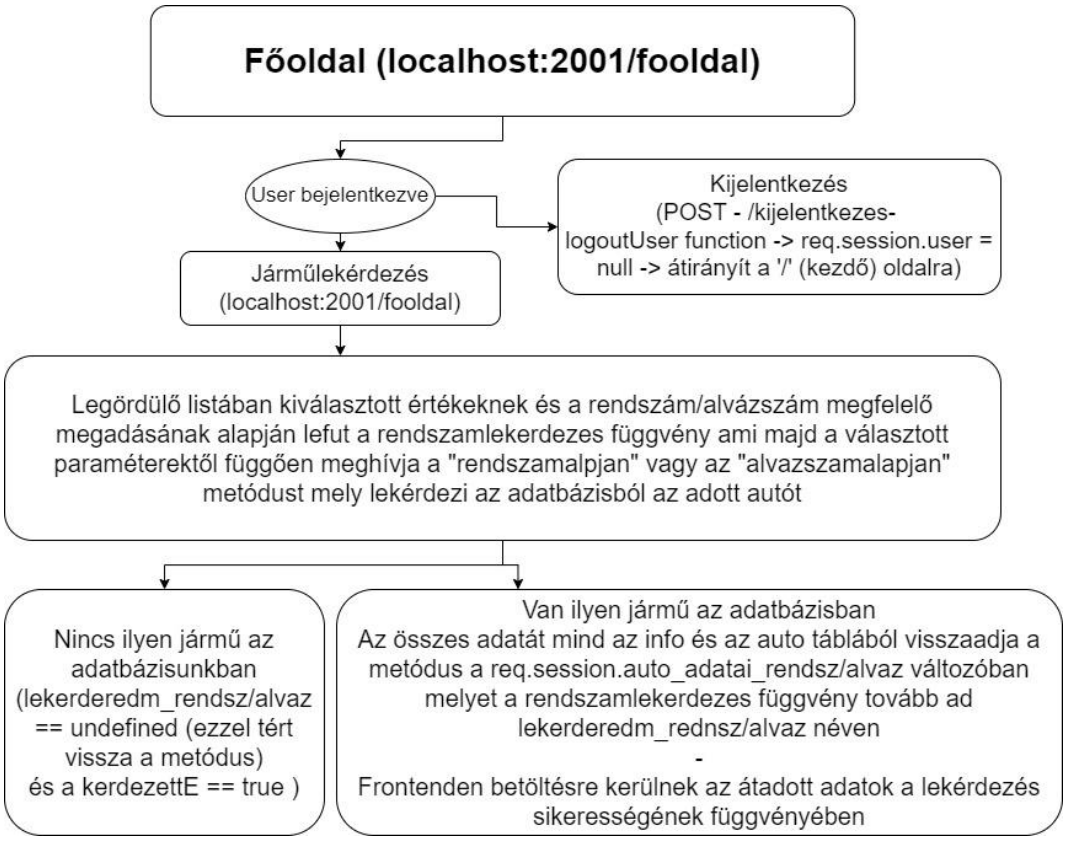
```
//Be van-e jelentkezve
exports.checkLogin = (req, res, next) => {
    if (req.session.user != null) {
        res.redirect("/fooldal");
    } else {
        res.redirect("/vendeg");
```

Amennyiben a **session** által létrehozott **user** változó létezik, tehát nem „null” vagy „undefined”, a szerver átirányít a **/fooldal** url-re, és a felhasználó bejelentkezve folytathatja tovább az oldal használatát.

4.4 Főoldal működését bemutató ábra



4.4.1 Jármű lekérdezés



Miután a felhasználó megadta milyen paraméter alapján szeretne jármű adatokat lekérdezni, és helyesen kitöltötte a beviteli mezőt (a hibás adatok bevitelét frontenden ellenőrizzük, így backenden már nem szükséges az ellenőrzés), a „Keresés” gomb lenyomására a **/fooldal** form **POST** kérése lefut - **/routes/main.js** felelős az irányításért – meghívásra kerül a **/controllers/main.js** – rendszamlekerdezes metódusa:

```
exports.rendszamlekerdezes = (req, res, next) => {
    if (req.body.R == "True") {
        lekerdR.rendszamalapjan(req, res, function(err,
data) {
```

A legördülő listából átadott érték alapján (**body.R** vagy **body.A**) eldől, hogy milyen paraméter alapján kérdezünk le járművet és ennek függvényében meghívásra kerül a **/models/users/auto.js** -ből importált „rendszamalapjan” vagy „alvazszamalapjan” függvény.

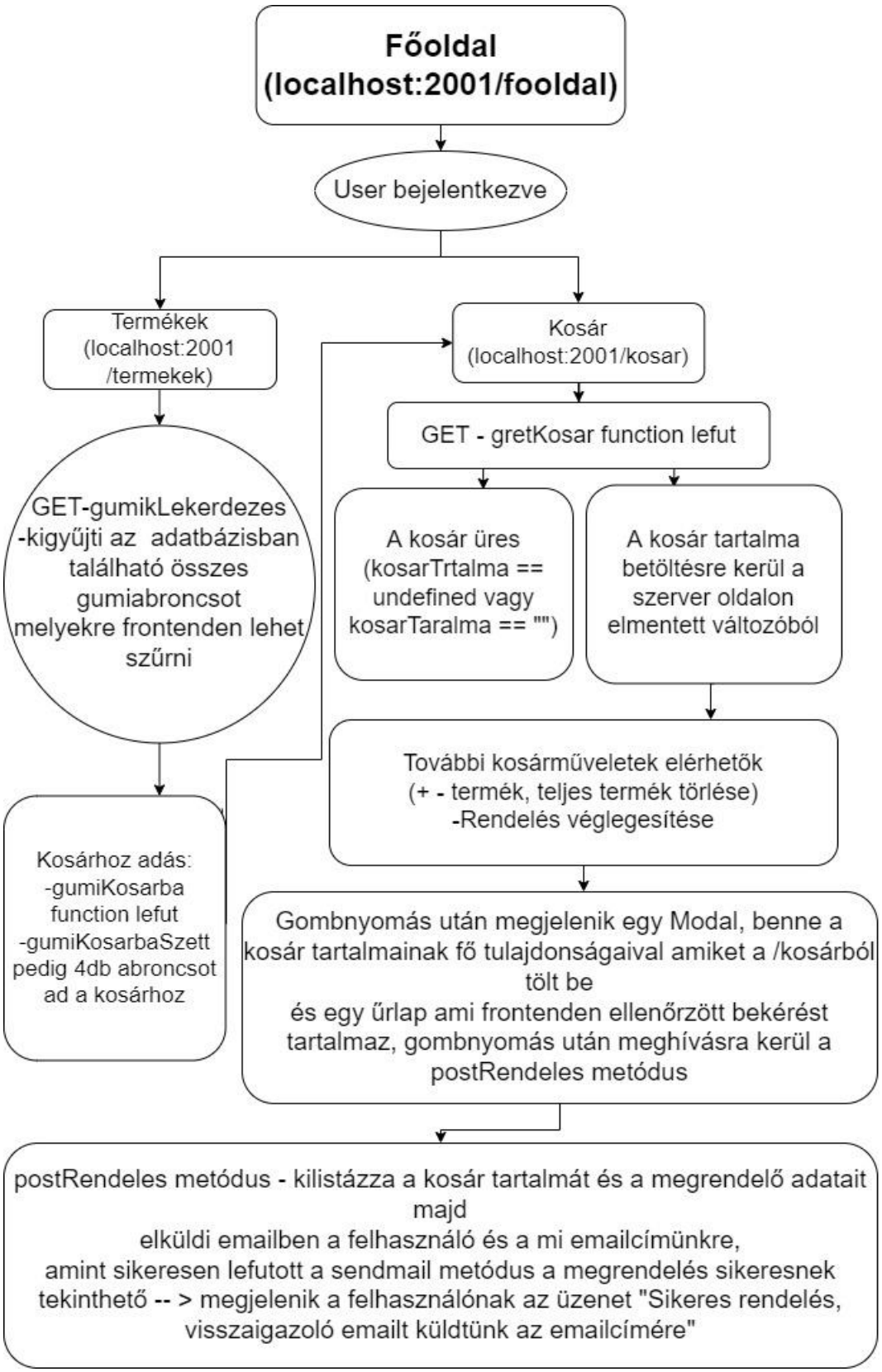
```
lekerd.prototype.rendszamalapjan = function(req, res,
callback){
    rendszam = req.body.rendszam,
    params = [rendszam],
      lekerdquery = 'SELECT auto.Gyarto, auto.Tipus,
info.Evjarat,        info.Allapot,        info.Futottkm,
info.VezetettSzervK, auto.Megbizhatosag, auto.Tipushiba,
info.Okmanyok,      info.Muszakierv,      info.Alvazszam,
info.Gumiabroncs, info.Kepcim, info.Torott FROM info
INNER JOIN auto ON info.Auto_AID = auto.AID WHERE
info.Rendszam = ?';
    mysqlPool.getConnection(function(err, connection){
                connection.query(lekerdquery,   params,
function(err, rows, fields){
            if(rows.length <= 0){
                connection.release();
                console.log(rendszam);
                console.log("Nincs ilyen rendszámú");
                req.session.auto_adatai_rendsz = null;
                req.session.auto_adatai_alvaz = null;
                callback(null,undefined);
            }else{
               req.session.auto_adatai_rendsz = rows[0];
                req.session.auto_adatai_alvaz = null;
                console.log(req.session.auto_adatai_ren
dsz)
                connection.release();
                callback(null,rows[0]);
            }
        });
    });
}
```

Definiálájuk a mySql lekérdezést (a fenti példában rendszám alapján), majd meghívjuk a **/models/db.js**-fájlból importált **getConnection** függvényt, ami ha üres tömbbel tér vissza akkor nincs a feltételnek megfelelő jármű az adatbázisban, ellenkező esetben átadásra kerül az **.auto_adatai_rendsz** tömbben.

Visszatérünk a **/controllers/main.js** fájl „rendzsamlekerdezes” metódusába ami a visszatérési értékeknek megfelelően le rendereli a **fooldal.ejs** fájlt és átadásra kerülnek a megfelelő paraméterek. Ha volt az adatbázisban a kérésnek megfelelő jármű, a **/views/fooldal.ejs** fájlban ejs kód segítségével kiírjuk az autó adatait, pl.:

```
<h1 class="card-title"><%=lekerderedm_rendsz.Gyarto %>
                        <%=lekerderedm_rendsz.Tipus %>
</h1>
     <h5>Évjárat: <%= lekerderedm_rendsz.Evjarat %></h5>
     <h5>Állapot: <%= lekerderedm_rendsz.Allapot %></h5>
     <h4>Futoásteljesítmény:</h4>
     <h5><%= lekerderedm_rendsz.Futottkm %> </h5>
```

4.4.2 Kosár


Főoldal
(localhost:2001/fooldal)
User bejelentkezve
Termékek
(localhost:2001
/termekek)
Kosár
(localhost:2001/kosar)
GET - gretKosar function lefut
GET-gumikLekerdezes
-kigyűjti az adatbázisban
található összes
gumiabroncsot
melyekre frontenden lehet
szűrni
A kosár üres
(kosarTrtalma ==
undefined vagy
kosarTaralma == "")
A kosár tartalma
betöltésre kerül a
szerver oldalon
elmentett változóból
További kosárműveletek elérhetők
(+ - termék, teljes termék törlése)
-Rendelés véglegesítése
Kosárhoz adás:
-gumiKosarba
function lefut
-gumiKosarbaSzett
pedig 4db abroncsot
ad a kosárhoz
Gombnyomás után megjelenik egy Modal, benne a
kosár tartalmainak fő tulajdonságaival amiket a /kosárból
tölt be
és egy űrlap ami frontenden ellenőrzött bekérést
tartalmaz, gombnyomás után meghívásra kerül a
postRendeles metódus
postRendeles metódus - kilistázza a kosár tartalmát és a megrendelő adatait
majd
elküldi emailben a felhasználó és a mi emailcímünkre,
amint sikeresen lefutott a sendmail metódus a megrendelés sikeresnek
tekinthető -- > megjelenik a felhasználónak az üzenet "Sikeres rendelés,
visszaigazoló emailt küldtünk az emailcímére"

4.4.3 Termékek kilistázása

Az adatbázisban szereplő gumiabroncsokat egy SQL lekérdezést végrehajtó függvény segítségével kiírjuk. Ezeket frontenden szűrni lehet.

4.4.4 Termékek kosárhoz adása

Ha a felhasználó kosárhoz ad egy vagy több terméket, akkor a következő metódus fut le:

/controllers/main.js

```
//Gumik kosárba helyezése
exports.gumiKosarba = (req, res, next) => {
    function kosarhozad() {
        termek = req.body.termek_id;
        if (req.session.user.kosar == undefined) {
            req.session.user.kosar = [];
            var kosar = { termek_id: termek, qty: 1 };
            req.session.user.kosar.push(kosar);
        } else {
            var contains = false;
            for (let termek of req.session.user.kosar){
                if (termek.termek_id ==
                    req.body.termek_id) {
                    termek.qty += 1;
                    var contains = true;}
            }
         if (contains == false){
         req.session.user.kosar.push({
         termek_id:termek, qty: 1});}}
        res.redirect("/termekek");
     }
    setTimeout(kosarhozad, 950);
};
```

A **„kosarhozad”** függvény ellenőrzi, hogy létezik-e már a jelenleg bejelentkezett felhasználó számára egy globális **„session.user.kosar”** változó. Ha nem, akkor létrehozzuk és eltároljuk a termék id-ját ahol a függvény meghívásra került, valamint a mennyiségét 1-re állítjuk. Ha már van a kosárban ilyen azonosítóval rendelkező termék, akkor csak a mennyiségét növeljük. – Később ezt töltjük be a **/kosár** oldalon.

4.5 Tesztelés

Teszteléshez a **Jest tester-t** használtuk a következő módon:

/public/test/server.test.js

```
const login = require("../../models/users/login.js");
const req = require("express/lib/request");

test("login-jo", () => {
    var req = [];
    req.body = [];
    req.body.email = "proba@proba.com";
    req.body.jelszo = "Proba1122";
    req.session = [];
    var res = [];
    var rows = [];
    login.loginUser(req, res, function(err, data) {
        expect(err).toBe(null)
    })
    req.session.user = null;
});
```

A kódban látott felhasználó már regisztrálva van, így ezekkel a paraméterekkel meghívott **loginUser** metódus hiba nélkül fut le.

A konzolba beírt **npm run test** parancs után a teszt lefut és a következő üzenetet látjuk:

```
PS C:\Vizsgaremek\OKJ_vizsgaremek> npm run test

> readme.md@1.0.0 test
> jest

  console.log
    6e5a4db2259e8a7779d9679ccea9511e

      at loginUser (models/users/login.js:14:13)

  console.log
    [ 'proba@proba.com', '6e5a4db2259e8a7779d9679ccea9511e', 0 ]

      at loginUser (models/users/login.js:17:17)

 PASS  public/test/server.test.js
  √ login-jo (29 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.569 s, estimated 1 s
Ran all test suites.
```

A teszt sikeresen lefutott és a console.log-ban látható, hogy a **loginUser** metódus lefutott és az md5 titkosítást is elvégezte az általunk megadott jelszón.

Ha a tesztet nem regisztrált felhasználóval futtatjuk le, akkor errort fogunk kapni, tehát a tesztet is ennek megfelelően kell megírni:

```
expect(err).toBe(true)
```

4.5.1 Regisztráció tesztelése

```
test("register-jo", () => {
    var req = [];
     req.body = [];
     req.body.felhasznalonev = "test";
     req.body.email = "test@test.com";
     req.body.jelszo = "Test11222";
     req.body.password_again = "Test11222";
     req.session = [];
     var res = [];
     var res = function(){};
     res.render = function(){};
     register.registerUser(req, res, function(err, data)
{    expect(err).toBe(null)
    })
});
```

A paraméterekben átadott felhasználó még nincs regisztrálva. A teszt sikeresen lefut, tehát nem hibával tér vissza a **registerUser** metódus, valamint az adatbázisban is megjelenik az új felhasználó, és a titkosított jelszava:

	FID	Felhasználónév	Jelszó	Email
Törlés	13	Proba1122	6e5a4db2259e8a7779d9679ccea9511e	proba@proba.com
Törlés	17	test	2b1f0ac358080bdcf0589dc1f8d75054	test@test.com

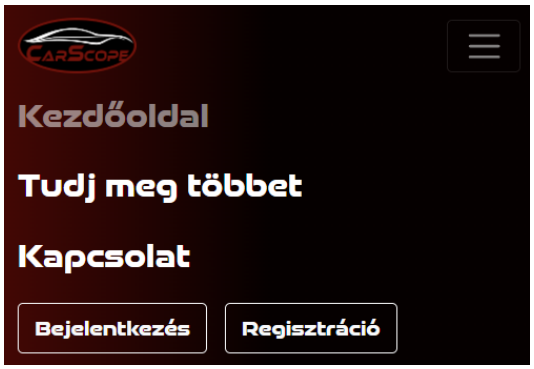
Rossz adatok esetén a tesztelés szintén lefut amennyiben az **err** paraméter **true**-ra van állítva, így értelemszerűen az adatbázisban nem lesz rögzítve a felhasználó.

5 Frontend/Design dokumentáció

A weboldal html5, a weblapok tartalmának és megjelenésének leírására szolgáló szabványos programozási nyelven íródott. Az oldalak formázása és kinézete .css fájlokkal és a bootstrap különböző osztályaival valósult meg.

Különböző megjelenítésekhez és validálásokhoz pedig a vue.js keretrendszert alkalmaztuk. A public/js mappában találhatók azok a fájlok melyek ezeket a metódusokat és függvényeket tartalmazzák.

Az oldal reszponzivitásához bizonyos helyeken bootstrapet használtunk, például a navigációs sávnál is.



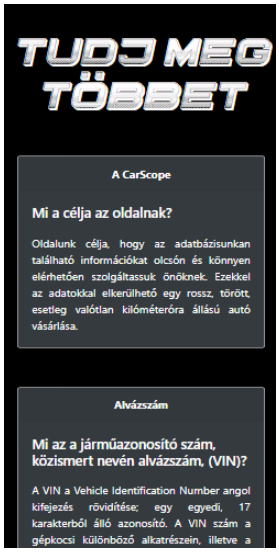
5.1 Mobilnézet

A Tudj meg többet/termékek felületen a kártyák elhelyezése is a bootstrap columns osztályjelölőjével lett formázva. Ennek a segítségével különböző felbontásokra be lehet állítani, hogyan ossza fel az adott elemeket az adott képernyőméreten. Összesen 12 oszlopra van felosztva a képernyő és különböző képernyő méreteken meglehet adni, hogy egy adott elem hány oszlopot foglalhat el ebből a 12 oszlopból.

```
<div class="col-12 col-md-6 col-lg-4 col-xl-3 col-xxl-2">
```

A mobilnézetnél az addig használt „inline” stílusok átlettek alakítva vertikális irányú stílusokká, azzal a céllal, hogy a felhasználó számára mobiltelefonon is könnyen olvashatóak legyenek a szövegek, valamint a gombok, linkek és lenyíló listák kezelése is sokkal kényelmesebb így mobilokon.

Bizonyos megoldások éppen ezért nagyobb felbontásokon elérhetőek csak, mint például a kalkulátor oldalon található 180°-kal elforduló kép.



Mobilnézeten és egyéb felbontásokon a megfelelő megjelenítéshez a .css-fájlokban a @media („felbontás”)-al formáztuk. Ezzel megadhatunk egy médialekérdezést és egy CSS-blokkot, amelyek csak akkor vonatkoznak a dokumentumra, ha a médialekérdezés megadott felbontása megegyezik az oldal pillanatnyi felbontásával, amelyen a tartalmat használják. Paraméternek egy minimum és maximum értéket is meg lehet adni egyszerre.

```
@media (min-width: 941px) and (max-width: 1200px) {
   .fizetesgomb {
      margin-left: 30px;
   }
}
```

A kosárnál a fizetés véglegesítése és az adatkezelési tájékoztató is a bootstrap modal ablakjával lett megvalósítva. A véglegesítéskor a felhasználó által megadott adatok megfelelését az input mezőben elhelyezett „pattern” is ellenőrzi. Ha nem megfelelő a beírt adat, a mező színe pirosra vált.

```
pattern="[ a-zA-ZÀ-ÿ\u00f1\u00d1\u0171\u0170\u0150 ^-,]*"
```

5.2 Főoldal/.js fájlok

A főoldalon a kettő lenyíló listát egy „watcher” vue.js szolgáltatás ellenőrzi, ami csak abban az esetben futtatja a hozzá tartozó függvényt, ha a figyelt elem tulajdonsága megváltozik.

```
watch: {},
```

Jelen esetben a „select” mezőbe elhelyezett „v-modell” direktíva segítségével tudjuk a lenyíló lista választott mezőjét átadni a figyelő függvénynek, valamint biztosítani a két irányú adatközlést.

```
<select v-model="selected" id="cusSelectbox" name="options">
```

A már előre deklarált változók itt a feltételnek megfelelően értéket kapnak, amit utána a vue.js „v-if” direktívájával megvizsgálunk.

```
selected(value)
    {
      if(value == "R")
      {
        this.valasztott = "R";
        this.country_valasztott = "";
      }
      if(value == "A")
```

```
    {
      this.valasztott = "A";
      this.country_valasztott = "";

    }
  },
```

Ennek megfelelően jelennek meg vagy tűnnek el például az országok zászlói, vagy jelenik meg egy újabb lenyíló lista.

```
<div v-if="valasztott == 'R' && country_valasztott != "" class="info">
```

A „v-if” direktíva egy feltétellel rendelkező elem megjelenítési, CSS tulajdonságának átváltására szolgál. Ha a feltétel igaz, akkor láthatóvá teszi, máskülönben láthatatlanná változtatja azt.

A vizsgálni kívánt tartományra hivatkozni kell a „mount” paranccsal, amit id-hez, class-hoz vagy html elemhez tudunk csatolni.

```
app.mount('#cars')
```

5.3 Termékek/rendelés

A termékek felületen egy termék hozzáadásakor, egy bootstrap modal jelenik meg, melyre egy automata bezáró függvény van meghívva, amit a „setTimeout” függvény segítségével időzíthetünk.

```
function timer(obj){
  setTimeout(function() {
    $('#Modal').modal('hide');
  }, 800);
};
```

A rendelés leadásakor pedig a felugró ablakon a bekért mezők értékét a megadott „pattern”-en kívül a rendelés.js fájlban található függvények is ellenőrzik.

```
watch: {
   postalcode(value) {
    if (!/\D/.test(value) == true && value.length == 4 &&  value.replace('
','') != '') {
      this.irsz_validate = true;
     } else {
      this.irsz_validate = false;
     }
   },
},
```

Itt szintén a „watcher” figyelő segítségével nézzük meg a bevitt mező értékét, valamint, hogy az megfelelően lett-e kitöltve.

A kosárnál a rendelés véglegesítésénél az általános szerződési feltételek ablak megnyitása után, ha az „Elfogad” gombra kattintunk, akkor a rendelés „modal”-on levő „checkbox” automatikusan el lesz fogadva a „gombelfogad” metódus hatására.

```
methods: {
   gombelfogad() {
    this.isChecked = 'checked'
    this.afsz_validate = true;
   }
  },
```

6 CarScope-MyAdmin Dokumentáció

6.1 A program célja

A programot a projektünkben használt adatbázisban történő alapvető műveletek elvégzésére készítettük és használjuk. Segítségével SQL parancsok ismerete nélkül tud a felhasználó/admin adatokat megjeleníteni, módosítani, törölni, illetve bővíteni is képes az adatbázist új rekorddal. Ehhez az alkalmazás az Oracle MySql.Data csomagot használja.

6.2 A program felépítése

Az alkalmazás az adatbázisunk 3 táblájának kezelésére alkalmas. Minden táblához készíttetünk egy osztályt, amelyben megtalálhatóak az adott modelhez tartozó adatbáziskezelő függvények. A táblák oszlopait egy-egy privát tulajdonság jelöli:

```
class AutoModel
    {
        private int _aId;

        public int aId
        {
            get { return _aId; }
            set { _aId = value; }
        }

        private string _gyarto;

        public string gyarto
        {
            get { return _gyarto; }
            set { _gyarto = value; }
        }

        private string _tipus;
```

	#	Név	Típus
☐	1	AID	int(10)
☐	2	Gyarto	varchar(20)
☐	3	Tipus	varchar(20)
☐	4	Megbizhatosag	int(11)
☐	5	Tipushiba	varchar(100)

```
        public string tipus
        {
            get { return _tipus; }
            set { _tipus = value; }
        }

        private int _megbizhatosag;

        public int megbizhatosag
        {
            get { return _megbizhatosag; }
            set { _megbizhatosag = value; }
        }

        private string _tipusHiba;

        public string tipusHiba
        {
            get { return _tipusHiba; }
            set { _tipusHiba = value; }
        }
```

Az adatbázisból MySqlDataReader segítségével kiolvassuk az adatokat, majd ezeket a tulajdonság nevének megfelelően tároljuk el. Valamennyi model osztály tartalmaz ezen felül négy függvényt, amely a négy Sql alapművelet (select, update, insert, delete) elvégzésére szolgál.

```
public static ObservableCollection<AutoModel> select()
{
     var lista = new ObservableCollection<AutoModel>();

using (var con = new
MySqlConnection(ConfigurationManager.ConnectionStrings[
"connectionString"].ConnectionString))
         {
           con.Open();
           var sql = "SELECT * FROM auto";
           using (var cmd = new MySqlCommand(sql, con))
         {
           using (var reader = cmd.ExecuteReader())
                 {
                     while (reader.Read())
                   {
```

```
                        lista.Add(new AutoModel(reader));
                    }
                }
            }
        }
            return lista;
}
```

A **select** függvény az adatbázishoz való kapcsolódás után soronként kiolvassa a benne található rekordokat, amiket példányosítva egy listában tárol el.

```
public static void update(int id, string gyarto, string
tipus, int megbizhatosag, string tipushiba)
        {

using          (var          con          =          new
MySqlConnection(ConfigurationManager.ConnectionStrings[
"connectionString"].ConnectionString))
{
 con.Open();
 var sql = "UPDATE auto SET AID = @id, Gyarto = @gyarto,
Tipus = @tipus, Megbizhatosag = @megbizhatosag, Tipushiba
= @tipushiba WHERE AID = @id";
using (var cmd = new MySqlCommand(sql, con))
 {
  cmd.Parameters.AddWithValue("@id", id);
  cmd.Parameters.AddWithValue("@gyarto", gyarto);
  cmd.Parameters.AddWithValue("@tipus", tipus);
   cmd.Parameters.AddWithValue("@megbizhatosag",megbizha
tosag);
  cmd.Parameters.AddWithValue("@tipushiba",tipushiba);

  cmd.ExecuteNonQuery();
  }
 }
}
```

Az **update** függvény a meghívásnál kapott id, jelenesetben autó azonosítójának sorában frissíti az értékeket.

```
public static void insert(string gyarto, string tipus,
int megbizhatosag, string tipushiba)
        {
using (var con = new
MySqlConnection(ConfigurationManager.ConnectionStrings[
"connectionString"].ConnectionString))
{
 con.Open();
 var sql = "INSERT INTO `auto`(`Gyarto`,`Tipus`,
`Megbizhatosag`," + " `Tipushiba`) VALUES (@gyarto,
@tipus, @megbizhatosag, @tipushiba)";
using (var cmd = new MySqlCommand(sql, con))
{
  cmd.Parameters.AddWithValue("@gyarto", gyarto);
  cmd.Parameters.AddWithValue("@tipus", tipus);
          cmd.Parameters.AddWithValue("@megbizhatosag",
megbizhatosag);
  cmd.Parameters.AddWithValue("@tipushiba", tipushiba);

  cmd.ExecuteNonQuery();
  }
 }
}
```

Az **insert** függvénnyel új rekordot tudunk felvinni az adatbázisba, a hozzáadott rekordok azonosítója automatikusan növekszik.

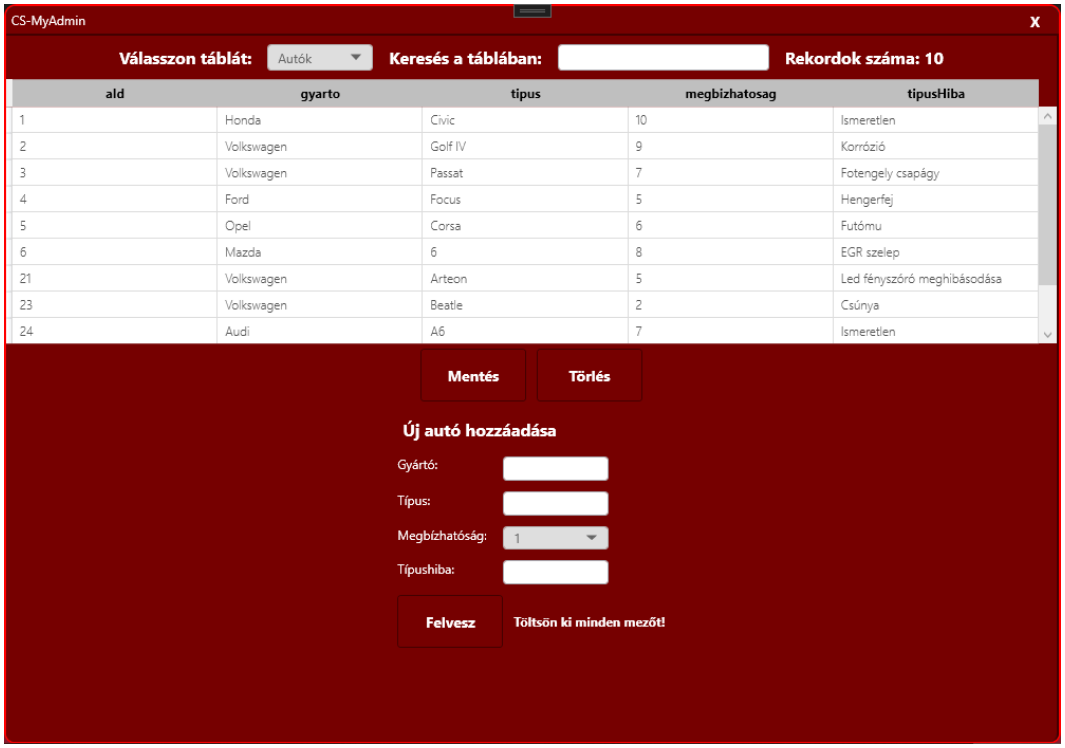
```
public static void delete(int id)
{
                        using (var con = new
MySqlConnection(ConfigurationManager.ConnectionStrings[
"connectionString"].ConnectionString))
      {
        con.Open();
        var sql = "DELETE FROM `auto` WHERE AID = @id";
        using (var cmd = new MySqlCommand(sql, con))
      {
        cmd.Parameters.AddWithValue("@id", id);

        cmd.ExecuteNonQuery();
      }
   }
}
```

A **delete** függvény értelemszerűen törli a kapott id-hoz tartozó rekordot az adatbázisból, és az azzal megegyező idegen kulccsal rendelkező rekordokat a többi táblából.

6.3 Indítás után

A program indulás után beolvassa az adatbázisban található adatokat, majd táblánként azokat egy külön listában tárolja el.



Az ablak felső részében található egy táblázat, amelyben alapértelmezetten az autok lista elemeit láthatjuk. Az efölött található lenyíló listában választhatjuk ki a megjeleníteni kívánt táblát. Váltás esetén a táblázat adatforrása frissül, és az alatta található űrlap is megváltozik. Ehhez **if** függvényt használunk.

Az ablak tetején továbbá találunk egy keresésre szolgáló beviteli mezőt, amely a kiválasztott lista elemei közötti keresésre szolgál. Továbbá itt láthatjuk az aktuálisan megjelenített sorok számát is.

```
private void cb_databases_SelectionChanged(object
sender, SelectionChangedEventArgs e)
        {
if (cb_databases.SelectedItem.ToString()=="Autók")
 {
      autok = AutoModel.select();
      DG_adatok.ItemsSource = autok;
      TB_searchbar.Text = "";

      SP_infokInsert.Visibility = Visibility.Collapsed;
      SP_gumikInsert.Visibility = Visibility.Collapsed;
      SP_autokInsert.Visibility = Visibility.Visible;
 }
else                                                if
(cb_databases.SelectedItem.ToString()=="Gumiabroncsok")
{
 gumik = GumiModel.select();
 DG_adatok.ItemsSource = gumik;
 TB_searchbar.Text = "";
 SP_infokInsert.Visibility = Visibility.Collapsed;
 SP_autokInsert.Visibility = Visibility.Collapsed;
 SP_gumikInsert.Visibility = Visibility.Visible;
}
else if (cb_databases.SelectedItem.ToString()=="Info")
{
 infok = InfoModel.select();
 DG_adatok.ItemsSource = infok;
 TB_searchbar.Text = "";

 CB_autoAzon.Items.Clear();
 foreach (var item in autok)
{
CB_autoAzon.Items.Add(item.aId + ":" + " " + item.gyarto
+ " " + item.tipus);
}
 CB_autoAzon.SelectedIndex = 0;
 SP_autokInsert.Visibility = Visibility.Collapsed;
 SP_gumikInsert.Visibility = Visibility.Collapsed;
 SP_infokInsert.Visibility = Visibility.Visible;
}
 LBL_recordCount.Content = "Rekordok száma: " +
DG_adatok.Items.Count.ToString();
}
```

Táblaváltásnál a táblázat (**Datagrid DG_adatok**) adatforrása a kiválasztott modelhez tartozó lista lesz, mely minden váltásnál frissíti a tartalmát. Törlődik a keresőmezőbe beírt szöveg, valamint a megfelelő űrlap kerül megjelenítésre. Az űrlapok elemeit Stackpanelbe csoportosítottuk.
Végül frissül a megjelenített sorok számát mutató Label.

6.4 Keresés

```
else if
(cb_databases.SelectedItem.ToString()=="Gumiabroncsok")
{
if (TB_searchbar.Text != "")
{
DG_adatok.IsReadOnly=true;
var      filteredList      =      gumik.Where(x      =>
x.gyarto.ToLower().StartsWith(TB_searchbar.Text.ToLower
())
                                                        ||
x.evszak.ToLower().StartsWith(TB_searchbar.Text.ToLower
()));

DG_adatok.ItemsSource = filteredList;
BTN_Delete.IsEnabled = false;
BTN_Save.IsEnabled = false;
}
else
{
DG_adatok.IsReadOnly = false;
DG_adatok.ItemsSource = gumik;
BTN_Delete.IsEnabled = true;
BTN_Save.IsEnabled = true;
}
}
```

Ha a keresőmezőbe szöveget gépelünk, eseménykezelő segítségével a program meghatározza, hogy mely adatbázisban kell keresést végrehajtani, majd a megfelelő listából egy másik listába kigyűjti a beírt szöveggel kezdődő sorokat.

Így a beviteli mezőbe beírt szöveg alapján kereshetünk egy adott rekordot vagy rekordokat az adatbázisban:

Volkswagen gyártmányú autók

- autót gyártó és típus szerint,
- autó infó-t rendszám és alvázszám szerint,
- gumiabroncsot pedig gyártó és évszak alapján.

A **Datagrid** adatforrása ezt követően a szűrt lista lesz, egészen addig, amíg ki nem töröljük a keresőmezőbe beírt szöveget.

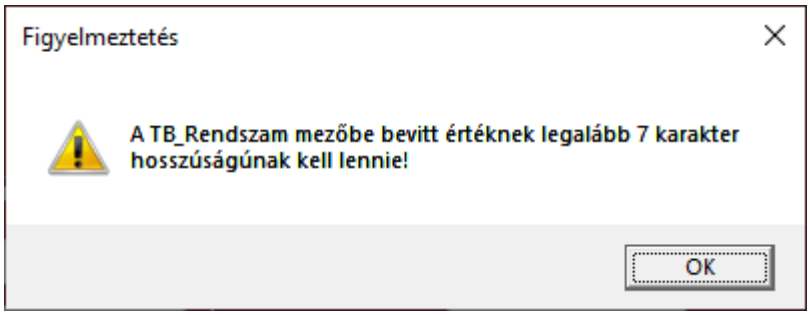
6.5 Módosítás és törlés

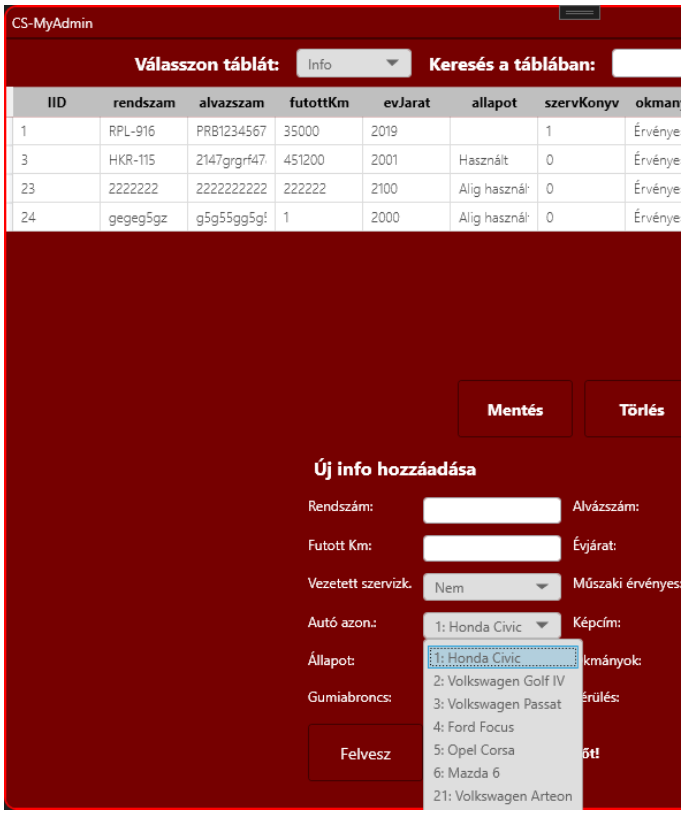
A Datagrid és az űrlap között található egy Mentés és egy Törlés gomb. Ha egy mezőre duplán kattintunk, tartalma módosíthatóvá válik. A Mentés gombra kattintva az adatok frissülnek a listában, és ezáltal a táblázatban is. A Törlés gombbal törölhetjük a kiválasztott rekordot az adatbázisból.

Annak eldöntésére, hogy mely táblában kerüljön végrehajtásra a módosítás/törlés, az adatforrás kiválasztását végzővel megegyező szerkezetű if függvényt használunk.

6.6 Új adat felvitele

Az űrlap Textboxait kötelező kitölteni, a lenyíló listák alapértelmezetten a 0. index-szel rendelkező opciót jelenítik meg, a dátumválasztó alapértelmezett értéke a mindenkori mai dátum. Ha egy mezőt üresen hagyunk, a Felvesz gombra kattintás után egy hibaüzenet jelenik meg, és nem történik meg az új adat felvitele. A rendszám, alvázszám, Futott km és évjárat mezőkbe csak számot lehet begépelni, a rendszámnak 7-8, az alvázszámnak 17, az évjáratnak 4 karakter hosszúnak kell lennie. Ha a felhasználó úgy vált mezőt, hogy valamely feltétel nem teljesül, annak megfelelő hibaüzenet kerül megjelenítésre, és a hibás beviteli mező értéke törlődik, megakadályozva a felvitelt.





Kiválaszthatjuk, melyik autótípushoz szeretnénk új járműadatokat hozzáadni.

Új info hozzáadásánál az Autó azon. lenyíló listában ki kell választanunk, hogy milyen típusú autóhoz szeretnénk adatokat felvinni. Ehhez a program betölti az adatbázisban található autók id-ját, gyártóját ás típusát, előbbit ':'- al elválasztva utóbbi kettőtől. Inzertáláskor a kiválasztott listaelemből egy kételemű tömböt készítünk, melyben a sor ':' előtti tartalma, azaz az id kerül elválasztásra a gyártótól és a típustól, ez lesz a tömb (splittedText) 0. eleme, amit **int** típussá kovertálva fel tudunk vinni az adatbázisba idegen kulcsként(pirossal).

```
string[]                    splittedText                    =
CB_autoAzon.SelectedItem.ToString().Split(':');
if (TB_Rendszam.Text !="" && TB_Alvazszam.Text !="" &&
TB_FutottKm.Text != "")
{
InfoModel.insert(TB_Rendszam.Text,    TB_Alvazszam.Text,
Convert.ToInt32(TB_FutottKm.Text),
int.Parse(TB_Evjarat.Text),
CB_allapot.SelectedItem.ToString(),
CB_szervkönyv.SelectedIndex,
CB_okmanyok.SelectedItem.ToString(),
DP_muszaki.SelectedDate.Value,
CB_GumiInfo.SelectedItem.ToString(),
Convert.ToInt32(splittedText[0]),TB_kepCim.Text,
CB_Torott.SelectedItem.ToString());
```

```
infok = InfoModel.select();
DG_adatok.ItemsSource = infok;
}
```

6.7 Az űrlapok visszaállítása

Inzertálás után a mezők alapértelmezett értékre való visszaállításához a **ResetSp** függvényt használjuk:

Mivel az űrlapok egy Stackpanelből és az abban levő egymásba ágyazott Stackpanelekből állnak, ezért legkönnyebben függvény segítségével érhetjük el a belső Stackpanelek objektumait. A függvénynek meghíváskor megadjuk, melyik űrlapban végezze el a visszaállítást. Megkeresi az abban található Stackpaneleket, amiket egy listába gyűjt egy foreach függvény segítségével. A kigyűjtött Stackpanelekben megkeressük a gyerekobjektumokat, amik a mi esetünkben már azok a vízszintes orientáltságú stackpanelek lesznek, amikben a beviteli mezők találhatók. Foreach használatával megkeresünk minden Textbox, Combobox, és Datepicker típusú objektumot, és az indításkori állapotra állítjuk vissza őket.

```
static void ResetSP(StackPanel Stackname)
{
 List<Object> spanels = new List<Object>();

 foreach (Object vizsgaltItem in Stackname.Children)
  {
    if (vizsgaltItem.GetType() == typeof(StackPanel))
     {
       spanels.Add(vizsgaltItem);
     }
}
foreach (StackPanel spanel in spanels)
{
foreach (Control item in spanel.Children)
{
  if (item.GetType() == typeof(TextBox))
   {
     ((TextBox)item).Text = string.Empty;
   }
  else if (item.GetType() == typeof(ComboBox))
   {
     ((ComboBox)item).SelectedIndex = 0;
   }
```

```
  else if (item.GetType() == typeof(DatePicker))
   {
     ((DatePicker)item).SelectedDate = DateTime.Now;
   }
  }
 }
}
```

7 Konklúzió

A fejlesztési folyamat során a munkát már a kezdetekkor három részre bontottuk, hogy minél hatékonyabban haladjunk. Az ötletelést közösen végeztük, majd ezután mindenki önállóan tanórákon, illetve otthon dolgozott a rábízott programrésszel. Egy-egy feladat elvégzése után együtt átbeszéltük a munka során felmerülő tapasztalatokat, problémákat és hibákat.

A fejlesztést nehezítette, hogy a program elkészítéséhez szükséges tudás nagy részét fejlesztés közben ismertük és tanultuk meg, legfőképp a tanórákon, így nem rendelkeztünk többéves tapasztalattal a felhasznált programozási nyelvek, struktúrák terén. Ennek ellenére alapötleteink legtöbbjét megvalósítottuk, és sikerült egy számunkra elfogadható minőségű alkalmazást létrehoznunk.

Terveztük még az oldalunkat egy olyan funkcióval bővíteni, mely megjelenítette volna az országban található, élményvezetésre elérhető autókat, illetve tulajdonosuk elérhetőségeit, ám erre időhiány miatt végül nem került sor.

Ha a jövőben fejlesztenénk az alkalmazást, kibővítenénk az elérhető nemzetiségek számát, valamint az egyes autókról megjelenített információ és képek mennyiségét, illetve szakértői véleményeket is fűznénk az egyes autó és gumi típusokhoz. Az admin alkalmazásban fejlesztenénk a validációt, illetve néhány funkciót automatizálnánk (pl., ha magyar rendszámmal rendelkező autóhoz a mainál korábbi dátumot írunk, az okmányok érvényességét mutató mező értéke „Lejárt magyar okmányokkal” értékre vált, így kizárva az egymásnak ellentmondó információk felvitelét az adatbázisba).

8 Telepítés

- Adatbázis feltöltése xampp segítségével a phpMyadmin oldalon (CarScopeDatabase.sql fájl)
- NodeJs telepítése: https://nodejs.org/en/
- Forrás fájlok mappájának megnyitása CMD-ben (vagy cisual studio code-ban), parancsok futtatása:
    - npm install
    - npm start
- Az oldal a localhost:2001-es porton fut
- Tesztelhető járművek (példa):
    - PWJ-532 Alvázszám: BMW001PUS255GHF32
    - HKR-115 Alvázszám: 2147GRGRF47845741
    - RPL-916 Alvázszám: PRB12345678901234
    - KÖNS5000 Alvázszám: LAMBO001FSTFSFS44
- Már regisztrált felhasználó:
    - Proba1122 Email: proba@proba.com Jelszó: Proba1122

9 Fejlesztők

Pusoma Gergő: Backend, adatbázis fejlesztése és dokumentákása – Node.js, MySql, .ejs kódok

Takács Marcell: Frontend, UI fejlesztése és dokumentálás – Grafika, design, Html, Vue.js

Székely Áron: Admin Grafikus alkalmazás fejlesztése és dokumentálása – C# WPF app

Github Commitok az oldal és az admin alkalmazás „branch”-ben (1st-try):

```
pusom@PCPG                                  MINGW64
/c/Vizsgaremek/OKJ_vizsgaremek (1st-try)
$ git shortlog -s
    80  Pusoma Gergő
    73  Székely Áron
   107  Takács Marcell
```

Összes Github commit (1st-try, Adatbázis, Dokumentáció, Terv+Logo)**:**

```
pusom@PCPG                                  MINGW64
/c/Vizsgaremek/OKJ_vizsgaremek
$ git shortlog -s --all
   124  Pusoma Gergő
    86  Székely Áron
   120  Takács Marcell
```

10 Források

https://stackoverflow.com

https://www.w3schools.com/w3css/w3css_web_css.asp

https://www.w3schools.com/css/default.asp

https://freefrontend.com/

https://speckyboy.com/

https://getbootstrap.com/docs/4.0/components/navs/

https://nodejs.org/en/

https://www.w3schools.com/nodejs/default.asp

https://www.w3schools.com/cs/index.php

https://stackoverflow.com/questions/2353818/how-do-i-get-started-with-node-js

https://www.mysql.com/

https://nodemailer.com/about/

https://hu.wikipedia.org/wiki/MD5

https://www.npmjs.com/package/md5