Szegedi Szakképzési Centrum Vasvári Pál Gazdasági és Informatikai Szakgimnáziuma

Az 54 213 05 számú Szoftverfejlesztő szakképesítés záródolgozata

CAR DOCKET

Készítette:

Marton Zoltán István

Szeged

2020

Tartalom

BEVEZETÉS

Személyiségemből adódóan nem szeretem a rendetlenséget az életemben és ezzel sokan másik is így vannak, valamint nagy rajongója vagyok az autóknak így adott volt, hogy egy autókkal kapcsolatos adatbáziskezelőt fogok elkészíteni.

A programom összetett megoldást kínál olyan szervezetek számára ahol nagyobb mennyiségű flotta van, de jelenleg Excel táblát vagy egyéb hasonló offline szoftver használnak a nyilvántartásra. Az ilyen offline programok legnagyobb akadálya pont abból adódik, hogy egyszerre csak egy gépen tudják módosítani, nem biztonságos, macerás a mozgatása, amennyiben nem kezeli a hibákat nagyon könnyű tönkre tenni egy-egy rossz adattal.

1. FEJLESZTŐI DOKUMENTÁCIÓ

1.1 ADATBÁZIS TERVEZÉS

A fejlesztés egyértelműen az adatbázis megtervezésével indult. Mivel már tisztában voltam azzal mit kell tudnia egy ilyen programnak ezért bátran kezdtem bele a tervezésbe. Egyedi azonosításhoz minden esetben a MySQL auto increment funkcióját használtam fel. Azzal kezdtem a tervezést, hogy feltérképeztem melyek azok az adatok, amelyeket érdemes nyilvántartani egy autóról és ezeket az adatokat milyen típusú változóban kellene tárolni.

1.táblázat

Autók tábla

Adat	Változó
id (elsődleges kulcs)	int
marka	varchar
tipus	varchar
gyartasi_ev	varchar
vetelar	int
rendszam	varchar
kilometeroraallas	int
alvazszam	varchar
gepkocsi_tipusa	varchar
uzemanyag	varchar
sebessegvalto_tipusa	varchar
tulid (másodlagos kulcs)	int

Az autót tábla tervezésénél figyelnem kellett arra, hogy bár egy-egy autóról nagyon sokféle adatot lehet tárolni, a felhasználónak ne kelljen számára felesleges és a használat szempontjából lényegtelen adatokat felvinni és a későbbiekben kezelni. És persze az adatbázis mérete sem elhanyagolható szempont. Ezeket a szempontokat minden táblánál figyelembe vettem. A megfelelő változók használatára is oda kellett figyelni az autó gyártási éve szándékosan lett szöveges érték, mert a későbbiekben a program működésében erre volt szükségem a megfelelő működéshez. Az autók tábla feltöltését követően következhetett a tulajdonosok tábla megtervezése és feltöltése. A táblában a vételár és a kilóméteróraállás lehet nulla érték a többinél nem engedélyezett.

2.táblázat

Tulajdonosok tábla

Adat	Változó
tulid (elsődleges kulcs)	int
tulajdonos_nev	varchar
tulajdonos_szemelyiigszam	varchar
jogositvany_azon	int
email_cim	varchar
telefonszam	int
cegid (másodlagos kulcs)	int

A tulajdonosokat eltároló táblánál is figyelembe vettem azt, hogy ne tároljunk teljesen felesleges adatokat az autók tulajdonosairól, kizárólag azokat, amelyek szükségesek. Az autók és tulajdonosok tábla feltöltését követően összekapcsoltam őket. Az autók táblában csak a tulajdonos azonosítóját (tulid) tároltam el. Egy autónak csak egy tulajdonosa lehet, de egy tulajdonosnak több autója is. A tulajdonosok táblában egyáltalán nincs engedélyezve nulla érték egyik mezőben sem.

3.táblázat

Cégek tábla

Adat	Változó
cegid (elsődleges kulcs)	int
cegnev	varchar
adoszam	int
varos	varchar
utca	szam
szam	int
ceg_email_cim	varchar

Ennél a táblánál is kellő odafigyelést fordítottam arra, hogy csak a legszükségesebb adatokat tároljam adatbázisban. Ebben a táblában sincs engedélyezve nulla érték egyik mezőben sem. Ebben a táblában már nincsen másodlagos kulcs. Miután feltöltöttem a tulajdonosok és a cégek táblát ezután a cégek táblában azonosításra használt értékkel (cegid) összekötöttem a két táblát.

4.táblázat

Felhasználók tábla

Adat	Változó
id	int
felhasznalonev	varchar
jelszo	varchar
emailcimfelhasznalo	varchar

Ez a tábla egy végtelenül egyszerű felépítéssel rendelkezik. A felhasználókat tárolja, akik rendelkeznek hozzáféréssel a webes és asztali alkalmazáshoz. Későbbiekben írni fogok a felhasználók kezeléséről is. Ebben a táblában szintén nem engedélyezett egyik mezőben sem a nulla, mint érték. A táblában csak kézzel adható hozzá két egyező felhasználónévvel rendelkező egyén. A weboldalon erre nincs lehetőség, mert az ellenőrzés le fut a regisztrációnál arra is, hogy létezik-e már ilyen nevű felhasználó az adatbázisban. Jelenleg a felhasználók tábla és a többi tábla között nincsen összeköttetés. Az email címre a későbbi fejlesztések érdekében is szükség volt, valamint a bejelentkezésnél email címmel is lehetősége van a felhasználónak bejelentkezni.

1.2 WEBOLDALFEJLESZTÉS

A weboldal tervezése során szem előtt kellett tartanom a legfontosabb tulajdonságokat, amelyekkel rendelkezni kell egy ilyen oldalnak. A teljesség igénye nélkül felsorolnék párat. Először is nagyon fontos az átlátható és könnyen tanulható oldal, amelyen a felhasználó otthon érzi magát és nem ül percekig a monitor előtt azon gondolkozva, hogy tulajdonképpen ahhoz, hogy elérjen egy funkciót, mit is kellene megnyomnia, vagy mire kellene kattintania. Másodszor a mai világban nagyon fontos a reszponzivitás, az hogy a weboldal több méretben és telefonon is megfelelően és természetesen ugyanúgy használhatóan jelenjen meg alapfeltétel volt. Többek között emiatt és az egységes dizájn miatt a Bootstrap 4 segítségével terveztem meg az egész oldalt. Legelőször megterveztem a felépítését az oldalnak, hogy nagyjából hogyan is szeretném elhelyezni a menüpontokat, milyen legyen az egységes design. Ezek után kezdtem bele annak a meghatározásába, hogy milyen funkciókat szeretnék a weboldalba implementálni. A Főoldal tetején megtalálható a menü valamint a bejelentkező felület.

1.2.1 REGISZTRÁCIÓ

A regisztráció során, ha a felhasználó helyesen adja meg a kért adatokat, abban az esetben hozzá adódik a rendszerhez az új felhasználó és az ő általa megadott adatok feltöltődnek a regisztrációkor az adatbázis előre megadott és erre a célra létrehozott táblájába a signup.inc nevű PHP állomány segítségével, amely lefut a „Regisztrálok” gomb lenyomására. A regisztrációs oldalt, amelyen a kitöltendő beviteli mezők találhatóak szintén PHP nyelvben írtam meg. A regisztráció csak abban az esetben lehet sikeres, ha a felhasználó a megadott feltételeket betartja, amikor kitölti az űrlapot. Jelen esetben két helyen is vizsgálom a beírt adatokat. Először kliens oldalon vizsgálom a bemeneti adatok helyességét valamint azt, hogy a felhasználó mindent oda írjon, ahova rendeltetésszerűen kell. Ebben az esetben, ha a felhasználó nem helyes adatot ad akkor egy figyelmeztető üzenet jelenik, meg amely tájékoztatja a felhasználót arról, hogy a bevitt adat hibás vagy adott esetben üres a mező. Másodjára szerver oldalon is vizsgálat alá vonom a bevitt adatokat még azelőtt, hogy feltöltődtek volna az adatbázisba. Legelőször az az ellenőrzés fut le, amely azt vizsgálja, hogy a beviteli mezők üresek-e. A felhasználónév esetében RegEx-et használtam annak ellenőrzésére, hogy a felhasználónév megfelel-e a követelményeknek. A bevitt email címet a PHP beépített email ellenőrzésével ellenőriztem le. A jelszó esetén az ellenőrzés azt vizsgálja, hogy a megadott jelszavak megegyeznek-e vagy sem. Amennyiben bármelyik

vizsgálat szerver oldalon hibát ad vissza, abban az esetben header php parancs segítségével visszairányítom a felhasználót a signup nevű php állományra és az url-be kiiratom hogy hol is futott hibára az lefutása alatt valamint kiléptetem a signup.inc nevű php állomány lefutásából egy exit() használatával. Ennek a PHP kódja látható az alábbi képen.

1.sz kép

```php
if (empty($username) || empty($email) || empty($password) || empty($passwordRepeat))
{
    header("Location: ../signup.php?error=emptyfields&uid=".$username."&email=".$email);
    exit();
}
else if (!filter_var($email, FILTER_VALIDATE_EMAIL) && !preg_match("/^[a-zA-Z0-9]*$/", $username))
{
    header("Location: ../signup.php?error=invalidmailuid");
    exit();
}
else if (!filter_var($email, FILTER_VALIDATE_EMAIL))
{
    header("Location: ../signup.php?error=invalidmail&uid=".$username);
    exit();
}
else if (!preg_match("/^[a-zA-Z0-9]*$/", $username))
{
    header("Location: ../signup.php?error=invaliduid&mail=".$email);
    exit();
}
else if ($password !== $passwordRepeat)
{
    header("Location: ../signup.php?error=passwordcheck&uid=".$username."&mail=".$email);
    exit();
}
```

Hibás adatok bevitelekor a felhasználó több féle hiba üzenetet is kaphat, attól függően, hogy milyen hiba adódott. Többek között emiatt is szükség van a headerrel való átirányításokra, mert az átirányított url tartalmát vizsgálom a signup.php-ban és ezért mindig a helyes hibaüzenet jelenik meg. A $_GET-el vizsgálom meg az átirányított url kérdőjel utáni tartalmát és ezalapján lefut a helyes echo és megjelenik a helyzetnek megfelelő hibaüzenet. Az alábbi kód részletben kettő darab ilyen kiírás látható.

3.sz kép

```php
if (isset($_GET['error'])) {
    if ($_GET['error'] == "emptyfields") {
        echo '<div id="alert" class="alert alert-danger alert-dismissible fade show">
                        <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
                        <p class"signuperror">Töltsön ki minden mezőt!</p>
                        </div>';
    } else if ($_GET['error'] == "invalidmailuid") {
        echo '<div id="alert" class="alert alert-danger alert-dismissible">
                        <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
                        <p class"signuperror">Helytelen felhasználó és email cím!</p>
                        </div>';
```

Amennyiben a regisztráció sikeres egy boostrapes toast ablak jelenik meg közvetlen a bejelentkezés után. Ezért a megjelenítésért felelős kódot itt láthatják.

4.sz kép

```
if (isset($_GET['signup'])) {
    if ($_GET['signup'] == "success") {
        echo '<div style="position: absolute; top: 56px; right: 40px;">
                <div class="toast" data-delay="5000" role="alert" aria-live="assertive" aria-atomic="true">
                <div class="toast-header">
                <img src="img/Check.png" style="width: 17px;" class="rounded mr-2" alt="erroricon">
                <strong class="mr-auto">Hiba</strong>
                <small class="text-muted">Épp most</small>
                <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
                </div>
                <div class="toast-body">
                    Sikeres regisztráció!
                </div>
                </div>
                </div>';
    }
}
```

A felhasználónevet RegEx-el ellenőriztem le, csak betűt és számot tartalmazhat különben hibát fog dobni a felhasználónak. Email címnél a fentebb említett beépített ellenőrző metódust használom.

1.2.2 BEJELENTKEZÉS

Sikeres bejelentkezés utána felhasználónak felóldódnak azok a funkciók, amelyek megtekintéséhez rendelkezik a kellő jogosultsággal. Kettő darab felhasználói szintet hoztam létre a weboldalon jelenleg ennyi érhető el. Ennek a két szintnek az elkülönítését úgy oldottam meg, hogy a weboldal kettő fajta header tud megjeleníteni belépés után. Amennyiben a felhasználó userid egyenlő adminnal, azaz a felhasználó neve admin (korlátozva van, mivel nem jöhet létre kettő ugyanolyan nevű profil ezért maximum manuálisan az adatbázis szerkesztésével lehet több admin nevű felhasználó, a weboldal nem enged többet regisztrálni). Ezt az alábbi kóddal oldottam meg.

5.sz kép

```php
if (isset($_SESSION['userid'])) {
    $userid = $_SESSION['userid'];
    $sql = "SELECT felhasznalonev FROM felhasznalok WHERE id = '$userid'";
    $res = $connection -> query($sql);
    $row = $res -> fetch_row();
    if ($row[0] == "admin")
    {
        require 'headerlogoutadmin.php';
    }
    else {
    require 'headerlogoutnotadmin.php';
    }
}
```

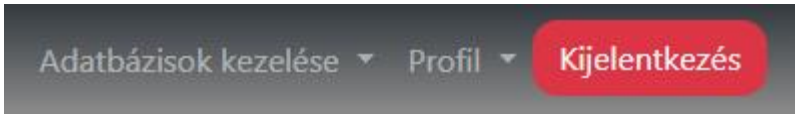
A fenti kódrészlet először is megnézi, hogy van e userid a jelenlegi session-ben, amennyiben van, az azt jelenti, hogy biztos, hogy a felhasználó bejelentkezett. Következő lényeges lépés a lekérés maga, amely lekéri a belépett felhasználó nevét.

Amennyiben a név egyenlő adminnal, abban az esetben az a headerlogoutadmin.php tölt be. Ezen alapértelmezetten megjelenítésre kerül a felhasználók kezelése fül, amelyen keresztül elérhetjük a már regisztrált felhasználók adatait.

6.sz kép

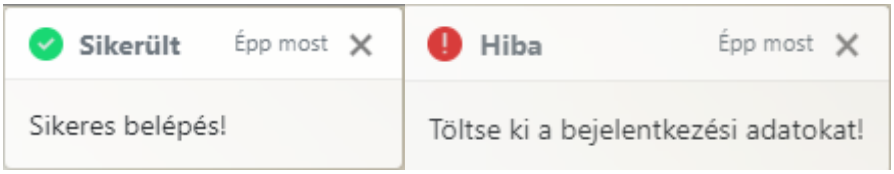
Amennyiben a felhasználó neve nem egyenlő azzal, hogy admin abban az esetben a másodlagos headerlogoutnotadmin.php fog betölteni.

7.sz kép



Sikeres belépés esetén a felhasználónak megjelenik egy kis üzenet, ami arról tájékoztatja, hogy a belépés sikeres volt vagy sikertelen.

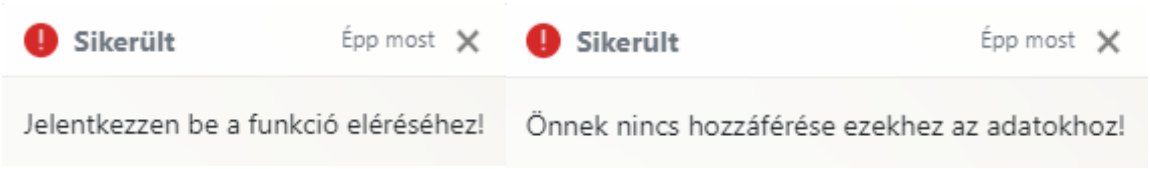
8.sz kép



Annak megelőzése érdekében, hogy ténylegesen ne férhessen hozzá senki olyan adatokhoz, amelyekhez nincsen jogosultsága. Minden olyan php állomány elejére raktam ellenőrzést, amelynél akadály lehet, ha a felhasználó beírva az url helyére lefuttathatja és ezzel olyan adatokhoz fér hozzá, amelyekhez nem kellene. Ebből kifolyólag több szinten is ellenőrzi a weboldal, hogy ki az, aki hozzá szeretne férni. A felhasználók kezeléséért felelős php állományban először az a szűrés fut le, hogy létezik ezen a $_SESSION [’userid’], amennyiben létezik, abban az esetben van bejelentkezett felhasználó az oldalon. De ez még nem elég a felhasználók kezelése fül eléréséhez, ezért a következő ellenőrzés arra fut le, amelyet fent is említettem, hogy a felhasználó neve egyenlő-e adminnal. Amennyiben igen, abban az esetben hozzá fér a felhasználók adataihoz és kezelni is tudja őket.

A felhasználó kettő féle hibát kaphat abban az esetben, ha nincs jogosultsága, azaz nincs bejelentkezve vagy nem ő az admin felhasználó.

9.sz kép



A bal oldali üzenetet akkor kaphatja egy felhasználó ha egyáltalán nincs bejelentkezve de ő megpróbálkozik a usercontrol.php elérésével, ebben az esetben már a legelső if hamisat ad vissza és else ágon átirányít az index.php-ra. A jobb oldali üzenetet a felhasználó abban az esetben kaphatja, ha rendelkezik felhasználóval aki be tud lépni az oldalra, viszont nem ő az adminisztrátor. Ellenben megpróbálja az url-be való beírással futtatni a usercontrol.php-t. Ilyenkor fel szólítja a weboldal hogy nincs hozzáférése ezekhez az adatokhoz.

1.2.3 PROFIL ADATOK SZERKESZTÉSE

Sikeres bejelentkezés után a menüben talál a felhasználó egy profil feliratú dropdown menü-t. Erre rányomva lenyílik, a menü kettő darab funkcióval megjelenik. Az első lehetőség a profil szerkesztése, a felhasználó ha erre kattint abban az esetben betöltésre kerül a useredit.php nevű php fájl. Ez esetben is használtam a fentebb már említett szűrési eljárást. Ennek az oldalnak a megjelenése előtt is lefut egy if, amely ellenőrzi hogy van e userid a $_SESSION-ben, amennyiben nincs, abban az esteben átirányít az index.php-ra és a megfelelő hibaüzenetet írja ki. Amennyiben rendelkezik, tehát van userid akkor elindul a megjelenített adatok lekéréséért felelős php, amely lekéri az adatbázisból a belépett felhasználó felhasználónevét, jelszavát és az email címét. Ezeket az adatokat betölti a php egy asszociatív tömbbe, ahonnan ki veszi külön-külön változókba. Valamint utána a lentebb található módszerrel bele tölti az űrlapon megtalálható beviteli mezőbe azt az oda szükséges adatokat. Ezért felelős kód látható az alábbiakban. A kódban megtalálható a required attribútum. Ez egy beépített attribútum, amely azért felelős, hogy ne maradhasson üresen egy mező sem, valamint az email cím helyére email cím kerüljön és ne más adat.

10.sz kép

```php
if (isset($_SESSION['userid'])) {
    $userid = $_SESSION['userid'];
    $sql = "SELECT felhasznalonev, jelszo, emailcimfelhasznalo FROM felhasznalok WHERE id = '$userid'";
    $res = $connection->query($sql);
    $row = $res->fetch_assoc();

    $felhasznev = $row['felhasznalonev'];
    $pwd = $row['jelszo'];
    $email = $row['emailcimfelhasznalo'];
```

11.sz kép

```php
<div class="form-group">
    <label class="control-label col-sm-2" for="felhasznev">Felhasználónév:</label>
    <div class="col-sm-10">
        <input type="text" class="form-control" value="<?php echo $felhasznev; ?>" id="felhasznev" name="felhasznev" required>
    </div>
</div>
```

Betöltődéskor a felhasználó jelenlegi felhasználó neve, jelszava és email címe jelenik meg. Amennyiben módosítani szeretné bármelyiket, van rá lehetősége. Mindegyik mező szerkeszthető, de üresen nem hagyható a fent már említett required attribútum miatt. A jelszó változtatásához arra van szüksége a felhasználónak, hogy a jelenlegi jelszavát tartalmazó beviteli mezőben beírja az új jelszót, amelyet használni szeretne, és az alatta lévő beviteli mezőben pedig újra meg kell adni az új jelszavát. Amennyiben a felhasználó bevitte az összes olyan adatot, amelyet meg akar változtatni, abban az esetben rákattint a módosítás

gombra. A módosítás gombra lefut a usereditcomplete nevű php fájl amely magáért a módosításért felel. Ez a php sem futtatható csak abban az esetben ha a gombra nyomunk ugyanis ebben az esetben a php lefutása nem egy olyan if-el kezdődik amely a userid-t vizsgálja a $_SESSION-ban hanem egy olyan amely azt nézi hogy a felhasználó megnyomta e a submit gombot azaz jelen esetünkben a módosítást. Ha igen akkor jön a lényeges része. Jelen esetben is változót készítünk azokból az adatokból, amelyet a felhasználó a beviteli mezőkben megadott. Ebben az esetben alkalmaztam a php beépített trim funkcióját, amely többek között eltávolítja a nem oda tartozó szóközöket a beírt adatok elejéről és végéről. Ezután következett a beírt jelszavak egyezésénem ellenőrzése. Amennyiben ez az ellenőrzést helyesen lefut abban az esetben a módosításért felelős sql változó segítségével feltöltjük az adatbázisba a frissített adatokkal. A felhasználó innentől kezdve a módosított adatokat látja a profil szerkesztése oldalon és ezeket az adatokat kell használnia bejelentkezéskor is. A fentebb leírt folyamat kódját lentebb látható. Ebben az esetben is használtam a header-t amely a hibaüzenetek kiírásához és az átirányításhoz is szükséges volt.

12.sz kép

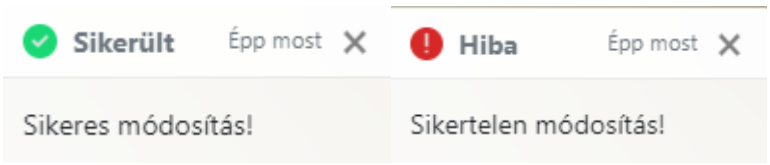
```php
if (isset($_POST['submit'])) {
    $id = $_SESSION['userid'];
    $felhasznev = trim($_POST['felhasznev']);
    $pwd = trim($_POST['pwd']);
    $pwdc = trim($_POST['pwdc']);
    $email = trim($_POST['email']);

    if ($pwd == $pwdc) {
        $sql = "UPDATE felhasznalok SET felhasznalonev = '$felhasznev' , jelszo = '$pwd' , emailcimfelhasznalo = '$email' WHERE felhasznalok.id = $id";
        $query = mysqli_query($connection, $sql);
        var_dump($sql);

        header("Location: useredit.php?mofidy=success");
```

Mind sikeres mind sikertelen módosítás esetén a szokásos módszerrel és dizájnnal megjelenítek egy felugró toast ablakot a felhasználónak, amelyről értesül arról, hogy a módosítás sikeres vagy sikertelen volt.

13.sz kép



1.2.4 TÁBLÁZATOK

Elsőként az olyan technológiákról írok, amelyeket minden táblázatban használtam. Ezek közül elsőként a leglényegesebbel, a php és sql segítségével megirt pagination amely azért felel hogy az adatbázis webes megjelenítésénél oldalakra osztja az adatokat. Minden oldalon 8 sornyi adat fér el, utána új oldalt nyit a php. Ez nagyban hozzá járult a reszponzív működéhez és ahhoz, hogy felhasználóbarát módon lehessen megtekinteni az adatbázisból lekért adatokat. Az ezért felelős kód részlet az alábbiakban látható. Ennek a kódnak egyedi folytatása van minden tábla megjelenítésekor mások a benne szereplő adatok.

14.sz kép

```php
$limit = 8;
$sql1 = "SELECT * FROM cars";
$result2 = mysqli_query($connection, $sql1);
$countAutokRows = mysqli_num_rows($result2);
$pageNumbers = ceil($countAutokRows / $limit);

if ($pageNumbers != 1) {
    $form = '<div class="dropdown show"><a class="btn btn-primary dropdown-toggle" href="#" role="button"'.
    'id="dropdownMenuLink"data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">Oldal</a>'.
    '<div class="dropdown-menu scrollable-menu" aria-labelledby="dropdownMenuLink">';
    for ($i = 1; $i <= $pageNumbers; $i++) {
        $form .= '<a class="dropdown-item" href="docketcarcontrol.php?page=' . $i . '">' . $i . '</a>';
    }
    $form .= '</div></div>';
    echo $form;
}
```

Minden táblázatot php echo segítségével iratok ki. De előtte lefut egy ellenőrzés az esetleges hibák elkerülése végett. Itt az autó táblában szereplő hibakezelés kódja látható valamint az első sorban a limit alkalmazása amely a pagination miatt volt szükséges. Az if feltételében tagadást vizsgál tehát ha nem fut le akkor átirányít a header, kiír egy hibaüzenetet és utána leállítja a php futását az exit paranccsal.

15.sz kép

```php
$sql = 'SELECT * FROM cars INNER JOIN tulajdonosok ON cars.tulid = tulajdonosok.tulid LIMIT ' . $start . ',' . $limit . ';';
$stmt = mysqli_stmt_init($connection);
if (!mysqli_stmt_prepare($stmt, $sql)) {
    header("Location: docketcarcontrol.php?error=sqlerror");
    exit();
```

Az oldalon egy dropdown menüre való kattintással megnyílik a legördülő menü és az ott van lehetősége a felhasználónak az általa kiválasztott oldalra lépni. Amennyiben a táblázatban nincs jelen elegendő adat, tehát 8-nál kevesebb sornyi adat van a táblázatban. Abban az esetben az oldalválasztó dropdown menü meg sem jelenik. Ezzel is egyszerűsödik a kezelés a felhasználó számára, hiszen ha nincs szükség rá nem jelenik meg.

Elsőként a táblázatok sorában egy kizárólag az admin felhasználó számára elérhető táblázatot mutatok be. Ez a táblázat felel a felhasználók kezeléséért. Ez a táblázat csak abban az esetben jelenik meg amennyiben a bejelentkezett felhasználó admin névvel rendelkezik. Amennyiben az ellenőrzés arra fut, hogy igen, a bejelentkezett felhasználó neve egyenlő adminnal, abban az esetben a menüpont megjelenik. Itt megjelennek a felhasználók és a hozzájuk tartozó adatok, amelyeket tárolok az adatbázisban. Ennél a menüpontnál is fontos volt a biztonság, azaz hogy csak is kizárólag olyan ember férjen hozzá az adatokhoz, aki erre jogosult és ne kerülhessen jogosulatlan felhasználók kezébe szenzitív adat. Egy php fájlban (usercontrol.php) oldottam meg a felhasználó ellenőrzést, hibakeresést és az adatbázis lekérdezést, valamint a megjelenítést is így nem volt szükség gombra, mert betöltődéskor azonnal lekérte az adatbázisból az értékeket. Hibák kezelése a szokásos módon megy végbe. A jelszó jelen esetben biztonsági okokból nem jelenik meg. Ha egy felhasználó esetleg elfelejti a jelszót, akkor kézzel visszaállítható/módosítható. Felhasználók törölhetők, ebben az esetben kitörlődnek az adatbázisból is és többé nem lesz jogosultságuk használni sem a weboldalt sem az asztali alkalmazáshoz nem fognak hozzáférni.

A maradék három darab táblázat. Az autókat, tulajdonosokat és a cégeket jelenít meg. Elsőként ejtenék pár szót az autók táblázatról és annak megjelenítéséről. Az autók táblázatban minden adatot megjelenítek. Mivel az adatbázisban csak a leglényegesebb adatokat tárolom így itt már nem válogattam a megjelenítendő adatokat, hanem az egész autók táblát kiírattam az adatbázisból egy adatot kivéve, ami az egyedi azonosítója minden autónak. Erre azért nincs szükség, mert abban az esetben, ha egy tulajdonosnak kettő teljesen ugyanolyan autója van a rendszerben, akkor sem lehet össze téveszteni, mert az alvázszám és a rendszám semmilyen körülmények között nem egyezhet kettő darab autónál. Rendezés Tulajdonos szerint van, mert ez a leglogikusabb rendezési forma. Az adatbázis szerkesztésére a weboldal jelen állapotában nincs lehetőség. Az adatok törölhetők az adatbázisból és minden törlés után frissül a megjelenített tábla így mindig aktualizálva van. Második táblázat megjelenítésénél hasonlóan az előzőkhez a tulajdonos egyedi azonosítóját nem írom ki, mert nem lényeges adat a felhasználó szempontjából. Az adatbázis szerkesztésére a weboldal jelen állapotában sajnos itt sincs lehetőség. Az adatok itt is törölhetők az adatbázisból és minden törlés után frissül a megjelenített tábla így mindig aktualizálva van. Harmadik táblázat a cégeket jeleníti meg és teljes mértékben azonos funkcionálisan, mint a másik kettő.

1.2.5 KAPCSOLAT MENÜ MŰKÖDÉSE

A még be nem jelentkezett oldallátogató tud nekünk email írni nekünk egy konfigurált php oldal segítségével, akár kérdése van, akár úgy gondolja meg szeretné osztani velünk a saját gondolatait, javaslatait a weboldallal kapcsolatban. Azt a mondandóm elején ki szeretném kötni, hogy sajnos az XAMPP konfigurációjának átírása és külső script használata nélkül localhost használatával nem lehetséges tesztelni ezt a funkciót. Egy saját bérelt domainnal kipróbáltam úgy tökéletesen működött. Localhoston való tesztelés is félig sikeresnek mondható, mert egy kellő alapossággal beállított XAMPP konfigurációval el lehet érni, hogy a google SMTP szerverén keresztül egy gmail-es címet használva lehessen email küldeni ám ennek beállítása és tesztelése időigényes folyamat. Ezt a funkciót azért szerettem volna beépíteni a weboldalra egy visszajelzés funkciót. Erre több féle megoldást is találtam az interneten körbe nézve, de a legjobban az tetszett amikor az oldal dizájnjába beleillő dedikált menüt kapott ez a funkció. Működése nem bonyolult normális esetben. Meg kell adnunk a saját email szerverünk elérhetőségét, majd beállítani saját preferenciáinkra. Értem ezalatt azt, hogy a saját email címünkre jöjjenek meg a visszajelzésből küldött levelek. A webes mappában található egy PHPMailer nevű mappa, ez kellett ahhoz, hogy localhoston keresztül tudjam tesztelni a küldésért felelős php kódot. (Letöltési link) Én nem használtam ki minden szegletét, viszont a projekt mappák közé oda raktam a teljes master-t amely letölthető github.com-ról. Következőkben írnék pár sort a konfigurációról valamint a tényleges kód működéséről.

Elsőként az XAMPP konfigurációs állományai között a php.ini és a sendmail.ini állományokat kell módosítanunk. A php.ini fájlban be kell írnunk a használni kívánt port számát valamint a használni kívánt smtp szervert. Ez látható az alábbi képen.

16.sz kép

```
[mail function]
; For Win32 only.
; http://php.net/smtp
SMTP=smtp.gmail.com
; http://php.net/smtp-port
smtp_port=465

; For Win32 only.
; http://php.net/sendmail-from
;sendmail_from = mzmoddingteam@gmail.com

; For Unix only.  You may supply arguments as well (default: "sendmail -t -i").
; http://php.net/sendmail-path
;sendmail_path = "\"D:\xampp\sendmail\sendmail.exe\" -t"
```

Amennyiben ezzel megvagyunk, át kell mennünk a sendmail.ini állományba. Itt szintén be kell állítanunk az smtp szervert valamint a portszámot, ezen felül pedig az ssl (secure socket layer) beállítását auto-ra kell állítani. Ezután következik a saját email címünk beállítása, amit az alábbi képen lehet látni. Valamint pár sorral lejjebb meg kell adni a force

17.sz kép

```
auth_username=mzmoddingteam@gmail.com
auth_password=[redacted]
```

Ha ezzel is megvagyunk, akkor indítsuk újra az XAMPP-on belül az apache-ot. Következő lépésben azon az email címen (jelen esetben mzmoddingteam@gmail.com) be kell állítani a nem megbízható alkalmazások hozzáférését, különben a gmail le fogja tiltani és nem fog történni semmi hiába fut le hiba nélkül a php. Innentől a php megírása volt a lényeges. A kapcsolat.php jelen esetben nem csak a megjelenítésért volt felelős, hanem a gombra lefuttattam egy ajax kérést, amely kommunikált az email küldő php állománnyal. Ez az ajax kérés átadta a beírt értékeket a form-to-email.php állománynak.

18.sz kép

```
if (isNotEmpty(name) && isNotEmpty(email) && isNotEmpty(subject) && isNotEmpty(body)) {
    $.ajax({
        url: 'form-to-email.php',
        method: 'POST',
        dataType: 'json',
        data: {
            name: name.val(),
            email: email.val(),
            subject: subject.val(),
            body: body.val()
        }, success: function (response) {
            if (response.status == "success")
                alert('Email elküldve!');
            else {
                alert('Kérjük probálja újra!');
                console.log(response);
            }
        }
    });
}
```

A form-to-email.php állomány egy megjelenítést nem tartalmazó php. Azért felel, hogyha a felhasználó rákattint a gombra abban az esetben lefusson és az email elküldje. Valamint a szokásos módon kezelt toast menüvel kiírt hibaüzenethez innen irányít át a megfelelő oldalra.

1.3 ASZTALI ALKALMAZÁS

Az asztali alkalmazás megtervezésénél elsődleges szempont volt az, hogy a lehetőségekhez mérten minden hibát-hibaesetet kezelni tudjak. Valamint az hogy egy teljes körű és teljes értékű partneralkalmazásként működjön együtt a webes alkalmazással. Az én esetemben nem az admin(ok) számára létrehozott támogató alkalmazásról van szó, hanem egy teljes értékű használható adatbáziskezelőről.

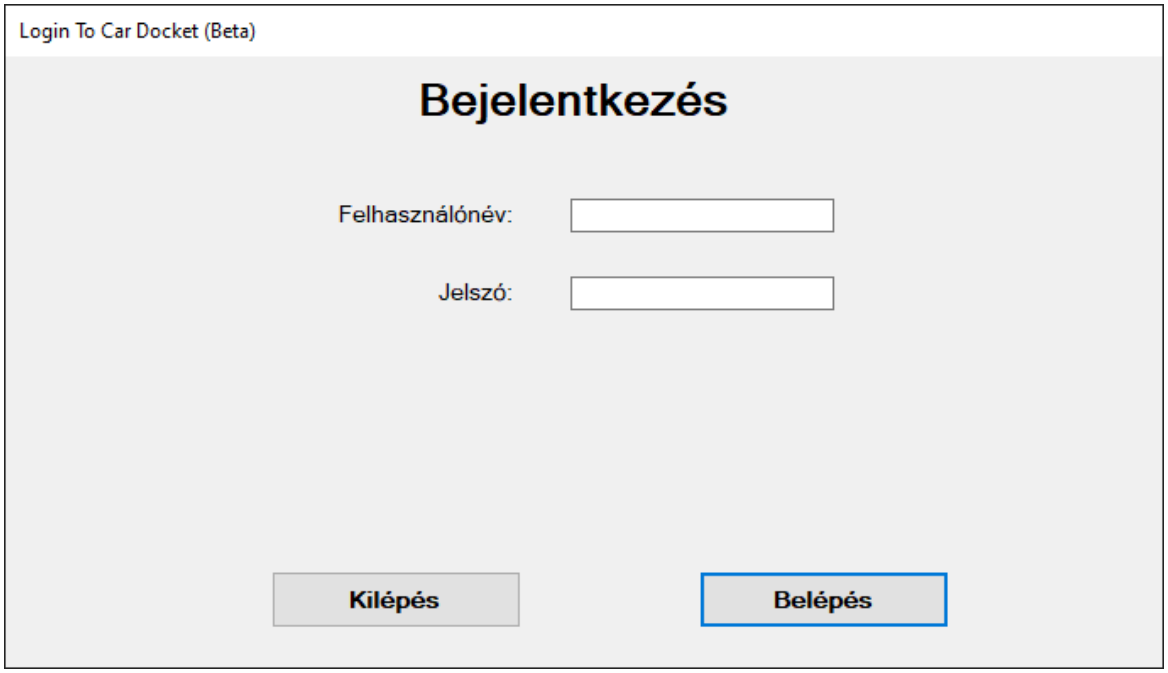
1.3.1 TERVEZÉS

A program az iskolában tanult és aktívan használt Visual Studio környezetben készült, C# nyelven. A készítésnél fő szempont volt a logikus, gyors működés amely nagy segítség lehet a felhasználónak. Ezen felül a bemutatás miatt pár bónusz funkció is bekerült, amelyekről később fog szó esni a dokumentum második felében. Ezen felül az adatmegjelenítés is szerves részét képezi a programnak, erre az órák során megismert és használt DataGridView táblás módszert használtam. Mivel több DataGridView-t is meg kellett jelenítenem ezért az adatok kezelésére használt formot is több oldalra osztottam. Ezzel az alkalmazással bárki bármilyen adatot módosíthat az adatbázisban így csak a beregisztrált felhasználók használhatják.

1.3.2 BEJELENTKEZÉS

A bejelentkező felületnek egy teljesen letisztult egyszerű, de nagyszerű felületet terveztem. Két beviteli mező és kettő darab gomb kapott rajta helyet, lásd 19.sz kép. Értelemszerűen a bejelentkezés gombra lefut egy adatbázis ellenőrzés amely azt nézi hogy létezik e ilyen felhasználó-jelszó páros az adatbázisban, lásd 20.sz kép. Amennyiben nem létezik abban az esetben egy üres label feltöltődik szöveggel és kiírja a felhasználónak, hogy rossz valamelyik adat, amelyet beírt a beviteli mezőbe. A programból kettő féleképpen lehet kilépni. Az egyik megoldás hogy a kilépés gombra kattintunk, (Vigyázat ilyenkor a teszt beállítás miatt törlődik az adatbázis!) vagy a kijelentkezés gombra kattintva majd a login formon lévő kilépést használva.

19.sz kép



20.sz kép

```
private void buttonBelepes_Click(object sender, EventArgs e)
{
    Connection cs = new Connection();
    i = 0;
    string felhasznalonev = textBoxFelhasznalonev.Text;
    string jelszo = textBoxJelszo.Text;
    MySqlConnection connection = new MySqlConnection(cs.getConnectionString());
    connection.Open();
    string query = "SELECT * FROM felhasznalok WHERE felhasznalonev ='"+felhasznalonev+"' AND jelszo='"+jelszo+"';";
    MySqlCommand cmd = new MySqlCommand(query, connection);
    cmd.ExecuteNonQuery();
    DataTable dt = new DataTable();
    MySqlDataAdapter da = new MySqlDataAdapter(cmd);
    da.Fill(dt);
    i = Convert.ToInt32(dt.Rows.Count.ToString());
    if (i == 0)
    {
        labelHibauzenet.Text = "Rossz felhasználónév vagy jelszó!";
        textBoxFelhasznalonev.Text = "";
        textBoxJelszo.Text = "";
    }
    else
    {
        Form1 form1 = new Form1();
        form1.Show();
        this.Hide();
    }
}
```

A bejelentkezéskor figyelembe vettem, hogy teszt adatbázisnak jelen esetben létre kell jönnie, ezért a Form1login betöltéséhez hozzá rendeltem az adatbázis és a felhasználók tábla létrehozását valamint ezek tesztadatokkal való feltöltését, amely a program éles használatakor nem szükséges funkciók, de jelen esetben így oldottam meg. Sikeres bejelentkezés esetén a felhasználó már bent is van az adatbáziskezelő programban.

1.3.3 FUNKCIÓK MEGTERVEZÉSE

A programot az iskolában tanultak alapján MVC alapokra helyezve Objektum Orientáltan készítettem el. Ennek a módszernek megfelelően külön mappákba rendeztem minden osztályt annak megfelelően, hogy milyen feladata van. Ezen módszer miatt könnyen átláttam melyik programrésznek milyen feladata van és miért felelős. Az asztali alkalmazás fő funkciója maga az adatbáziskezelés. Három táblát kezeltem az adatbázisból, az autók, tulajdonosok és végül a cégek táblát. Az adatbázisból való lekérés csak abban az esetben sikeres amennyiben van kapcsolat a program és az adatbázis között. A

1.3.4 AUTÓK KEZELÉSE

Az autók adatainak eltárolása a Car nevű osztályban történik. Az osztály rendelkezik konstruktorral (lásd 21.sz kép), amely tartalmaz minden adatot. Ezen felül természetesen minden változóra írtam Set és Get metódusokat is.

21.sz kép

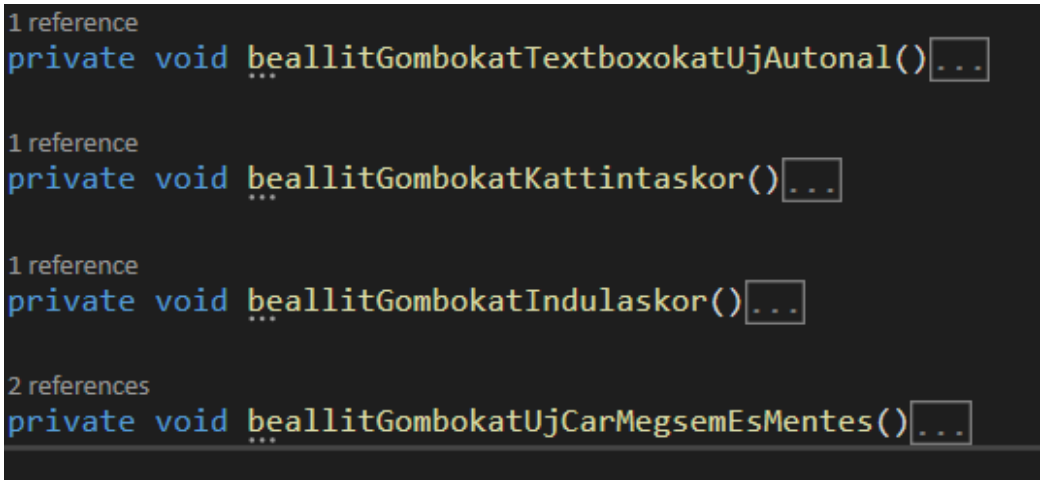
```
public Car(int id, string marka, string tipus, string gyartasi_ev, int vetelar,
{
    this.id = id;
    this.marka = marka;
    this.tipus = tipus;
    this.gyartasi_ev = gyartasi_ev;
    this.vetelar = vetelar;
    this.rendszam = rendszam;
    this.kilometeroraallas = kilometeroraallas;
    this.alvazszam = alvazszam;
    this.gepkocsi_tipusa = gepkocsi_tipusa;
    this.uzemanyag = uzemanyag;
    this.sebessegvalto_tipusa = sebessegvalto_tipusa;
    this.tulajdonos_nev = tulajdonos_nev;


}
```

Az autók megjelenítéséért egy DataGridView a felelős. A Tab Controlnak ezen oldalán minden szükséges gomb, beviteli mező, hiba jelzésért felelős elem megtalálható. Ám ezek a gombok és beviteli mezők csak abban az esetben jelennek meg amikor szükség van rájuk azaz a felhasználó meghívja azt a funkciót ami szükségessé teszi azt hogy megjelenjenek. Például a beviteli mezők csak abban az esetben jelennek meg amennyiben a felhasználó kiválaszt egy elemet a DataGridView-ból vagy rákattint az Új gombra ezáltal meghívja a beviteli mezők megjelenéséért felelős program kódot. Amennyiben a felhasználó mégsem szeretne hozzá adni új adatot, lehetősége van kilépni az új adat hozzáadás menüjéből. Valamint abban az esetben ha még nem kattintott egy adatra sem az össze gomb és beviteli mező el van rejtve kivéve az új adatbevitelért felelős gomb. Ezeket a vezérlő részeket

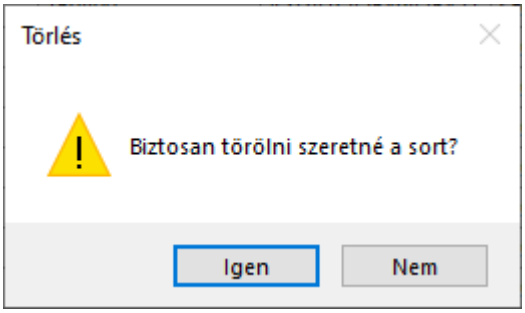
szemlélteti az alábbi kép. Ezeknek a beviteli mezők és gombok megjelenítésének vezérlését megkönnyíti a Visual Studio által felkínált panelek használata amelyre több gombot vagy beviteli mezőt is rá tudunk tenni és így elég annak az egy panelnek a megjelenítését kezelni, nem szükséges külön az össze beviteli mezőt, gombot és labelt egyesével kezelni.

22.sz kép



Ahogy képen is látható többféle eseményre létrehoztam a metódusokat és mindegyik másik állapotért felel, lehet akár új adat feltöltésénél vagy adat betöltés utáni állapot is. Amennyiben a felhasználó adatot szeretne módosítani, rá kell kattintania egy tetszőleges sorra, majd a beviteli mezők feltöltődnek az aktuálisan kiválasztott sor adataival (jelen esetben ez egy-egy autó adata). Amennyiben módosítani szeretné az adatokat szabadon írhat az adatokkal feltöltött cellákba (azonosító kivétel, azt nem lehet módosítani mert egyedi), amikor végzett a felhasználó akkor a módosít gombbal véglegesítheti az adatmódosítást. Amennyiben hibás adatot szeretne felvenni, abban az esetben hibaüzenetet fog megjeleníteni a program és nem hajtja végre a módosítást. Abban az esetben, ha a felhasználó törölni szeretne az adatbázisból a program figyelmeztet és egy felugró ablakban megkérdezi hogy biztosan törölni szeretnénk-e az éppen kijelölt adatot vagy sem.

23.sz kép



A programban a márkák megadását egyedi módon oldottam meg egy külső adatbázist hívtam segítségül. A programhoz csatoltam a Visual Stúdió külső adatbázis kezelő segítségével egy „.mdb” (Microsoft Data Base) kiterjesztésű Excelből importált és adatokkal feltöltött Microsoft Access 2016-ban elkészített adatbázist, amelyet a Visual Stúdió társított és bele helyezett a projekt mappájába (Bin>Debug) . A combobox amely a márkákat tárolja betöltéskor a Form1_Load eseményben feltöltődik a beállított adatbázis értékeivel (lásd 24.sz kép). Azért választottam ezt a módszert, mert gyorsabb könnyebb és egyszerűbb adatkezelést és adatbázis bővítést tesz lehetővé amennyiben igény van rá, hiszen a csatolt adatbázis egyszerűen megnyitható és szerkeszthető a megfelelő Microsoft által készített felhasználóbarát szoftverrel. Hozzá szeretném fűzni hogy ugyanezzel a megoldással töltöttem fel a cégek formon lévő városokat tartalmazó combobox tartalmát is.

24.sz kép

```
this.varosokTableAdapter.Fill(this.comboboxvarosokDataSet.varosok);
this.autokTableAdapter.Fill(this.comboboxautoDataSet.autok);
```

1.3.5 TULAJDONOSOK KEZELÉSE

A tulajdonosok kezelése nem sokban tér el az autók kezelésétől, mindössze adattípusok különböznek. Egy fontos dolog hogy addig nem engedélyezett egy tulajdonos törlése a rendszerből ameddig rendelkezik autókkal. Tehát amennyiben egy tulajdonost törölni szeretnénk, a rendszerből előtte el kell távolítani azokat az autókat amelyekkel rendelkezik. Mert egy autó sem maradhat tulajdonos nélkül. Amennyiben megpróbáljuk törölni abban az esetben hibát fog jelezni a program, mert a törlés nem lehetséges.

1.3.6 CÉGEK KEZELÉSE

A cégek kezelése szintén nem sokban tér el az előző adatok kezelésétől, mindössze annyi különbség van az autók kezeléséhez képest, hogy itt sem lehet, addig egy céget törölni ameddig vannak hozzárendelt tulajdonosok, hiszen minden tulajdonos tartozik egy céghez. Tehát amennyiben egy céget törölni szeretnénk az adatbázisból, először törölnünk kell a hozzá tartozó tulajdonosok autóit majd a tulajdonosokat és azután következhet a cég törlése. Ebben az esetben is hibát dob a program, amennyiben úgy próbálunk meg törölni egy céget hogy még tartalmaz tulajdonosokat mert ebben az esetben nem engedélyezett a törlés.

1.3.7 TESZTELÉS

Az egységtesztek megírása egy rendkívül fontos eleme a dokumentációnak, mert ezekkel tudjuk ellenőrizni és ezek segítségével lehetünk biztosak abban, hogy amely értékeke visszakapjuk, azok helyesek. Ennek a megállapítására írtam egységteszteket mind az autókra, mind a tulajdonosokra és a cégekre is. Ebből szeretném a leglényegesebbeket megmutatni a dokumentációban. Elsőként egy DateTime ellenőrzés tesztjét csatoltam, amelyhez a RegEx-et használtam segítségül, ez felelős azért hogy a gyártási év mezőbe beírt adat megfelelő formátumban legyen bele írva a beviteli mezőbe.

25.sz kép

```
public bool isValidDateTime(string gyartasi_ev)
{
    var regex = new Regex(@"^\d{4}[/.-]((0\d)|(1[012]))[/.-](([012]\d)|3[01])$");
    return regex.IsMatch(gyartasi_ev);
}
```

26.sz kép

```
public void isValidDateTimeTest()
{
    try
    {
        Car c = new Car(1, "Audi", "A5", "2020-02-15", 25000, "RRR-234", 65481, "AZ345345", "SzGK", "Benzin", "Automata", "Kiss Ferenc")
        Assert.IsTrue(true);
    }
    catch (Exception ex)
    {
        Assert.Fail("A gyártási év mező nem lehet üres");
    }
}
```

Másodikként szintén egy RegEx kifejezést használtam annak az ellenőrzésére, hogy mind a tulajdonoshoz mind a céghez beírt email cím mező helyesen legyen kitöltve.

27.sz kép

```
public static bool ValidEmail(string email_cim)
{
    var regex = new Regex(@"([a-z0-9][-a-z0-9_\+\.]*[a-z0-9])@([a-z0-9][-a-z0-9\.]*[a-z0-9]\.
    return regex.IsMatch(email_cim);
}
```

28.sz kép

```
[TestMethod()]
0 references
public static void ValidEmail()
{
    try
    {
        Ceg c = new Ceg(1, "Teszt1", 1234, "Szeged", "Radnóti", 12, "valami@gmail.com");
        Assert.IsTrue(true);
    }
    catch (Exception ex)
    {
        Assert.Fail();
    }
}
```

2. FELHASZNÁLÓI DOKUMENTÁCIÓ

2.1 TELEPÍTÉSI ÚTMUTATÓK

2.1.1 ASZTALI ALKALMAZÁS TELEPÍTÉSE

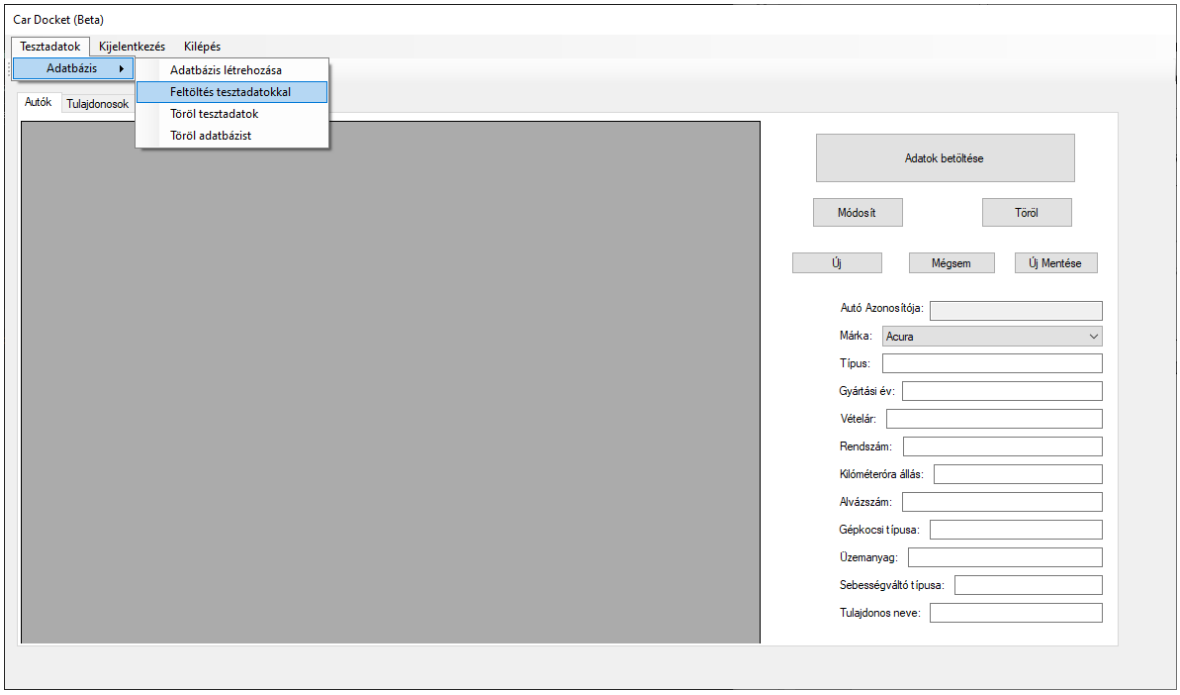
Az asztali alkalmazás elindításához futtassa le a Szakdoga_autonyilvantartas.exe nevű fájlt. Ez alapértelmezetten a projekt mappáján belüli bin és az azon belüli Debug mappában található. Telepíteni nem szükséges. Az alkalmazásba való belépés tesztelési célzattal az alábbi adatokkal lehetséges:

- Felhasználónév: admin
- Jelszó: admin

Az alkalmazás indítása előtt elengedhetetlen az adatbázishoz szükséges Apache és MySQL szerver elindítása. Az adatbázis létrehozása automatikusan történik meg a program indulásakor. (Adatbázis és azon belül a felhasználók tábla teljes egészében elkészül a bejelentkező felület betöltése alatt). Sikeres belépés esetén a program teszt jellege miatt fel kell tölteni tesztadatokkal az alábbiakban látható módon.

Figyelem! A kilépés gombra kattintva eldobja a program az egész adatbázist. És következő indításkor újra az alap teszt adatokkal fog rendelkezni.

29.sz kép



2.1.2 WEBES ALKALMAZÁS HASZNÁLATA

Első lépésként el kell indítani az XAMPP segítségével az Apache kiszolgálót és a MySQL szervert. Amennyiben nem fut az asztali alkalmazásunk ebben az esetben nincs adatbázis, amely elengedhetetlen az oldal helyen működéséhez, ebben az esetben lépjen tovább az adatbázis telepítése menüponthoz és annak a végrehajtása után folytatható a webes alkalmazás tesztelése.

Egy számunkra szimpatikus böngészőt kiválasztva (ajánlott: Google Chrome vagy Microsoft Edge) vagy közvetlenül, vagy a localhost/ cím segítségével nyissuk meg a weboldal fő mappájában elhelyezkedő index.php nevű állományt. Ez maga a Főoldal ahonnan a navigációs menü segítségével minden funkciót elérünk. Tesztelése célzattal 2 felhasználóval lehet bejelentkezni és ezesetben a felhasználó email címe is beírható a bejelentkezésnél mert azzal is működik a rendszer. A teszt felhasználók adatai az alábbiakban láthatóak:

- Felhasználónév: admin
- Email cím: proba1@gmail.com
- Jelszó: admin

- Felhasználónév: Teszt1
- Email cím: proba2@gmail.com
- Jelszó: 1234

Fontos hogy a két felhasználó különböző adatokat ér el. Az admin felhasználó tudja kezelni a többi felhasználót még a Teszt1 felhasználó nem képes erre.

Ki szeretnék térni arra, hogy amennyiben localhost alatt szeretné használni a felhasználó a kapcsolat.php email küldő funkcióját tesztelni ebben az esetben kérem, tekintse meg a dokumentáció erről szóló részét (1.2.5-ös rész Kapcsolat php működése), amit ide kattintva elérhet.

2.1.3 Adatbázis telepítése

Az adatbázis egyszerűen telepíthető a localhost/phpmyadmin címen elérhető adatbázis kezelő rendszer segítségével. De ehhez szükséges a MYSQL szerver elindítása az XAMPP segítségével. Az oldal betöltése utána navigáljunk el az importálás fülre a felső menüt használva. Itt nyomjunk a Fájl kiválasztása gombra, itt be tudjuk tallózni a web projekt sql mappájából az autonyilvantartas.sql nevű fájlt. Ez egy kis időbe fog telni az adatbázisban lévő tesztadat mennyiség miatt. A folyamat végén ott lesz az adatbázisunk minden táblával és adattal feltöltve.

2.2 Asztali alkalmazás példa funkció

2.2.1 Új cég adat felvétele

Az alkalmazás sikeres bejelentkezés után betölti a programot. A funkció működéséhez szükség lesz adatbázisra. A programba beépített tesztadat feltöltéssel létrejönnek a táblák és a teszt adatok és minden funkció helyesen fog működni. Ezek után kattintson a Tab Control Cégek menüpontra. Ezek után az Adatok betöltése gombra kattintva megjelennek a teszt adatok a DataGridView-ban. Ezután kattintson az új gombra, ekkor megjelennek a beviteli mezők üresen. A helyes kitöltéshez fontos hogy az adószámhoz számot írjunk valamint a város kiválasztásánál, ha elkezdünk írni egy város nevet akkor a combobox tartalmában elkezd keresni és oda fog ugrani (a felhasználó nem látja hogy írna bármit) itt kiválasztjuk a várost majd az utcához logikusan az utcát írjuk be a számhoz pedig a házszámot. Az email címhez is érvényes email címet írjunk be. Ezeket az adatokat helyes kitöltés után az új mentése gombra kattintva menthetjük az adatbázisban.

2.3 Weboldali funkciók

2.3.1 Belépett felhasználó adatainak módosítása

Sikeres bejelentkezés után a navigációs sávon található egy Profil feliratú menü amelyet lenyitva megtalálja a profil szerkesztése menüpontot. Erre kattintva megjelenik egy beviteli mezőkkel rendelkező oldal ahol látjuk a felhasználónevünket, rejtve de jelen van a jelszó is, valamint a felhasználóhoz tartozó email cím. Most tetszőlegesen módosítsuk a felhasználó jelszavát és email címét majd nyomjuk meg a módosítás gombot. A gombnyomás után a módosítás sikeressége vagy épp sikertelensége függvényében kapunk egy üzenetet amely tájékoztatja a felhasználót arról hogy sikeres vagy sikertelen volt a módosítása.

3. Összegzés

Az eredetileg kigondolt és megtervezett funkciók nagyrészét sikerült megvalósítani, de sajnálatos módon vannak funkciók, amelyeket nem sikerült kellően implementálni a szakdolgozatomba. Voltak akadályok, amelyeket az elején el sem tudtam volna képzelni, hogy meg fogom tudni oldani, de örömmel tölt el, hogy rengeteg ilyet az iskolában elsajátított tudásommal meg tudtam oldani. Sokat fejlődtem a projekt készítése közben. Úgy gondolom sikerült egy kerek egész zökkenőmentesen működő összetett alkalmazást létrehoznom. A jövőben tervezem a szoftver fejlesztését és bővítését plusz funkciókkal. A legfontosabb ilyen fejlesztés az adatbázis teljes körű kezelése webes alkalmazással is, ehhez sajnos még nem rendelkezem elég tudással.

4. FORRÁSMEGJELŐLÉSEK

- https://stackoverflow.com/
- https://getbootstrap.com/
- https://jquery.com/
- https://www.php.net/
- https://www.w3schools.com/
- https://regex101.com/
- https://www.iconfinder.com/
- https://startbootstrap.com/
- https://github.com/PHPMailer/PHPMailer
- Órai munkák anyagai
- Tanárok segítségével írt órai kódsorok

Weboldalon felhasznált ikonok:

- https://www.iconfinder.com/icons/381599/error_icon
- https://www.iconfinder.com/icons/381607/complete_icon
- https://www.iconfinder.com/icons/2411789/account_avatar_male_person_profile_user_icon

5.RENDSZERKÖVETELMÉNYEK

A program 64 bites Windows 10-en lett tesztelve, azon is készült. Gyakorlatilag Windows 7-en is képes futni, így a Windows 7 rendszerigényét írom le:

- 1 gigahertzes (GHz) vagy gyorsabb 32 bites (x86) vagy 64 bites (x64) processzor.
- 1 gigabájt (GB) RAM (32 bites rendszerhez) vagy 2 GB RAM (64 bites rendszerhez).
- 16 GB (32 bites rendszerhez) vagy 20 GB (64 bites rendszerhez) szabad lemezterület.
- DirectX 9 grafikus eszköz WDDM 1.0 vagy újabb illesztőprogrammal .
- Hiba alakulhat ki, ha: A Xampp nincs elindítva.
- A megfelelő adatbázis nincs importálva.
- Nincs internet.
- A regisztráció manuálisan történt, így a jelszó nem hash-elt.
- A külső fájlok nincsenek importálva.
- Szoftver esetén hiányzik a MySql.Data fájl.
- A szoftver nem a C meghajtóról van indítva.
- A webes elérés esetén a documentroot nincs helyesen beállítva.
- A webes fájlok nem htdocs mappában találhatóak.

5. KÖSZÖNETNYILVÁNÍTÁS

Ezúton szeretnék köszönetet mondani Gyuris Csaba és Bálint Róbert témavezető szaktanáraimnak, akik munkájukkal hozzájárultak fejlődésemhez és szakmai előre menetelemhez. Hozzáállásukkal, tanácsaikkal és tanítási módszereikkel az utóbbi két évben mindvégig támogatták céljaim elérését.

Tisztelettel köszönöm!

6. PLÁGIUMNYILATKOZAT

TANULÓI NYILATKOZAT

Alulírott Marosi Márk Dániel, Szegedi Szakképzési Centrum Vasvári Pál Gazdasági és Informatikai Szakgimnáziuma tanulója kijelentem, hogy a „Fotós segéd” című záródolgozat a saját munkám.

Kelt:

_____

aláírás