**Ceglédi SZC Közgazdasági és Informatikai**

**Technikum**

Szakképzés megnevezése: Szoftverfejlesztő és -tesztelő

Szakma azonosító szám: 5 0613 12 03

**ZÁRÓDOLGOZAT**

|  |  |
| --- | --- |
| Témavezető:  Szabó Dániel | Készítette:  Dúzs Maja Flóra  Kiss Ádám |

**Mancs Állatmenhely**

Cegléd, 2024

## Tartalomjegyzék

[**Tartalomjegyzék** 2](#_Toc164163742)

[1. Bevezetés 5](#_Toc164163743)

[1. A záródolgozat témája 5](#_Toc164163744)

[2. Dokumentáció 5](#_Toc164163745)

[1. Az elkészítéshez felhasznált szoftverek 5](#_Toc164163746)

[2. A használt programozási nyelvek 5](#_Toc164163747)

[3. Az adatbázis 6](#_Toc164163748)

[3.1 A táblák szerkezete 7](#_Toc164163749)

[3.2 Megkötések a táblák között 9](#_Toc164163750)

[4 A weboldal felhasználói része 11](#_Toc164163751)

[4.1 Főoldal 11](#_Toc164163752)

[4.2 Örökbefogadható állataink 14](#_Toc164163753)

[4.3 Információk 15](#_Toc164163754)

[4.4 Galéria 15](#_Toc164163755)

[4.5 Kapcsolat és vélemények 16](#_Toc164163756)

[4.6 Belépés és regisztráció 18](#_Toc164163757)

[4.7 Örökbefogadás 19](#_Toc164163758)

[4.8 Segítenék 20](#_Toc164163759)

[4.9 Profil 21](#_Toc164163760)

[5 A weboldal admin része 22](#_Toc164163761)

[5.1 Galéria szerkesztés 22](#_Toc164163762)

[5.2 Örökbefogadható állataink szerkesztése 22](#_Toc164163763)

[5.3 Éves adatok szerkesztése 22](#_Toc164163764)

[5.4 Adminjogosultság kapcsoló 23](#_Toc164163765)

[5.5 Kijelentkezés 23](#_Toc164163766)

[6 A felhasználói oldal fejlesztése 25](#_Toc164163767)

[6.1 Közös részek 25](#_Toc164163768)

[6.2 index.html felépítése 27](#_Toc164163769)

[6.3 allataink.html felépítése 30](#_Toc164163770)

[6.4 informaciok.html felépítése 30](#_Toc164163771)

[6.5 galeria.html felépítése 30](#_Toc164163772)

[6.6 kapcsolat.html felépítése 32](#_Toc164163773)

[6.7 login.html és regisztracio.html felépítése 34](#_Toc164163774)

[6.8 orokbefogadas.html felépítése 38](#_Toc164163775)

[6.9 segitenek.html felépítése 42](#_Toc164163776)

[6.10 jelszomodositas.html felépítése 43](#_Toc164163777)

[7 Az admin felület fejlesztése 43](#_Toc164163778)

[7.1 galeriaszerkesztes.html felépítése 43](#_Toc164163779)

[7.2 További admin oldalak 45](#_Toc164163780)

[3. C# asztali alkalmazás 46](#_Toc164163781)

[1. Bevezetés 46](#_Toc164163782)

[2. Alapok 46](#_Toc164163783)

[3. Funkciók 46](#_Toc164163784)

[3.1 Bejelentkezés 46](#_Toc164163785)

[3.2 Adatok megjelenítése 48](#_Toc164163786)

[3.3 Adatok módosítása és törlése 48](#_Toc164163787)

[4. Az alkalmazás felépítése 49](#_Toc164163788)

[5. Az alkalmazás kódja 49](#_Toc164163789)

[6. Összegzés 49](#_Toc164163790)

[4. Tesztelési hibák 49](#_Toc164163791)

[1. Üresen is beküldhető az űrlap 49](#_Toc164163792)

[2. A diagram mérete 50](#_Toc164163793)

[3. Regisztráció, bejelentkezés 50](#_Toc164163794)

[5. Fejlesztési lehetőségek 51](#_Toc164163795)

[1. Fejlesztendő funkciók 51](#_Toc164163796)

[1.1 Adományozás funkció 51](#_Toc164163797)

[1.2 Jelszó titkosítás 51](#_Toc164163798)

[1.3 Bejelentkezési adatok tárolása, munkamenetek létrehozása. 51](#_Toc164163799)

[6. Felhasznált irodalom 52](#_Toc164163800)

[1. Irodalomjegyzék 52](#_Toc164163801)

[**Köszönetnyilvánítás** 53](#_Toc164163802)

# Bevezetés

## A záródolgozat témája

Záródolgozatunkként egy állatmenhely weboldalát és egy hozzá tartozó asztali alkalmazást készítettünk el. Egyrészt azért választottuk ezt a témát, mert mindketten szeretjük az állatokat, illetve tapasztalatunk is van a nevelésükben, hiszen a való életben mi is gazdik vagyunk. Másrészt, a kóbor állatok száma évről évre növekszik, a menhelyek befogadóképessége pedig ezzel együtt csökken. Oldalunk készítése során az volt a célunk, hogy létrehozzunk egy mindenki számára elérhető, könnyen kezelhető, korszerű felületet, ezzel ösztönözve az embereket az örökbefogadásra.

# Dokumentáció

## Az elkészítéshez felhasznált szoftverek

A weboldal kódjának megírásához, szerkesztéséhez a Microsoft által fejlesztett **Visual Studio Code-**ot használtuk, a beépített Git és Node.js támogatás jelentősen megkönnyítette a fejlesztési folyamatot. Az asztali alkalmazást pedig a **Visual Studio** fejlesztői környezetben hoztuk létre.

Az adatbázis létrehozásához és teszteléséhez a **XAMPP** nevű szoftvercsomagot használtuk, amely tartalmazza az Apache HTTP szervert és a **MySQL** adatbázis-kezelő rendszert.

## A használt programozási nyelvek

**HTML** (HyperText Markup Language): A projekt során az alapvető építőelemet a HTML jelentette, ez alkotja a weboldalunk alapját. A HTML egy olyan nyelv, amely lehetővé teszi a weboldalak strukturálását és tartalmának megjelenítését. A leggyakrabban használt elemek közé tartoznak például: <div>, <p>, <h1>, <img>, <a>. Az elemeket osztályokkal és azonosítókkal láthatjuk el, amelyek segítségével stílusokat és funkciókat lehet hozzájuk rendelni.

**CSS** (Cascading Style Sheets): A CSS segítségével alakítottuk ki a weboldalak vizuális megjelenését, esztétikusabbá téve azokat. Az osztályok és az ID-k használata megkönnyíti az egyedi stílusok és design elemek létrehozását.

**JavaScript**: A JavaScript egy dinamikus, magas szintű programozási nyelv, amelyet általában a böngészőkben futó webalkalmazásokhoz használnak, ahogy mi is tettük. Az eseménykezelők révén lehetővé teszi az interakciót az elemekkel, például a gombokkal, az űrlapokkal stb. **React**-ot is használtunk, ami egy JavaScript könyvtár, felhasználói felületek készítésére alkalmas. Segítségével interaktívvá tudjuk tenni a weboldalainkat.

**SQL** (Structured Query Language): Az SQL egy nyelv, amelyet az adatok tárolására, lekérdezésére, frissítésére és törlésére használunk az adatbázisokban. Lekérdezések segítségével adatokat kaphatunk vissza az adatbázisból, illetve módosíthatjuk azokat az adatokat, amelyeket tárol. Általunk leggyakrabban használt parancsai a SELECT, UPDATE, DELETE, INSERT.

**C#:** A Microsoft által a .NET keretrendszer részeként kifejlesztett objektumorientált programozási nyelv. Ebben készítettük el az asztali alkalmazásunkat.

## Az adatbázis

1. ábra Az adatbázis táblái közötti kapcsolatok

A menhely különböző folyamatainak zavartalan működéséhez elengedhetetlen volt egy adatbázis. A weboldal és az asztali alkalmazás is ezzel az adatbázissal dolgozik, adatfeltöltések, -törlések és -módosítások révén.

Az adatbázis egyik része a regisztrációt, bejelentkezést és a felhasználói adatok tárolását szolgálja. A másik része a menhelyi szolgáltatásokhoz szükséges adatokat tartalmazza.

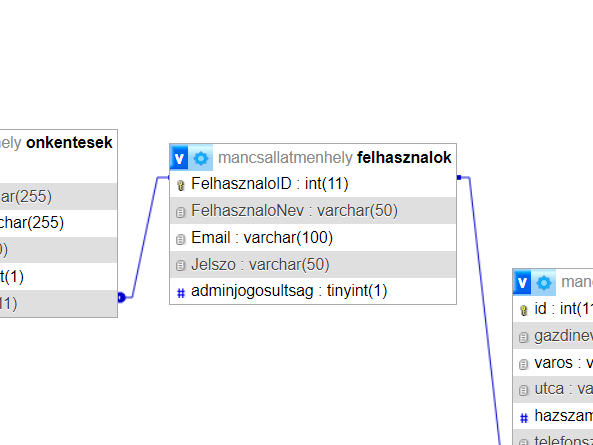
### A táblák szerkezete

Alkalmazott rövidítések:

PK: Primary Key – Elsődleges kulcs

FK: Foreign Key – Idegen kulcs

„felhasznalok” tábla

„FelhasznaloID”: a felhasználó azonosítója (szám, automatikusan generálódik, PK)

2. ábra "felhasznalok" tábla

„FelhasznaloNev”: felhasználónév (karakterek, a felhasználó adja meg)

„Email”: a felhasználó e-mail címe (karakterek, a felhasználó adja meg)

„Jelszo”: a felhasználó jelszava (karakterek, a felhasználó adja meg)

„adminjogosultsag”: a felhasználó adminjogosultsága (boolean, true: van, false: nincs, alapértelmezett beállítás: false)

„onkentesek” tábla

3. ábra "onkentesek" tábla

„ID”: az önkéntesek azonosítója (szám, automatikusan generálódik, PK)

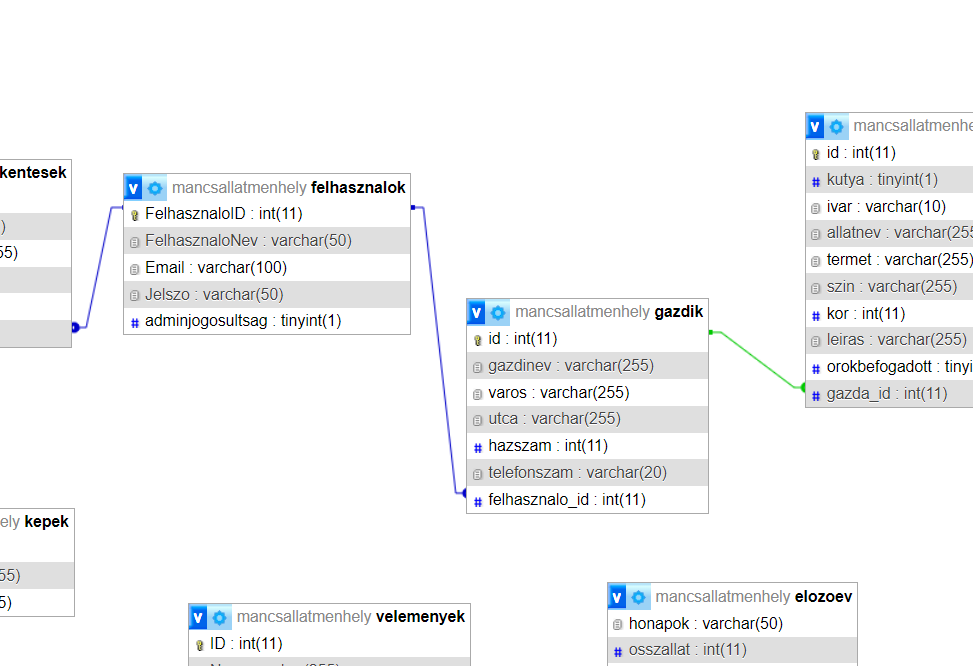
„onkentesnev”: az önkéntes neve (karakterek, a felhasználó adja meg)

„onkentesemail”: az önkéntes e-mail címe (karakterek, a felhasználó adja meg)

„telszam”: az önkéntes telefonszáma (karakterek, a felhasználó adja meg)

„kozepiskolas”: az önkéntes középiskolás-e (boolean, true: igen, false: nem, felhasználó adja meg)

„felhasznalo\_id”: az önkéntes felhasználói azonosítója (szám, FK)

„gazdik” tábla

4. ábra "gazdik" tábla

„id”: a gazda azonosítója (szám, automatikusan generálódik, PK)

„gazdinev”: a gazda neve (karakterek, a felhasználó adja meg)

„varos”: a gazda lakhelyének városa (karakterek, a felhasználó adja meg)

„utca”: a gazda lakhelyének utcája (karakterek, a felhasználó adja meg)

„hazszam”: a gazda lakhelyének házszáma (szám, a felhasználó adja meg)

„telefonszam”: a gazda telefonszáma (karakterek, a felhasználó adja meg)

„felhasznalo\_id”: a gazda felhasználói azonosítója (szám, FK)

„allatok” tábla

5. ábra "allatok" tábla

„id”: az állat azonosítója (szám, automatikusan generálódik, PK)

„kutya”: kutya vagy macska-e az állat (boolean, true: kutya, false: macska, alapértelmezett beállítás: false, admin adja meg)

„ivar”: az állat ivara (karakterek, az admin adja meg)

„termet”: az állat termete (karakterek, az admin adja meg)

„szin”: az állat színe (karakterek, az admin adja meg)

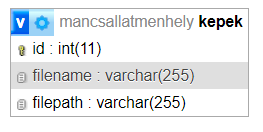
„kor”: az állat kora (szám, az admin adja meg)

„leiras”: az állat leírása (karakterek, az admin adja meg)

„orokbefogadott": az állat örökbefogadott-e (boolean, true: igen, false: nem, alapértelmezett beállítás:false)

„kep\_id”: az állathoz tartozó kép azonosítója (szám, FK)

„gazda\_id”: az állat gazdájának azonosítója (szám, FK)

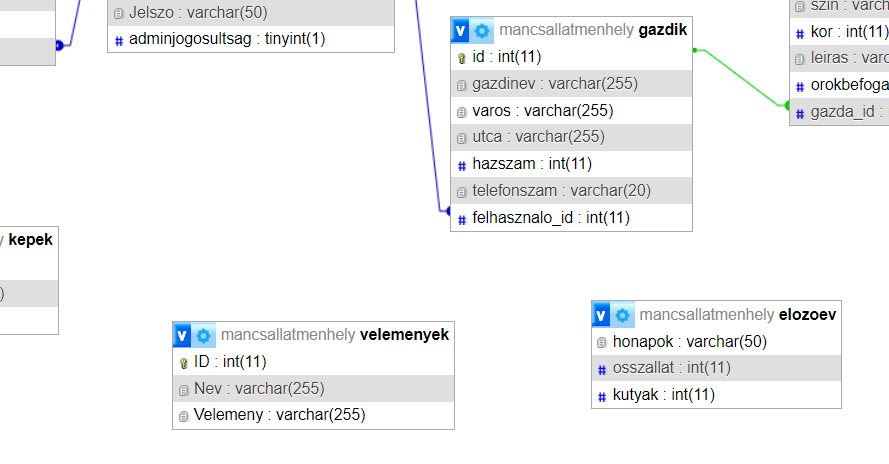
„kepek” tábla

„id”: a kép azonosítója (szám, automatikusan generálódik, PK)

„filename”: a kép file neve (karakterek)

„filepath”: a kép file útja (karakterek)

6. ábra "kepek" tábla

„velemenyek” tábla

„id”: a vélemény azonosítója (szám, automatikusan generálódik, PK)

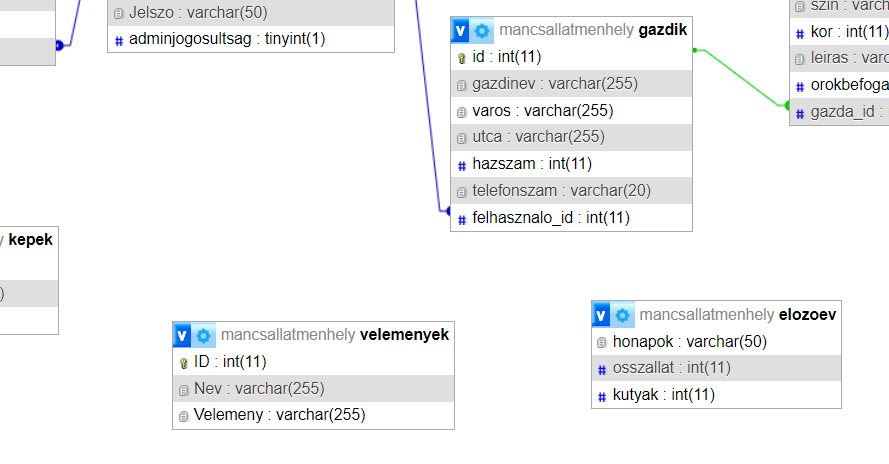
„Nev”: a vélemény írójának neve (karakterek, felhasználó adja meg)

„Velemeny”: a beküldött vélemény (karakterek,

7. ábra "velemenyek" tábla

felhasználó adja meg)

„elozoev” tábla

„honapok”: hónap neve (karakterek, felhasználó adja meg)

8. ábra "elozoev" tábla

„osszallat”: összes örökbefogadott állat (szám, admin adja meg)

„kutyak”: összes örökbefogadott kutya (szám, admin adja meg)

### Megkötések a táblák között

* Az „onkentesek” tábla ’felhasznalo\_id’ mezőjének referenciája a „felhasznalok” tábla ’FelhasznaloId’ mezője
* A „gazdik” tábla ’felhasznalo\_id’ mezőjének referenciája a „felhasznalok” tábla ’FelhasznaloId’ mezője
* Az „allatok” tábla ’gazda\_id’ mezőjének referenciája a „gazdik” tábla ’id’ mezője
* Az „allatok” tábla ’kep\_id’ mezőjének referenciája a „kepek” tábla ’id’ mezője

Az adatbázis bemutatása után következzenek az alkalmazások, amelyek az adatbázisból szerzik be a működésükhöz szükséges adatokat.

Az adatbázis szerver.js-sel való összekötése és a használt modulok:

const bodyParser = require('body-parser');

const express = require('express');

const multer = require('multer');

const path = require('path');

const fs = require('fs');

const app = express();

const cors = require('cors');

app.use(cors());

const mysql = require('mysql');

const kapcsolat = ()=>{

    return mysql.createConnection({

        host: 'localhost' ,

        user: 'root' ,

        password: '' ,

        database: "mancsallatmenhely"

    })

}

Szerveroldali logika (Node.js és Express)

Az adatbázis kapcsolatot a kapcsolat() függvény hozza létre, amely egy MySQL adatbázis-objektumot ad vissza.

## A weboldal felhasználói része

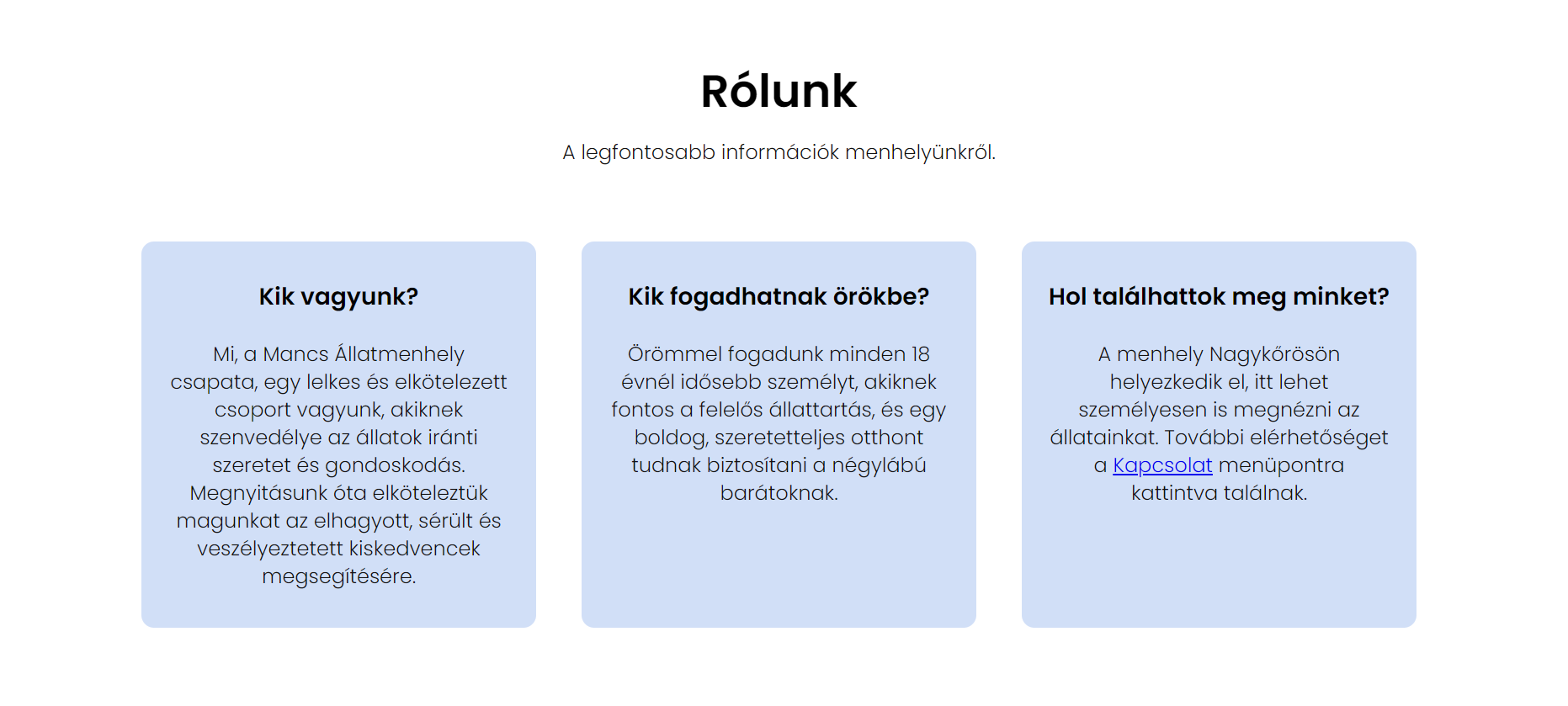
Weblapunk a Mancs Állatmenhelyet és annak működését, állatait, tevékenységeit mutatja be. A készítés során törekedtünk arra, hogy mindenki számára áttekinthető, felhasználóbarát felületet készítsünk.

Weboldalunk három fő részből áll:

1. Regisztráció nélkül elérhető oldalak: A nem regisztrált felhasználók számára elérhető oldalak között az elsődleges a főoldal, a többi bárki számára látható oldalt pedig könnyedén megnyithatjuk a logó melletti navigációs menüből.
2. Regisztrációval, bejelentkezéssel elérhető felhasználói oldalak: Ezek a felületek olyan funkciókat kínálnak, amelyek személyes adatokat igényelnek és kezelnek. Ezeket az oldalakat kizárólag bejelentkezés után lehet elérni.
3. Bejelentkezéssel elérhető admin oldal: Ezeket az oldalakat csak az adminjogosultsággal rendelkező felhasználók érhetik el.

### Főoldal

9. ábra index.html

 A kezdőoldalon található a menhely neve és szlogene. Ezután a főoldal három szekcióra van osztva. A „Tudj meg Rólunk többet!” gomb az első, „Rólunk” című szekcióra visz.

10. ábra "Rólunk” szekció

A következő szekció a menhely alapelveit mutatja be.

11. ábra "Alapelvek" szekció

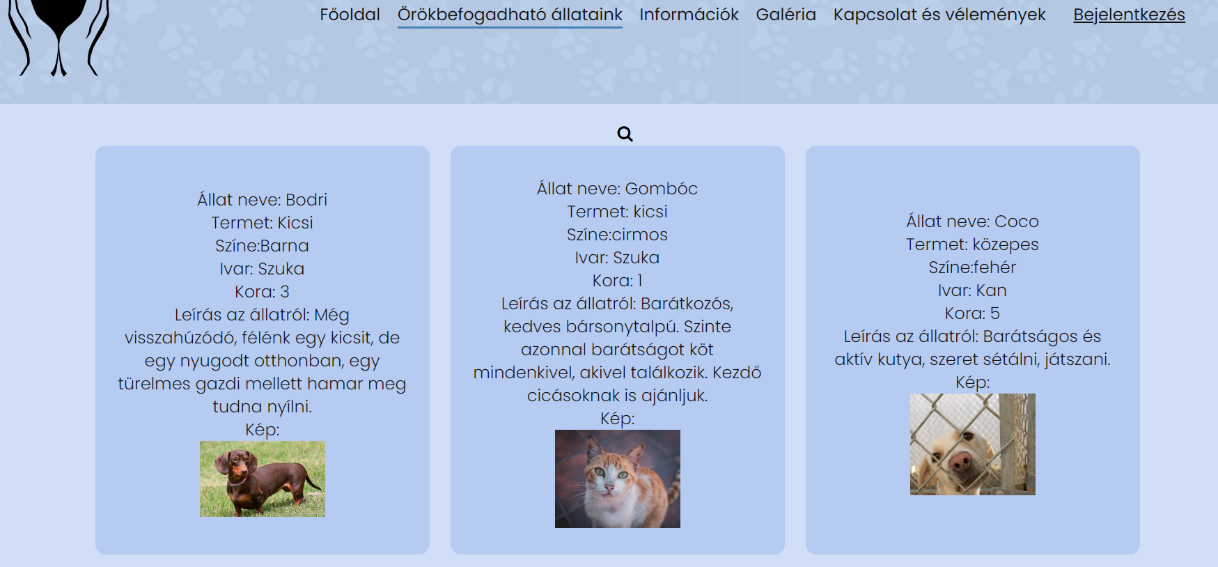
Az utolsó részben egy diagram található, amely a menhely féléves örökbefogadási statisztikáját mutatja be. Az oszlopra húzva a kurzort, megjelennek a részletes adatok.

12. ábra "diagram" szekció

Minden oldal alján megtalálható a footer:

13. ábra footer

### Örökbefogadható állataink

Erre az oldalra kattintva a felhasználó a jelenleg örökbefogadható állatokról talál adatokat. A keresés ikonra kattintva lehetséges a szűrés fajta, ivar és kor alapján.

15. ábra örökbefogadható állataink szűrése

14. ábra allataink.html

Azok az állatok eltűnnek a keresés gombra kattintva, amelyekre a megadott feltételek nem tartoznak.

### Információk

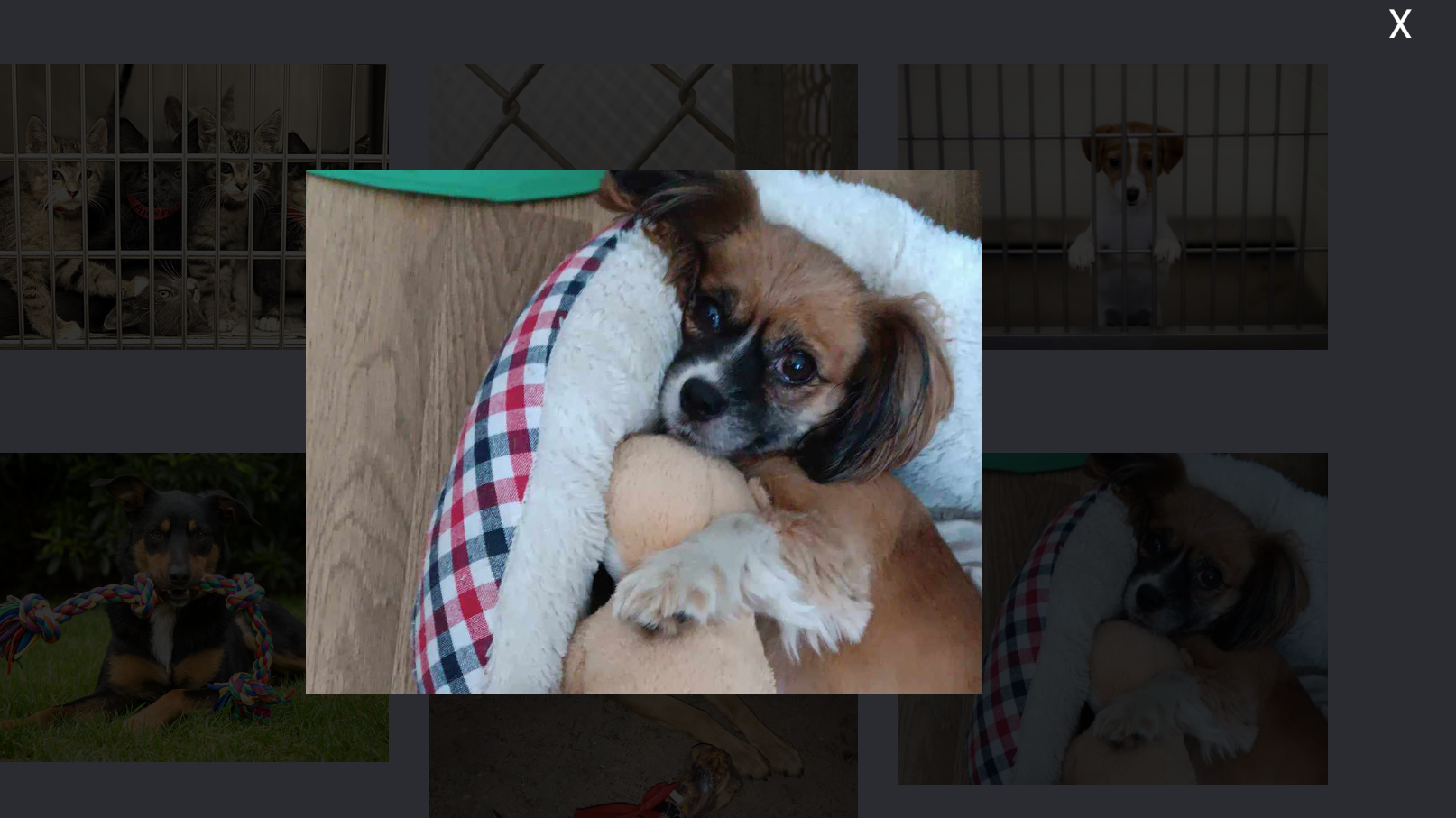
Ezen az oldalon található minden, az örökbefogadáshoz szükséges információ.

16. ábra informaciok.html

### Galéria

A galéria oldalon az állatokról találhatók képek. A képekre kattintva nagyobb méretben is meg lehet tekinteni őket.

17. ábra galeria.html

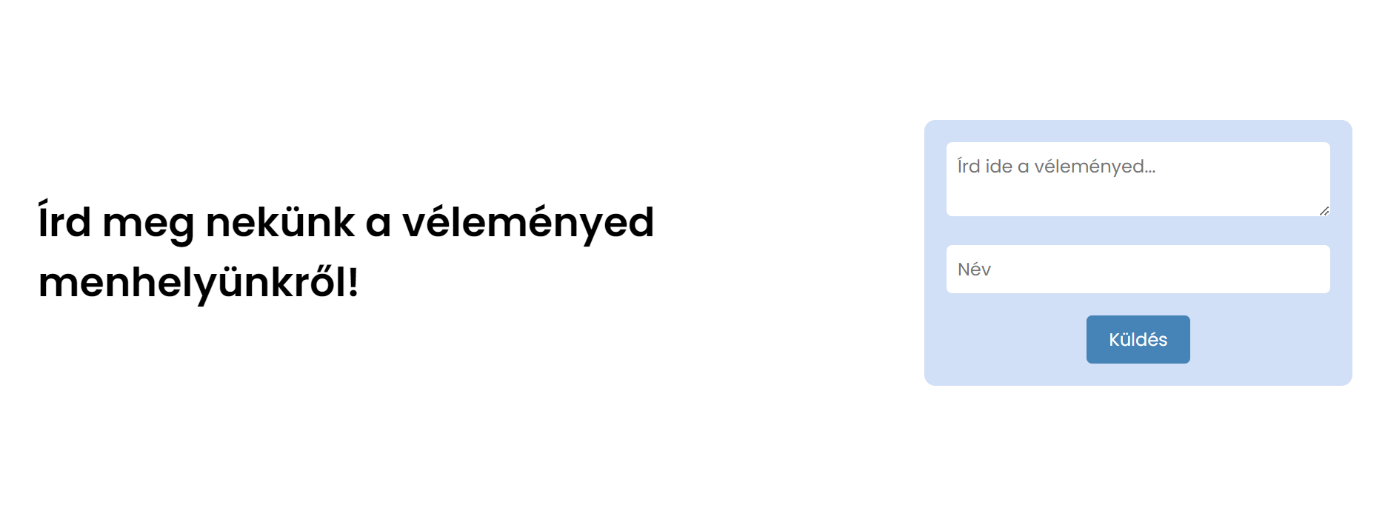


18. ábra nagyított kép

### Kapcsolat és vélemények

Ez az oldal két részből áll. Az egyik részén a menhely elérhetősége van, fiktív adatokkal dolgoztunk: 

19. ábra kapcsolat.html

A másik részén lehetőség van véleményt írni, és a három legfrissebb bejegyzés meg is található az oldal alján.

21. ábra megjelenített vélemények

20. ábra véleményírás

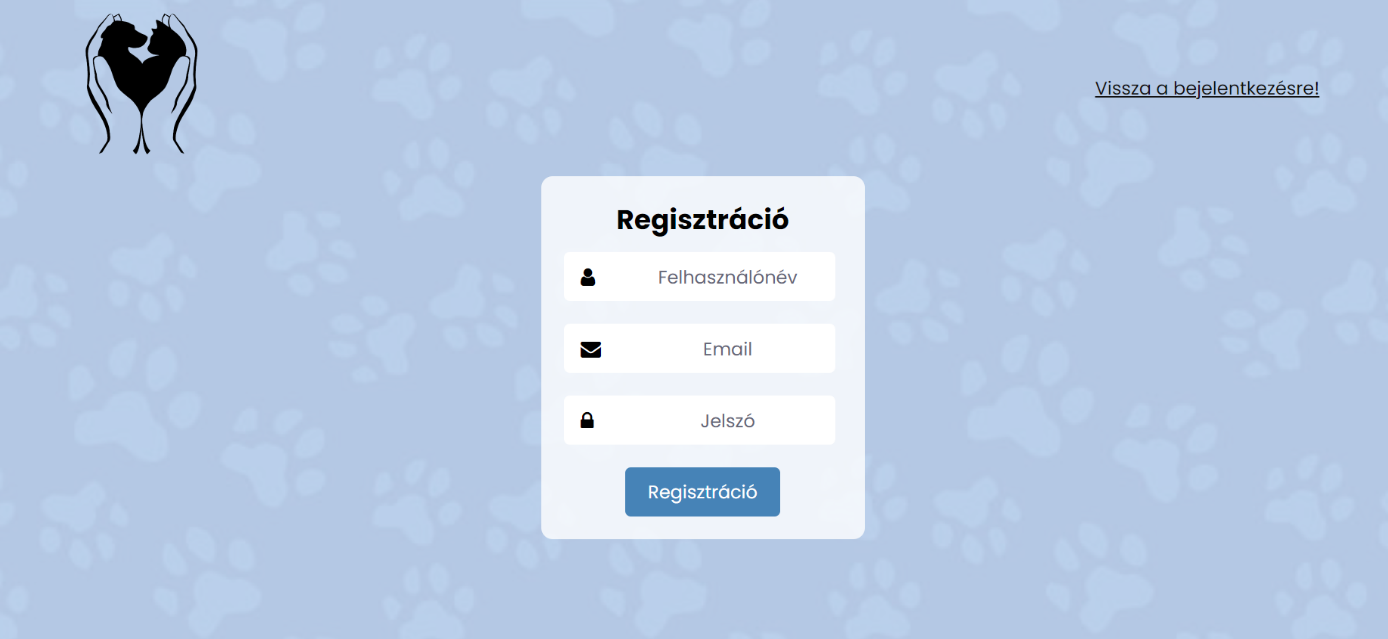
Az eddigi oldalakon is látható volt, hogy a kék-fehér színeket váltogattuk háttérként, hogy jól elkülönüljenek egymástól a különböző szekciók.

Ezek voltak azok az oldalak, amelyeknek eléréséhez nem szükséges regisztráció.

### Belépés és regisztráció

22. ábra login.html

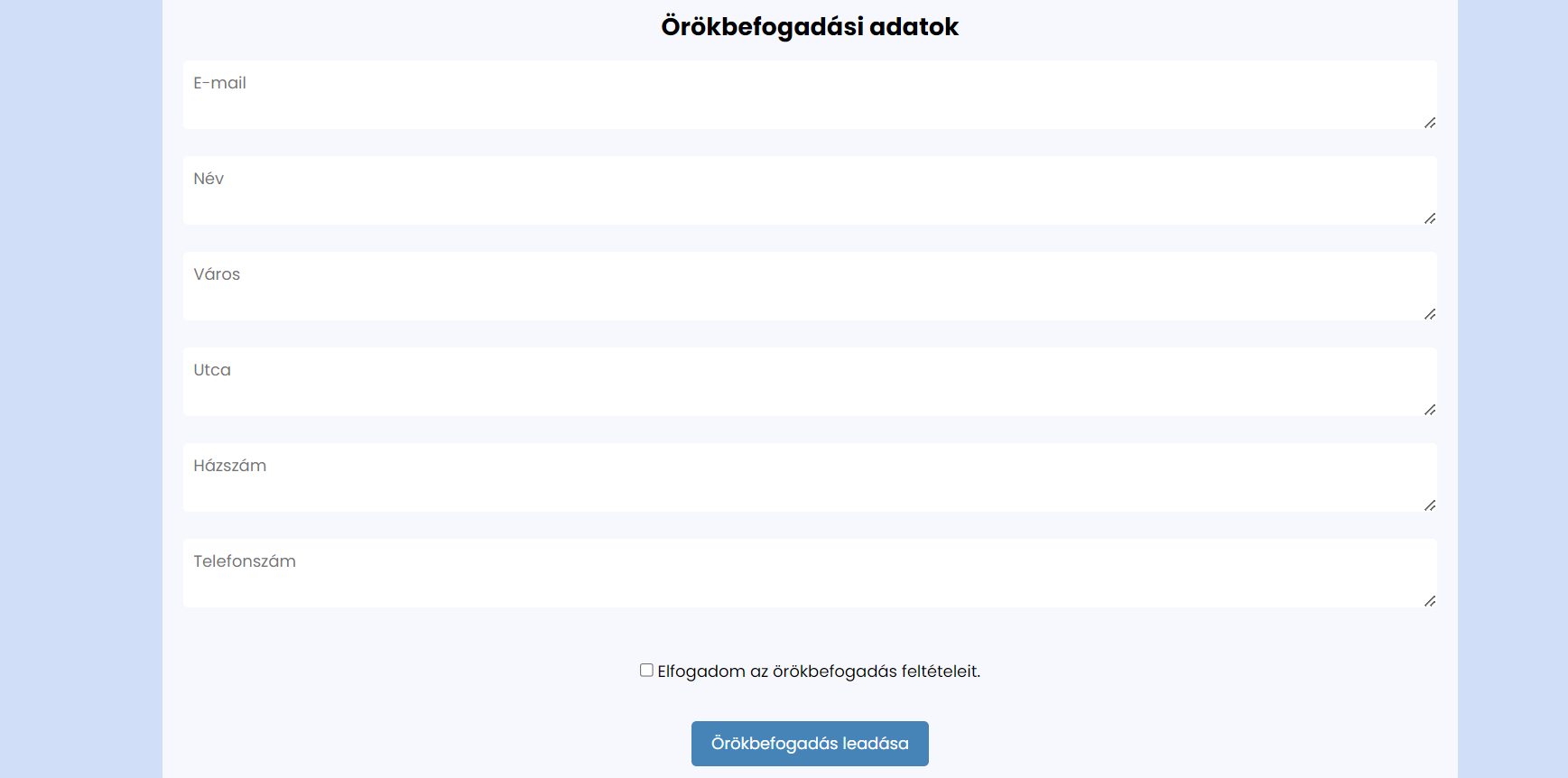
A navigációs menü „Bejelentkezés” gombjára kattintva nyílik meg a belépési felület.

A felhasználók csak regisztrációt követően van lehetőségük a bejelentkezésre, a felületet itt a „Regisztrálok” gombra kattintva érhetik el.

23. ábra regisztracio.html

Sikeres regisztrációt követően az oldal automatikusan visszairányít a bejelentkező felületre.

### Örökbefogadás

Bejelentkezés után az örökbefogadási oldal jelenik meg. Itt az örökbefogadni vágyó felhasználóknak először választaniuk kell egy állatot, a listában mindig az aktuálisan örökbefogadható állatokat találják. Ezután a személyes adatok megadása következik. Fontos, hogy az örökbefogadási űrlap csak regisztrált e-mail címet, illetve a házszám és telefonszám mezőkben csak számot fogad el! Az űrlap végén be kell jelölni, hogy „Elfogadom az örökbefogadás feltételeit”, a feltételek az Információk oldalon találhatók.

24. ábra orokbefogadas.html

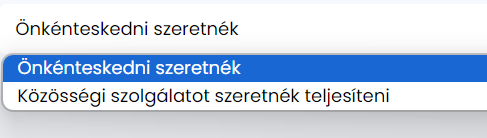
25. ábra adatok megadása

Sikeres örökbefogadás esetén a következő üzenet jelenik meg a felhasználónak, ezt igyekeztünk minden olyan oldalon megjeleníteni, ahol felhasználói interakció történt:

### Segítenék

26. ábra visszajelzés

A „Segítenék” oldalon a felhasználók jelentkezhetnek önkénteskedésre, a középiskolás korú diákok az ötven óra kötelező közösségi szolgálatot is eltölthetik menhelyünkön. Az örökbefogadáshoz hasonlóan, itt is csak regisztrált e-mail címmel lehet jelentkezni.

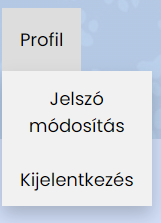
Az utolsó mezőben lehet kiválasztani, hogy ki az, aki önkéntes munkát és ki az, aki közösségi szolgálatot szeretne teljesíteni:

27. ábra segitenek.html

28. ábra önkénteskedés

### Profil

A következő elem a navigációs menüben a „Profil”, ami egy legördülő menü. Itt két választási lehetősége van a felhasználónak: jelszómódosítás és kijelentkezés.



29. ábra profil

A jelszómódosításra kattintva ez az űrlap jelenik meg:

30. ábra jelszomodositas.html

Sikeres és sikertelen jelszómódosítás esetén a fentiekhez hasonlóan, itt is üzenetet kap a felhasználó.

A kilépésre kattintva a felhasználó a Főoldalra kerül átirányításra.

## A weboldal admin része

Amikor a bejelentkezési folyamat olyan adatokkal megy végbe, amikhez tartozik adminjogosultság, akkor egy tartalmilag teljesen más felület jelenik meg a felhasználó előtt.

### Galéria szerkesztés

31. ábra galeriaszerkesztes.html

Ez a weboldal admin részének kezdőfelülete. Itt van lehetőség képet feltölteni, a kép ezután megjelenik a Galéria oldalon. Természetesen itt is kap az admin visszajelzést, hogy a művelet sikeres volt-e.

### Örökbefogadható állataink szerkesztése

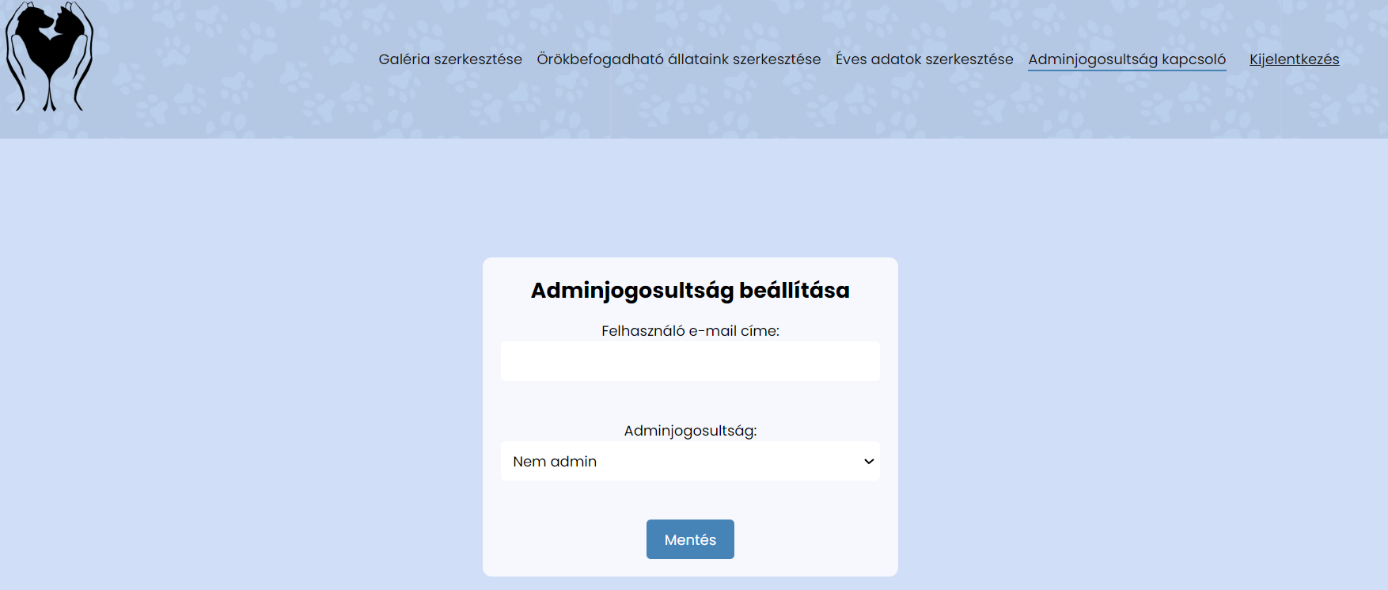
Itt a fentiekhez hasonló kinézetű űrlap jelenik meg. Ki kell választani az állat tulajdonságait, elküldés után pedig képet is lehet az állathoz rendelni, egy olyan felületen, mint ami a galeriaszerkesztes.html-en volt.

### Éves adatok szerkesztése

Ezen az űrlapon lehet a diagram adatait szerkeszteni, a kinézete a fenti űrlapokkal azonos.

Amennyiben az admin véletlen nagyobb értéket adott meg a havi örökbefogadott kutyáknak, mint a havi összes örökbefogadott állatnak, nem fogja elküldeni az űrlap az adatokat.

### Adminjogosultság kapcsoló

Az adminjogosultság kapcsoló oldalon bármelyik felhasználó adminjogosultságát át lehet állítani.

32. ábra adminjogosultság beállítása

### Kijelentkezés

A kijelentkezés gombra kattintva az alkalmazás a bejelentkezés oldalra irányít vissza.

A sima felhasználó és admin felületre is érvényes az, hogy üres mezőt nem lehet beküldeni, minden mezőnek adni kell értéket.

Az oldalak egyik közös jellemzője az, hogy mobilnézetben a navigációs menü kinyitható/becsukható menüvé alakul át.



34. ábra menü kinyitva

33. ábra menü becsukva

Ezek voltak azok az oldalak, amelyeknek eléréséhez szükséges a regisztráció.

## A felhasználói oldal fejlesztése

Habár sok különböző tartalmú HTML fájlunk van, igyekeztünk egy keretet adni nekik, hogy az egységességet megőrizzük, függetlenül attól, hogy melyik felületen van a felhasználó.

Így a kódunkban vannak olyan részek, amelyek mindegyik oldalon megtalálhatók.

### Közös részek

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Mancs Állatmenhely</title>

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

    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@100;200;300;400;600;700&display=swap" rel="stylesheet">

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">

</head>

<body>

<section class="header">

       <nav>

        <a href="index.html"><img src="logo.png" id="logo"></a>

        <div class = "pages" id="links">

<i class="fa fa-times" id=".fa-times"></i>

<ul>

                <li><a href="index.html" id="fooldal">Főoldal</a></li>

                <li><a href="allataink.html" id="allataink">Örökbefogadható állataink</a></li>

                <li><a href="informaciok.html" id="informaciok" > Információk</a></li>

                <li><a href="galeria.html" id="galeria">Galéria</a></li>

                <li><a href="kapcsolat.html" id="kapcsolat">Kapcsolat és vélemények</a></li>

                <li><a href="login.html" id="bejelentkezes" > Bejelentkezés</a></li>

</ul>

        </div>

<i class="fa fa-bars" id=".fa-bars"></i>

       </nav>

Egy HTML weboldal kötelező elemei: <!DOCTYPE html> tag, <html> nyitó és </html> záró tag, <head></head> fejrész és <body></body> törzsrész.

A fejrészben tudjuk megadni az oldalra vonatkozó beállításokat, például a karakterkódolást, az oldal címét és itt tudjuk belinkelni az oldalhoz szükséges stíluslapokat, a mi esetünkben külső hivatkozásokat is.

Mi a betűtípushoz és az ikonokhoz használtunk külső hivatkozásokat.

Ahhoz, hogy mobilnézetben lehetséges legyen a menü megjelenítése, az oldalakon ezeket az ikonokat alkalmaztuk:

<i class="fa fa-times" id=".fa-times"></i>

<i class="fa fa-bars" id=".fa-bars"></i>

A header/navbar rész is a legtöbb oldalon ugyanaz a felépítésű, a szövege változik attól függően, hogy az oldal három része közül melyiken vagyunk.

A footer rész tartalma szintén közös az összes oldalon, a színe változik a bejelentkezés és regisztráció oldalakon.

<footer class="footer" >

       <div class="footer-content">

           <p>Mancs Állatmenhely</p>

           <p>© 2024 Minden jog fenntartva.</p>

       </div>

</footer>

Az egységességet segítik a JavaScript és CSS beállítások is.

Az oldal három részének, a bejelentkezés nélkül elérhető oldalaknak és a bejelentkezést igénylő felhasználói- és adminfelületnek a közös tulajdonsága, hogy mindhárom felületnek az oldalait összefogja egy-egy univerzális JS file.

A bejelentkezés nélküli oldalak végén a script.js file, a felhasználói felület végén a userscript.js, az admin felület végén pedig az adminscript.js szerepel tag-ként.

Mindhárom JavaScript file-ra elmondható, hogy a felépítésük, funkciójuk a következő:

var navLinks = document.getElementById("links");

var menuButton = document.querySelector(".fa-bars");

var closeButton = document.querySelector(".fa-times");

menuButton.addEventListener("click", function() {

    navLinks.classList.add("active");

});

closeButton.addEventListener("click", function() {

    navLinks.classList.remove("active");

});

var currentPath = window.location.pathname.split("/").pop();

var pageIds = {

    "orokbefogadas.html":"orokbefogadas",

    "segitenek.html":"segitenek"

};

var activeLinkId = pageIds[currentPath];

 if (activeLinkId) {

     var activeLink = document.querySelector("#" + activeLinkId);

     if (activeLink) {

         activeLink.parentNode.classList.add("active");

     }

 }

Az első részben a menümegjelenítés és bezárás szerepel, a második részben pedig az, hogy a navigációs menüben az éppen megnyitott oldal neve legyen aláhúzva, ezzel is biztosítva az oldal átláthatóságát.

Ehhez CSS szabályokat is alkalmaztunk:

.pages{

    position:absolute;

    background: #4683b7;

    height: 100vh;

    width:100%;

    top: 0;

    right:0;

    text-align: left;

    z-index: 2;

    visibility: hidden;

}

.pages.active{

    visibility: visible;

}

Itt az látszik, hogy a „pages” osztály alap beállítása az, hogy „hidden”, azaz rejtett legyen. Amikor viszont megkapja az „active” osztályt, „visible”, azaz látható lesz.

Az oldal reszponzívvá tételéhez ’@media’ tageket használtunk.

### index.html felépítése

Az index.html négy szekcióból áll, az első a fentebbi képen látható header, amely a fejlécet, logót, kezdőoldalt jeleníti meg az oldalon.

A második és a harmadik felépítése hasonló, a section-ben vannak <div>-ek, ezek a <h1>, <h3>, <p> tageket tartalmazzák, amiket a szöveg megjelenítésére használtumk. Képeket az <img> tag segítségével szúrtunk be.

<section id ="about" class="about">

    <h1>Rólunk</h1>

    <p>A legfontosabb információk menhelyünkről.</p>

    <div class ="row">

        <div class = "about-col">

            <h3>Kik vagyunk?</h3>

<section class="alapelvek">

    <h1>Legfontosabb alapelveink:</h1>

    <div class="row">

       <div class="alapelvek-col">

            <img src="kutya1.png">

            <div class="layer">

                <h3>Állataink jóllétének elsődlegessége</h3>

            </div>

       </div>

Az utolsó section-ben helyezkedik el a diagram, a weboldalunk React része, ezért a HTML kódban ebben csak a diagram címe és a root található.

<section class="react">

    <h1>Az örökbefogadott kutyák száma az összes örökbefogadott állathoz képest, 2023 első félévében</h1>

    <div id="root"></div>

</section>

React kódunkat a diagram.js komponensben írtuk meg. A diagram elkészítéséhez a **Recharts** nevű könyvtárat használtuk. Az oszlopdiagram az értékeit az adatbázisunk ’elozoev’ tábla rekordjaiból nyeri ki.

A fetchData függvény és a useEffect hook felel az adatok lekérdezéséért a szerverről és azok formázásáért. A szerveroldali kód a '/data' végponton keresztül lekéri az adatokat az ’elozoev’ táblából.

app.get('/data', (req, res) => {

    const connection = kapcsolat();

    connection.connect();

    const query = `SELECT honapok, osszallat, kutyak FROM elozoev`;

    connection.query(query, (error, results) => {

        if (error) {

            console.error('Hiba az adatok lekérdezése során:', error);

            res.status(500).json({ error: 'Hiba az adatok lekérdezése során' });

        } else {

            console.log('Adatok sikeresen lekérdezve');

            res.status(200).json(results);

        }

        connection.end();

    });

});

import React, { useState, useEffect } from 'react';

import { BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend } from 'recharts';

function Diagram() {

  const [data, setData] = useState([]);

  useEffect(() => {

    fetchData();

  }, []);

  const fetchData = async () => {

    try {

      const response = await fetch('http://localhost:8080/data');

      if (!response.ok) {

        throw new Error('Hiba az adatok lekérdezése során');

      }

      const jsonData = await response.json();

      const formazottData = jsonData.map(item => ({

        name: item.honapok,

        uv: item.osszallat,

        pv: item.kutyak

      }));

      setData(formazottData);

    }

Ezen kívül a komponensben megtalálhatók még a diagram formázásért felelős kódrészek, illetve a jelmagyarázat szövege is.

Az index.html végén a footer és a script.js tag szerepel.

Ahogy az a képeken is látszott, a section, div tagek mind kaptak egy osztály azonosítót is. Ez az összes oldalon így található, hiszen ezeken a class vagy id értékeken keresztül tudjuk szabályozni az oldal stílusát CSS-ben.

Példa:

* React beállításai

.react{

    width: 100%;

    margin: auto;

    text-align: center;

    padding-top: 50px;

    background-color: #fff;

    box-sizing: border-box;

    height:500px;

}

### allataink.html felépítése

Az állatok egy wrapperben jelennek meg, amit adatbázisból töltünk fel.

<section class="orokbefogadhato-wrapper">

</section>

Az allataink.js kódban először lekezeltük az állatok adatait. A szerver oldai ’/kereses’ végponton kértük ezeket le. Ahhoz, hogy a képeket megjelenítsük, kihasználtuk a ’kepek’ és az ’allatok’ tábla közötti adatbázis kapcsolatot.

'SELECT \* FROM allatok as allatok left join kepek as kepek on allatok.kep\_id = kepek.id';

Ezután a JavaScript kódban hozzáadtuk az állat összes lekérdezett adatához a képeket is.

text.innerHTML = `Állat neve: ${allat.allatnev}<br>Termet: ${allat.termet}<br>Színe:${allat.szin} <br>Ivar: ${allat.ivar}<br>Kora: ${allat.kor}<br>Leírás az állatról: ${allat.leiras}<br>Kép:<br><img src="http://localhost:8080/images/${allat.filename}" alt="no picture found"/>`;

### informaciok.html felépítése

Az Információk oldalnak ugyanaz a felépítése, mint az index.html-en is megtalálható

„about” section-nek

### galeria.html felépítése

A közös elemek itt is megtalálhatók. Ezen kívül az oldal két div-et tartalmaz:

<div id="gallery" class="gallery">

</div>

<div class="wholeimg" id="wholeImgBox">

<img src="images/img1.jpg" id="wholeImg">

<span onclick="closeWholeImg()">X</span>

</div>

Ezek a div-ek az adatbázisból töltődnek fel, a képek megjelenítése pedig JavaScriptben, a „galeria-script.js” file-ban, a következő módon történik:

var wholeImgBox = document.getElementById("wholeImgBox");

var wholeImg = document.getElementById("wholeImg");

function openWholeImg(pic) {

    wholeImgBox.style.display = "flex";

    wholeImg.src = pic;

}

function closeWholeImg() {

    wholeImgBox.style.display = "none";

}

Ez teszi lehetővé CSS beállításokkal azt, hogy nagyban is meg lehessen nézni a képeket.

A képeket az adatbázisból jeleníti meg az oldal, a következő módon:

`SELECT \* FROM kepek`;

        const images = results.map(image => {

            return {

                filename: image.filename,

                filepath: path.join(\_\_dirname, 'public', 'images', image.filename)

            };

        });

A kód lekéri az adatbázisból a képek nevét, elérési útvonalát. Ehhez az express.static( ) függvényt használtuk, ahol megadtuk a képek könyvtárját.

app.use("/images", express.static('public/images'));

A fetchImages aszinkron függvény a böngésző betöltésekor automatikusan lekéri a képeket. Amennyiben a lekérés sikeres, a válaszként kapott képeket megjeleníti a galériában, appendChild segítségével. Ezután hozzáad mindegyik képhez egy kattintás eseményt, amelynek hatására az „openWholeImg”, azaz a teljes méretű kép megnyílik. Hiba esetén a konzolon hibaüzenet jelenik meg.

async function fetchImages() {

    try {

        const response = await fetch('http://localhost:8080/kepek');

        if (!response.ok) {

            throw new Error('Nem sikerült lekérni a képeket.');

        }

        const images = await response.json();

        const gallery = document.getElementById('gallery');

        images.forEach(image => {

            const img = document.createElement('img');

            img.src = `http://localhost:8080/images/${image.filename}`;

            img.alt = image.filename;

            img.classList.add('thumbnail');

            img.addEventListener('click', () => openWholeImg(`http://localhost:8080/images/${image.filename}`));

            gallery.appendChild(img);

        });

### kapcsolat.html felépítése

A közös elemeken kívül tartalmaz egy containert, amiben az elérhetőségek leírása és egy térkép van. Az oldal interaktív részének kódja:

<section class="velemenyek-wrapper">

    <h1>Írd meg nekünk a véleményed menhelyünkről!</h1>

    <div class="velemeny-form">

        <form id="velemeny-form" onsubmit="event.preventDefault(); Velemeny()">

            <textarea id="velemeny" placeholder="Írd ide a véleményed..." required></textarea>

            <input type="text" id="nev" placeholder="Név" required>

            <button type="submit" class="btn">Küldés</button>

        </form>

    </div>

</section>

<div class="cim">

<h1>Ti írtátok</h1>

</div>

<section class="tiirtatok-wrapper">

</section>

A „velemenyek-wrapper” a véleményküldési felületet foglalja magában.

A „tiirtatok-wrapper” pedig a beküldött véleményeket jeleníti meg.

Az oldal tartalmaz a „script.js” -en kívül egy másik JavaScript file-t is, a „kapcsolat-script.js” -t.

Amikor az oldal betöltődik, az onload eseménykezelő a „LegfrissebbVelemenyek” függvényt hívja meg. Ez a függvény aszinkron módon fetch kérést indít a szerver felé a legfrissebb vélemények lekérdezésére. Ha a kérés sikeres, dinamikusan megjeleníti a véleményeket az oldalon, a „tiirtatok-wrapper” section-ben. A vélemények megjelenítéséhez használt diveket, és azok osztályait is itt hoztuk létre.

window.onload = function() {

    LegfrissebbVelemenyek();

}

        const tiirtatok = document.querySelector('.tiirtatok-wrapper');

        tiirtatok.innerHTML = '';

        velemenyek.forEach(velemeny => {

            const velemenyElem = document.createElement('div');

            velemenyElem.classList.add('velemeny');

            const nevElem = document.createElement('div');

            nevElem.classList.add('nev');

            nevElem.textContent = velemeny.Nev;

            const szovegElem = document.createElement('div');

            szovegElem.classList.add('szoveg');

            szovegElem.textContent = velemeny.Velemeny;

            velemenyElem.appendChild(szovegElem);

            velemenyElem.appendChild(nevElem);

            tiirtatok.appendChild(velemenyElem);

        });

A „/legfrissebb\_velemenyek” végpont:

const selectQuery = `SELECT \* FROM Velemenyek ORDER BY ID DESC LIMIT 3`;

Ez egy egyszerű lekérdezés, ami a vélemények tábla rekordjait kéri le, az azonosítójuk szerinte rendezve. Ez a rendezés azért van, mert szerettük volna, ha mindig csak a három legfrissebb véleményt jeleníti meg az oldal.

Az oldal másik része a véleményküldési lehetőség.

Amikor a felhasználó kitölti az űrlapot és elküldi a véleményét, a „Velemeny” nevű függvény fut le.

A szerver oldali kód fogadja a kéréseket a /velemenyek végpontra, majd az adatokat beilleszti az adatbázisba.

async function Velemeny() {

    const nev = document.getElementById('nev').value;

    const velemeny = document.getElementById('velemeny').value;

    try {

        const response = await fetch('http://127.0.0.1:8080/velemenyek', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({ nev:nev, velemeny:velemeny })

        });

    const insertQuery = `INSERT INTO Velemenyek (Nev, Velemeny) VALUES  ("${nev}", "${velemeny}")`;

Ezek voltak a regisztrációt nem igénylő oldalak felépítései.

### login.html és regisztracio.html felépítése

A közös részektől eltérően, ehhez a két oldalhoz másik CSS stíluslap van csatolva, „style\_login.css”, illetve a JavaScriptük is más, „login-script.js”.

A header szekció felépítése is különbözik mindkét oldalnál:

 <nav>

    <a href="index.html"><img src="logo.png"></a>

    <div class = "gomb" id="links">

        <ul>

            <li><a href="login.html">Vissza a bejelentkezésre!</a></li>

        </ul>

    </div>

</nav>

Itt csak egy gomb található a navigációs menüben, így persze menü sem volt szükséges ezeknél az oldalaknál.

Az oldal lényegi részének felépítése is ugyanaz mindkét oldalnál:

<section class="wrapper">

    <div class = "login-box">

        <h2>Regisztráció</h2>

            <div class ="input-box">

                <i class="fa fa-user" aria-hidden="true"></i>

                <input type="text" id="FelhasznaloNev" name="felhasznalonev" required>

                <label for="FelhasznaloNev">Felhasználónév</label>

            </div>

            <div class ="input-box">

                <i class="fa fa-envelope" aria-hidden="true"></i>

                <input type="email" id="Email" name="email" required>

                <label for="Email">Email</label>

            </div>

            <div class="input-box">

                <i class="fa fa-lock" aria-hidden="true"></i>

                <input type="password" id="Jelszo" name="jelszo" required>

                <label for="Jelszo">Jelszó</label>

            </div>

            <button type="submit" class="btn" onclick="Regisztracio()" >Regisztráció</button>

    </div>

</section>

Mindkettő oldalnál egy űrlap található és mindkettő oldalnál az adatokat a gombra kattintással tudja a felhasználó elküldeni a szervernek.

async function Login() {

    const email = document.querySelector('input[type="email"]').value;

    const felhasznalonev = document.querySelector('input[type="text"]').value;

    const password = document.querySelector('input[type="password"]').value;

    const response = await fetch('http://127.0.0.1:8080/login', {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify({ email: email, felhasznalonev:felhasznalonev, password: password })

    });

    const data = await response.json();

    if (response.ok) {

        if (data.isAdmin) {

            window.location.href = "galeriaszerkesztes.html";

        } else {

            window.location.href = "orokbefogadas.html";

        }

    } else {

        alert(data.error);

    }

}

async function Regisztracio() {

    const FelhasznaloNev = document.querySelector('input[id="FelhasznaloNev"]').value;

    const Email = document.querySelector('input[id="Email"]').value;

    const Jelszo = document.querySelector('input[id="Jelszo"]').value;

    if (FelhasznaloNev === '' || Email === '' || Jelszo === '') {

        alert("Minden mező kitöltése kötelező.");

        return;

    }

    if (!Email.includes('@')) {

        alert("Érvénytelen e-mail cím.");

        return;

    }

    const response = await fetch('http://127.0.0.1:8080/regisztracio', {

    method: 'POST',

    headers: {

        'Content-Type': 'application/json'

    },

    body: JSON.stringify({ FelhasznaloNev:FelhasznaloNev, Email:Email, Jelszo:Jelszo })

    });

    const data = await response.json();

    console.log(data);

    if (response.ok) {

        console.log('Sikeres regisztráció');

        alert("Sikeres regisztráció.");

        window.location.href="login.html";

    } else {

        alert("A regisztráció sikertelen. Az e-mail cím vagy a felhasználónév már foglalt.");

        window.location.href="regisztracio.html";

    }

}

A „Login” és „Regisztracio” függvények a kliensoldali kódot tartalmazzák.

A két függvény működése:

„Login”: Első lépésként a felhasználótól az e-mail címét, felhasználónevét és jelszavát kéri le a HTML input mezőiből. Ezután egy POST kérést küld a szerver ’/login’ végpontra JSON formátumban. Ha sikeres választ kap, akkor továbbít annak megfelelően, hogy az „isAdmin” igaz-e, azaz attól függően, hogy van-e a felhasználónak adminjogosultsága.

„Regisztracio”: Első lépésként ugyanazt csinálja, mint a Login. Ezután ellenőrzi, hogy minden mező ki legyen töltve, valamint, hogy érvényes legyen az e-mail cím. A POST kéréssel elküldi ezeket az adatokat a /regisztracio címre. A választ feldolgozza, és mindkét esetben ad visszajelzést, akkor is, ha sikeres volt a regisztráció és akkor is, ha nem.

Szerver oldali kód:

    const cmd = `SELECT \* FROM Felhasznalok WHERE Email = "${email}" AND FelhasznaloNev = "${felhasznalonev}" AND Jelszo = "${password}"`;

    connection.query(cmd, (error, results, fields) => {

        if (error) {

            res.status(500).json({ "error": "Hiba a bejelentkezés során"

        } else {

            if (results.length > 0) {

                const user = results[0];

                if (user.adminjogosultsag === 1) {

                    res.status(200).json({ "message": "Sikeres bejelentkezés", "user": user, "isAdmin": true });

                } else {

                    res.status(200).json({ "message": "Sikeres bejelentkezés", "user": user, "isAdmin": false });

                }

            } else {

                res.status(401).json({ error: "Rossz felhasználónév vagy jelszó" });

            }

        }

    const checkQuery = `SELECT \* FROM Felhasznalok WHERE Email = "${Email}" OR FelhasznaloNev = "${FelhasznaloNev}"`;

    connection.query(checkQuery, (checkError, checkResults) => {

        if (checkError) {

            console.error('Hiba az ellenőrzés során:', checkError);

            res.status(500).json({ error: 'Hiba az ellenőrzés során' });

            connection.end();

            return;

        }

        if (checkResults.length > 0) {

            res.status(400).json({ error: 'Ezzel az email-címmel/felhasználónévvel már regisztráltak!' });

            connection.end();

            return;

        }

        const insertQuery = `INSERT INTO Felhasznalok (FelhasznaloNev, Email, Jelszo) VALUES ("${FelhasznaloNev}", "${Email}", "${Jelszo}")`;

        connection.query(insertQuery, (insertError, result) => {

            if (insertError) {

                console.error('Hiba a felhasználó adataival:', insertError);

                res.status(500).json({ error: 'Hiba a regisztráció során' });

                connection.end();

                return;

            }

            console.log('Felhasználó sikeresen regisztrálva');

            res.status(200).json({ message: 'Sikeres regisztráció' });

            connection.end();

        });

    });

});

A két kód működése:

’/login’ végpont: Az adatok megkapása után lekérdezést futtat a „Felhasznalok” táblában a kapott e-mail cím, felhasználónév és jelszó alapján. Az eredmény alapján válaszol: Ha talál felhasználót az adatbázisban ezekkel az adatokkal, megnézi, hogy az adminjogosultsága egyenlő-e eggyel, és kiírja, hogy az „isAdmin” értéke true. Ez azért szükséges, mert sikeres bejelentkezés után, ha az isAdmin true, akkor az admin felületre kerül a felhasználó, ha false, akkor a felhasználói felületre. Ha nem talál ilyen felhasználót, akkor azt írja ki, hogy hiba a bejelentkezés során.

’/regisztracio’ végpont: A POST kérésre válaszol, a ’/login’-hoz hasonlóan.

Először ellenőrzi, hogy az adott e-mail cím vagy felhasználónév már regisztrálva van-e.

Ha már van regisztrált felhasználó ezzel az e-mail címmel vagy felhasználónévvel, akkor hibát küld. Ha nincs ilyen felhasználó, akkor új felhasználót regisztrál az adatbázisba. Sikeres regisztráció esetén jelzi a felhasználónak a sikeres regisztrációt, hibás kérések esetén pedig hibaüzenetet küld. Mindkét végpont hibakezelést is tartalmaz, és az adatbáziskapcsolatokat a műveletek végén lezárja.

### orokbefogadas.html felépítése

A közös elemeken kívül a lényeges rész ebben a HTML-ben két form, ebből az egyik egy űrlap, ami nagyon hasonlít az eddigiekhez. A másik egy legördülő menü, amiből állatot lehet választani:

        <section class="felhasznalok-wrapper">

            <div class="admin-form">

                <label for="allatok">Válassz egy állatot:</label>

                <select id="allatok">

                </select>

            </div>

        </section>

Az oldal működését biztosító kód az „orokbefogadas-script.js”-ben található.

Két részből áll a JS:

const allatokDropdown = document.getElementById('allatok');

fetch('http://127.0.0.1:8080/allatok\_nevei')

    .then(response => response.json())

    .then(data => {

        data.forEach(allatnev => {

            const option = document.createElement('option');

            option.text = allatnev;

            option.value = allatnev;

            allatokDropdown.appendChild(option);

        });

    })

    .catch(error => console.error('Hiba történt az állatnevek lekérése során:', error));

async function orokbefogadas(event) {

    event.preventDefault();

    const feltetelekCheckbox = document.getElementById('feltetelek');

    if (!feltetelekCheckbox.checked) {

        alert("Az örökbefogadási feltételeket el kell fogadnia!");

        return;

    }

    const gazdiemail = document.getElementById('gazdiemail').value;

    const gazdinev = document.getElementById('gazdinev').value;

    const varos = document.getElementById('varos').value;

    const utca = document.getElementById('utca').value;

    const hazszam = document.getElementById('hazszam').value;

    const telefonszam = document.getElementById('telefonszam').value;

    const allatnev = allatokDropdown.value;

    if (gazdiemail === '' ||gazdinev === '' || varos === '' || utca === '' || hazszam === '' || telefonszam === '') {

        alert("Minden mező kitöltése kötelező.");

        return;

        }

        const response = await fetch('http://127.0.0.1:8080/orokbefogadas', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({gazdiemail:gazdiemail, gazdinev:gazdinev, varos:varos, utca:utca, hazszam:hazszam, telefonszam:telefonszam, allatnev:allatnev })

Az első rész segít abban, hogy a legördülő listában megjelenjenek az állatok nevei az

adatbázisból. Ennek a szerver oldali kódja a következő:

app.get('/allatok\_nevei', bodyParser.json(), (req, res) => {

    const connection = kapcsolat();

    connection.connect();

    const query = `SELECT allatnev FROM allatok WHERE orokbefogadott = 0`;

    connection.query(query, (error, results) => {

        if (error) {

            console.error('Hiba az adatok lekérdezése során:', error);

            res.status(500).json({ error: 'Hiba az adatok lekérdezése során' });

        } else {

            const allatNevek = results.map(row => row.allatnev);

            console.log('Állatok nevei sikeresen lekérdezve');

            res.status(200).json(allatNevek);

        }

        connection.end();

    });

});

Itt látható egy egyszerű lekérdezés az állatok nevére vonatkozóan, illetve egy szűrés arra, hogy biztosan csak azok az állatok jelenjenek meg a listában, akik még nincsenek örökbefogadva.

A második rész, az „orokbefogadas” aszinkron függvény kezeli a felhasználó által beküldött adatokat. Ezen kívül először is ellenőrzi, hogy a checkbox ki legyen pipálva, azaz a feltételek elfogadása teljesüljön, ha ez nem valósul meg, figyelmezteti a felhasználót.

Ezután az a feltétel következik, hogy nem lehet üres értékkel elküldeni az űrlapot.

A hibákat ez a függvény is kezeli.

A szerver oldali kód:

app.post('/orokbefogadas', bodyParser.json(), (req, res) => {

    const {gazdiemail, gazdinev, varos, utca, hazszam, telefonszam, allatnev } = req.body;

    const connection = kapcsolat();

    connection.connect();

    const checkUserQuery = `SELECT FelhasznaloID FROM Felhasznalok WHERE Email = "${gazdiemail}"`;

    connection.query(checkUserQuery, (error, results) => {

        if (error) {

            console.error('Hiba az ellenőrzés során:', error);

            res.status(500).json({ error: 'Hiba az ellenőrzés során' });

            connection.end();

            return;

        }

        if (results.length === 0) {

            res.status(404).json({ error: 'Nincs felhasználó ezzel az e-mail címmel' });

            connection.end();

            return;

        }

        const felhasznaloId = results[0].FelhasznaloID;

        const insertQuery = `INSERT INTO gazdik (gazdinev, varos, utca, hazszam, telefonszam, felhasznalo\_id) VALUES ("${gazdinev}", "${varos}", "${utca}", ${hazszam}, ${telefonszam}, ${felhasznaloId})`;

        connection.query(insertQuery, (error, result) => {

            if (error) {

                console.error('Hiba az örökbefogadás mentése során:', error);

                res.status(500).json({ error: 'Hiba az örökbefogadás mentése során' });

            } else {

                console.log('Örökbefogadás mentve');

                const newGazdiId = result.insertId;

                const insertAllatQuery =`UPDATE allatok SET gazda\_id = ${newGazdiId}, orokbefogadott = true WHERE allatnev = "${allatnev}"`;

                connection.query(insertAllatQuery, (error) => {

                    if (error) {

                        console.error('Hiba az állat táblába való beszúrás során:', error);

                        res.status(500).json({ error: 'Hiba az örökbefogadás mentése során' });

                    } else {

                        console.log('Állat gazdája frissítve');

                        res.status(200).json({ message: 'Örökbefogadás mentve' });

                    }

                });

            }

            connection.end();

        });

    });

});

Ez egy, az eddigieknél összetettebb kód. Mivel csak azzal az e-mail címmel lehet örökbefogadni, amelyikkel regisztrált a felhasználó, ezért először egy erre vonatkozó ellenőrzés van a kódban. Ha a felhasználó jó e-mail címmel regisztrált, akkor lekéri az e-mail címhez tartozó azonosítót, és beilleszt minden adatot a gazdik táblába. Mikor ez is rendben lezajlott, lekéri az új gazdi azonosítóját és beilleszti az állatok táblába, ahhoz az állathoz, amelyiket örökbefogadta a felhasználó. Ezt a szűrést név alapján végeztük. Egy nagyon fontos lépés, hogy ezzel együtt az állat „orokbefogadott” mezőjének true értéket ad.

A legtöbb script kódra igaz az, hogy miután végig ért a folyamat, eredménytől függően újratölti a weboldalt vagy átirányít egy másikra.

### segitenek.html felépítése

A HTML oldal szerkezete egy ugyanolyan űrlapból áll, mint ami az orokbefogadas.html-en található. Ehhez az oldalhoz a „segitenek.js” JavaScript file tartalmazza a működést biztosító kódot.

Itt is a gombra való kattintás hívja meg a „jelentkezes” függvényt.

Ez a függvény az előző oldalakhoz tartozó script file-okhoz hasonlóan tartalmaz egy ellenőrzést arra, hogy üres értékeket, illetve érvénytelen e-mail címet ne lehessen beküldeni.

Ezen kívül, hasonló szerver oldali metódussal küldi be az adatbázisba a felhasználótól kapott adatokat:

app.post('/jelentkezes', bodyParser.json(), (req, res) => {

    const { onkentesnev, onkentesemail, telszam, kozepiskolas } = req.body;

    const connection = kapcsolat();

    connection.connect();

    let kozepiskolasBool = false;

    if (kozepiskolas === '1') {

        kozepiskolasBool = true;

    }

    const checkUserQuery = `SELECT FelhasznaloID FROM Felhasznalok WHERE Email = "${onkentesemail}"`;

    connection.query(checkUserQuery, (error, results) => {

        if (error) {

            console.error('Hiba az ellenőrzés során:', error);

            res.status(500).json({ error: 'Hiba az ellenőrzés során' });

            connection.end();

            return;

        }

        if (results.length === 0) {

            res.status(404).json({ error: 'Nincs felhasználó ezzel az e-mail címmel' });

            connection.end();

            return;

        }

        const felhasznaloId = results[0].FelhasznaloID;

        const insertQuery = `INSERT INTO Onkentesek (onkentesnev, onkentesemail, telszam, kozepiskolas, felhasznalo\_id) VALUES ("${onkentesnev}", "${onkentesemail}", "${telszam}", ${kozepiskolas}, ${felhasznaloId})`;

        connection.query(insertQuery, (error, result) => {

            if (error) {

                console.error('Hiba a jelentkezés mentése során:', error);

                res.status(500).json({ error: 'Hiba a jelentkezés mentése során' });

            } else {

                console.log('Jelentkezés sikeresen mentve');

                res.status(200).json({ message: 'Jelentkezés sikeresen mentve' });

            }

            connection.end();

        });

    });

});

Itt is a fentebb is látható módon kéri le a regisztrált felhasználó ID-ját, majd ezeket az adatokat beilleszti az „Onkentesek” táblába

### jelszomodositas.html felépítése

Ennek az oldala szintén egy, a fentiekhez hasonló űrlapból és a felhasználót tájékoztató szövegből áll. A „jelszo.js” tartalmazza a kliens oldali kódot, ami nagyon hasonló a bejelentkezéshez, egy plusz szűrést tartalmaz, arról, hogy a régi és új jelszó nem lehet ugyanaz.

A szerver oldali kód is ugyanazzal a metódussal működik, mint a regisztráció, a különbség, hogy itt UPDATE lekérdezést használtunk:

`UPDATE Felhasznalok SET Jelszo = "${ujjelszo}" WHERE Email = "${email}" AND Jelszo="${regijelszo}"`;

## Az admin felület fejlesztése

Bejelentkezés után, ha van a felhasználónak adminjogosultsága, a galeriaszerkesztes.html oldalra irányítja át a weboldal.

### galeriaszerkesztes.html felépítése

Ezen az alábbi képfeltöltő felület található, amely a galériában jelenít meg képeket.

<section class="wrapper" >

    <div class="admin-form">

        <h2>Galéria szerkesztése</h2>

        <form id="imageUploadForm" action="/upload" method="POST" enctype="multipart/form-data">

            <input type="file"  id="fileInput" name="image" accept="image/\*">

            <button type="submit" class="btn" onclick='upload(event)' >Feltöltés</button>

        </form>

    </div>

</section>

Az oldal mögötti működési mechanizmus a „galeriaszerkesztes-script.js” file tartalmazza.

async function upload(event) {

    event.preventDefault();

    const fileInput = document.getElementById('fileInput');

    const file = fileInput.files[0];

    if (file) {

        try {

            await uploadFile(file);

        } catch (error) {

            console.error('Hiba történt a fájl feltöltése során:', error);

        }

    } else {

        console.error('Nincs fájl kiválasztva.');

    }

}

async function uploadFile(file) {

    const formData = new FormData();

    formData.append('image', file);

    try {

        const response = await fetch('http://127.0.0.1:8080/upload', {

            method: 'POST',

            body: formData

        });

        if (!response.ok) {

            throw new Error('Hiba történt a fájl feltöltése során.');

        }

        const data = await response.json();

        alert('Fájl sikeresen feltöltve', data);

        window.location.reload();

    } catch (error) {

        alert('Nem sikerült a feltöltés', data);

        window.location.reload();

    }

}

Ahogy az látható, két függvény segíti a file feltöltést, async és await használatával oldottuk meg a gördülékeny működést.

const storage = multer.diskStorage({

    destination: (req, file, cb) => {

        cb(null, 'public/images');

    },

    filename: (req, file, cb) => {

        const ext = path.extname(file.originalname);

        const now = new Date();

        const formattedDate = `${now.getFullYear()}${now.getMonth() + 1}${now.getDate()}${now.getHours()}${now.getMinutes()}${now.getSeconds()}${now.getMilliseconds()}`;

        cb(null, `${formattedDate}${ext}`);

    }

});

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

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

app.post('/upload', upload.single('image'), (req, res) => {

    const { filename, path: filePath } = req.file;

    const connection = kapcsolat();

    connection.connect();

    const insertQuery = `INSERT INTO kepek (filename, filepath) VALUES (?, ?)`;

    connection.query(insertQuery, [filename, filePath], (error, results, fields) => {

        if (error) {

            console.error('Hiba az adatbázisba történő mentés során:', error);

            res.status(500).json({ error: 'Hiba az adatbázisba történő mentés során' });

            return;

        }

        res.status(200).json({ message: 'Fájl sikeresen feltöltve és mentve az adatbázisba' });

        connection.end();

    });

});

A képek tárolásához a „multer” nevű modult alkalmaztuk. A képeket a feltöltési dátum szerint tároljuk az adatbázisban. A feltöltött képek a ’public/images’ mappába kerülnek

### További admin oldalak

További admin felületek:

* allatokszerkesztese.html: Itt az ’allatok’ táblához lehet új állatokat hozzáadni. Miután minden adatot sikeresen megadtunk, átdob a kepfeltoltes.html oldalra, ahol hozzá lehet rendelni az állathoz egy képet, és ez a kép a galériába is feltöltődik.
* elozoev.html: Ezen az oldalon az ’elozoev’ táblához lehet hozzáadni, amiből a diagram is működik. Ha olyan hónapot ad hozzá az admin, ami már szerepel az adatbázisban, akkor csak a számadatokat cseréli ki a program.
* admin.html: Itt lehet választani a regisztrált e-mail címek közül, és a kiválasztott felhasználó adminjogosultságát módosítani.

Ezeknek a szerver és kliens oldali kódja az eddig bemutatott oldalak kódjainak szerkezetével azonos, csak a nevek és azonosítók változtak. A kinézetük és HTML kód szerkezetük is az eddigi űrlapokkal azonosak.

# C# asztali alkalmazás

## Bevezetés

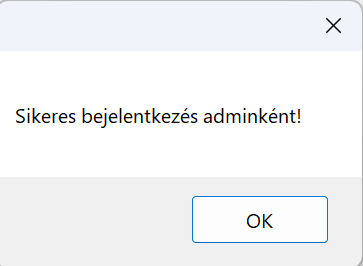
A Mancs Állatmenhely alkalmazás egy olyan szoftver, amely segíti az állatmenhely adminisztrátorait az állatok adatainak kezelésében és nyomon követésében. Az alkalmazás célja a menhelyen tartott állatok részletes adatainak nyilvántartása, ideértve az állatok nevét, fajtáját, korát, stb. Az alkalmazás lehetővé teszi az adminisztrátorok számára, hogy könnyen hozzáférjenek ezekhez az adatokhoz, és módosítsák vagy frissítsék azokat szükség esetén.

## Alapok

Az alkalmazás egy Windows Forms alkalmazás, amelyet a Microsoft .NET Framework keretrendszer segítségével írtunk C# programozási nyelven. Az adatok tárolására és kezelésére MySQL adatbázist használ. Az alkalmazás az adatbázis-kezeléshez a MySQL Connector/.NET nevű .NET adatbázis-illesztőprogramot használja.

## Funkciók

### Bejelentkezés

Az alkalmazás lehetővé teszi az adminisztrátorok számára a bejelentkezést felhasználónév és jelszó megadásával. Az adminisztrátorok csak megfelelő jogosultságokkal rendelkező felhasználók lehetnek, akik a bejelentkezés után különleges műveleteket végezhetnek az adatokkal.

35. ábra C# bejelentkezési felület

36. ábra sikeres bejelentkezés

string username = textBoxUsername.Text;

string password = textBoxPassword.Text;

try

{

connection.Open();

string query = $"SELECT \* FROM Felhasznalok WHERE FelhasznaloNev = '{username}' AND Jelszo = '{password}'";

MySqlCommand cmd = new MySqlCommand(query, connection);

MySqlDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

bool isAdmin = reader.GetBoolean("adminjogosultsag");

if (isAdmin)

{

MessageBox.Show("Sikeres bejelentkezés adminként!");

panel2.Visible = true;

comboBox1.SelectedIndexChanged += comboBox1\_SelectedIndexChanged;

}

else

{

MessageBox.Show("Sajnos nem rendelkezel admin jogosultsággal!");

}

}

else

{

MessageBox.Show("Hibás felhasználónév vagy jelszó!");

}

}

catch (Exception ex)

{

MessageBox.Show("Hiba történt a bejelentkezés során: " + ex.Message);

}

finally

{

connection.Close();

}

### Adatok megjelenítése

37. ábra adatok megjelenítése

Az alkalmazás képes megjeleníteni az adatbázisban tárolt adatokat egy DataGridView komponensen keresztül. A felhasználó választhat a rendelkezésre álló táblák közül, és azok tartalmát megjeleníteni a grafikus felhasználói felületen.

### Adatok módosítása és törlése

Az alkalmazás lehetővé teszi az adminisztrátorok számára az adatok módosítását és törlését közvetlenül a DataGridView segítségével. A felhasználó képes módosítani a táblákban található rekordokat, és azokat azonnal menteni az adatbázisba. Ilyenkor beírjuk az adott szöveget, avagy számot a megfelelő helyre és kijelöljük az egész sort a legszélső üres oszlop segítségével, mivel csak így fog módosulni vagy törlődni az egész sor. Emellett lehetőség van rekordok törlésére is az alkalmazás segítségével.

39. ábra rekord beszúrva

38. ábra rekord törölve

## Az alkalmazás felépítése

Az alkalmazás két fő részből áll: a felhasználói felületből (GUI) és az adatbázis-kezelő részből. A felhasználói felületen a felhasználók megjeleníthetik az adatokat, módosíthatják azokat és hozzáadhatnak új adatokat. Az adatbázis-kezelő rész felelős az adatbázishoz való kapcsolódásért, az adatok lekérdezéséért és azok kezeléséért.

## Az alkalmazás kódja

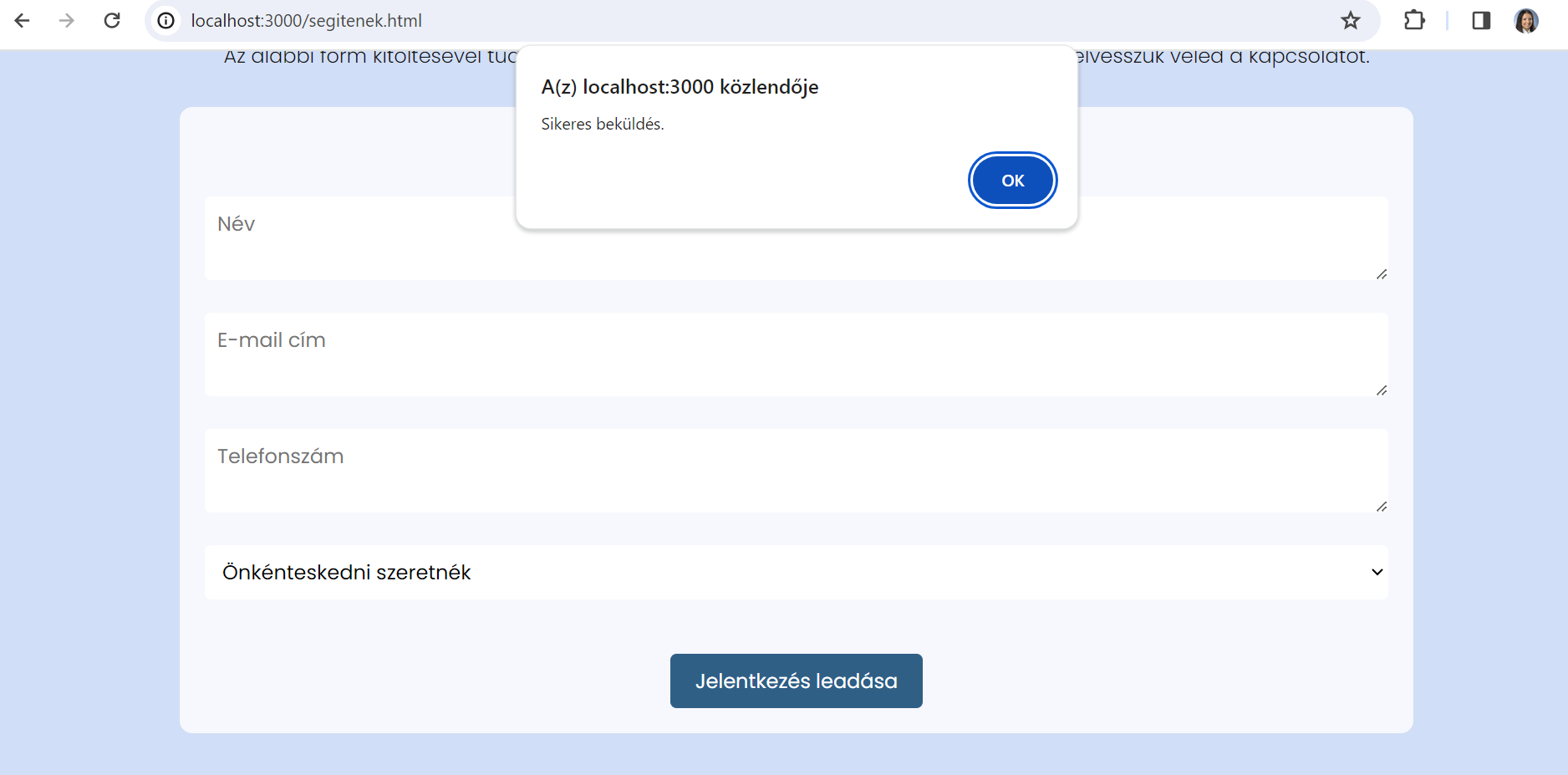
Az alkalmazás kódja jól strukturált és moduláris. A kód különböző osztályokra és függvényekre van felosztva, amelyeket könnyű megérteni és karbantartani. A kód tiszteletben tartja az objektumorientált programozás elveit, és lehetővé teszi az új funkciók hozzáadását vagy a meglévő funkciók módosítását.

## Összegzés

Az alkalmazás egy hatékony eszköz az állatmenhely adminisztrátorai számára az állatokkal kapcsolatos adatok kezelésében és nyomon követésében. A felhasználóbarát felhasználói felület és a könnyen kezelhető funkciók lehetővé teszik az adminisztrátorok számára, hogy hatékonyan dolgozzanak az állatokkal kapcsolatos információkkal és tevékenységekkel.

# Tesztelési hibák

## Üresen is beküldhető az űrlap



40. ábra első hiba

Ezt próbáltuk az összes oldalnál kezelni, JavaScript kódban.

Például a regisztrációnál:

if (FelhasznaloNev === '' || Email === '' || Jelszo === '') {

        alert("Minden mező kitöltése kötelező.");

        return;

    }

## A diagram mérete

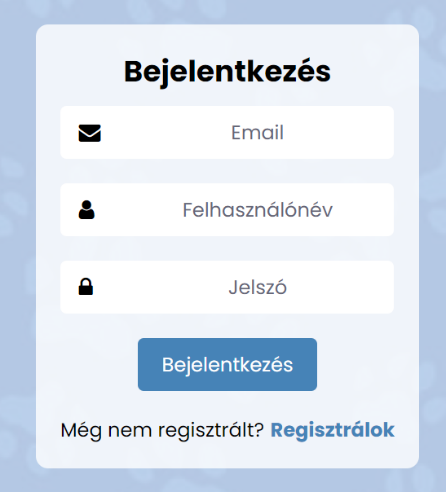
41. ábra második hiba

A diagram eleinte csak hat hónapot tudott megjeleníteni úgy, hogy minden látszódjon.

Javításként megnöveltük a szélességet úgy, hogy tizenkét hónap kiférjen:

<BarChart width=**{1200}** height={300} data={data} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

## Regisztráció, bejelentkezés

A regisztrációnál kértünk felhasználónevet, a bejelentkezésnél viszont csak e-mail címet, ezt is kijavítottuk.

42. ábra harmadik hiba

# Fejlesztési lehetőségek

## Fejlesztendő funkciók

### Adományozás funkció

### Jelszó titkosítás

### Bejelentkezési adatok tárolása, munkamenetek létrehozása.

# Felhasznált irodalom

## Irodalomjegyzék

* <https://en.wikipedia.org/wiki/MySQL>
* <https://en.wikipedia.org/wiki/.NET>
* <https://www.w3schools.com/mysql/default.asp>
* <https://hu.wikipedia.org/wiki/HTML>
* <https://www.w3.org/Style/CSS/Overview.en.html>
* <https://hu.wikipedia.org/wiki/JavaScript>
* <https://webiskola.hu/javascript-ismeretek/mi-az-a-javascript-a-js-bemutatasa/>
* <https://nodejs.org/en>
* <https://hu.legacy.reactjs.org/docs/getting-started.html>
* <https://hu.wikipedia.org/wiki/Cascading_Style_Sheets>
* <https://en.wikipedia.org/wiki/XAMPP>

## **Köszönetnyilvánítás**

Ezúton szeretnénk köszönetet mondani témavezetőnknek, Szabó Dániel tanár úrnak a sok

segítségért és tanácsért, amelyek hozzájárultak a záródolgozatunk létrejöttéhez.