**Ceglédi SZC Közgazdasági és Informatikai**

**Technikum**

Szakképzés megnevezése: Szoftverfejlesztő és -tesztelő

Szakma azonosító szám: 5 0613 12 03

**ZÁRÓDOLGOZAT**

**Mancs Állatmenhely**

Témavezető: Készítette:

Szabó Dániel Dúzs Maja Flóra

Kiss Ádám

Cegléd, 2024

Tartalomjegyzék

[1. Bevezetés 4](#_Toc164079928)

[1. A záródolgozat témája 4](#_Toc164079929)

[2. Dokumentáció 4](#_Toc164079930)

[1. Az elkészítéshez felhasznált szoftverek 4](#_Toc164079931)

[2. A használt programozási nyelvek 4](#_Toc164079932)

[3. Az adatbázis 5](#_Toc164079933)

[3.1 A táblák szerkezete 6](#_Toc164079934)

[3.2 Megkötések a táblák között 8](#_Toc164079935)

[4. A weboldal felhasználói része 9](#_Toc164079936)

[4.1 Főoldal 10](#_Toc164079937)

[4.2 Örökbefogadható állataink 12](#_Toc164079938)

[4.3 Információk 12](#_Toc164079939)

[4.4 Galéria 13](#_Toc164079940)

[4.5 Kapcsolat és vélemények 14](#_Toc164079941)

[4.6 Belépés és regisztráció 15](#_Toc164079942)

[4.7 Örökbefogadás 16](#_Toc164079943)

[4.8 Segítenék 17](#_Toc164079944)

[4.9 Profil 18](#_Toc164079945)

[5. A weboldal admin része 19](#_Toc164079946)

[5.1 Galéria szerkesztés 19](#_Toc164079947)

[5.2 Örökbefogadható állataink szerkesztése 19](#_Toc164079948)

[5.3 Éves adatok szerkesztése 19](#_Toc164079949)

[5.4 Adminjogosultság kapcsoló 20](#_Toc164079950)

[5.5 Kijelentkezés 20](#_Toc164079951)

[6. Az oldal fejlesztése 21](#_Toc164079952)

[6.1 Közös részek 22](#_Toc164079953)

[6.2 index.html felépítése 25](#_Toc164079954)

[6.3 allataink.html felépítése 28](#_Toc164079955)

[6.4 informaciok.html felépítése 28](#_Toc164079956)

[6.5 galeria.html felépítése 28](#_Toc164079957)

[6.6 kapcsolat.html felépítése 30](#_Toc164079958)

[6.7 login.html és regisztracio.html felépítése 34](#_Toc164079959)

# 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.

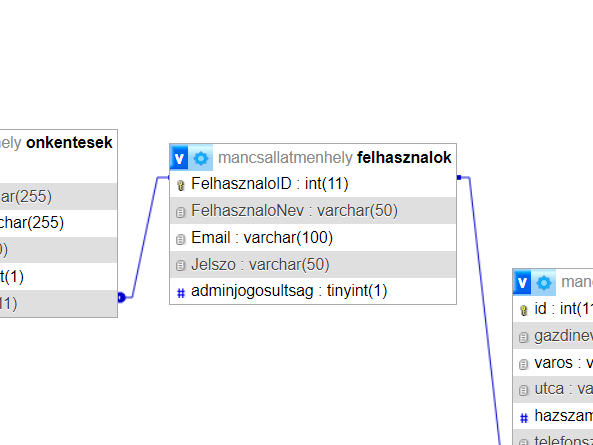
### 3.1 A táblák szerkezete

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)

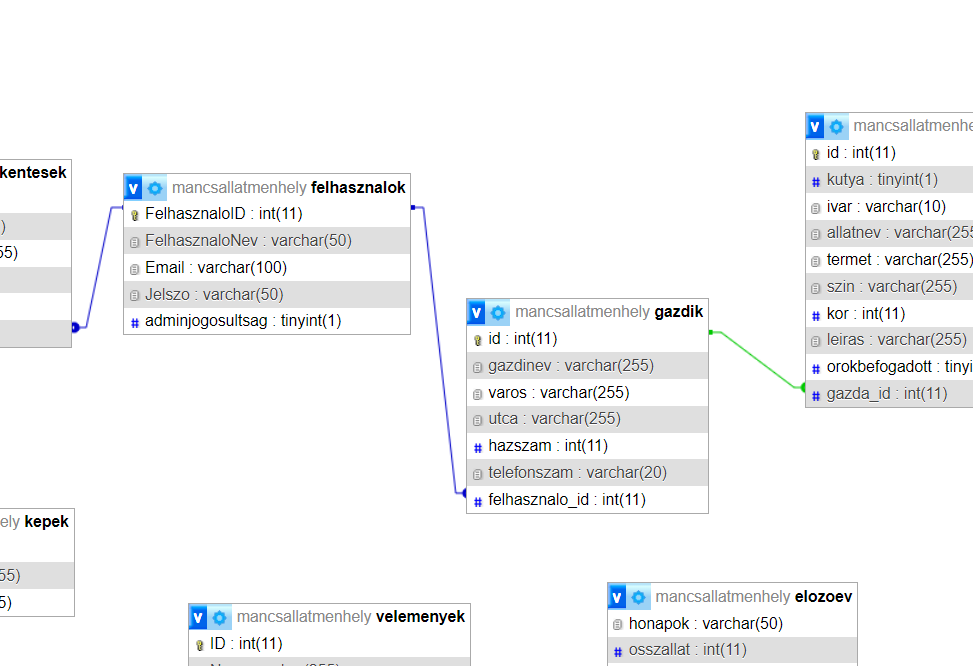
„onkentesnev”: az önkéntes neve (karakterek, a felhasználó adja meg)

„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)

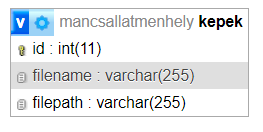
„kor”: az állat kora (szám, az admin adja meg)

„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

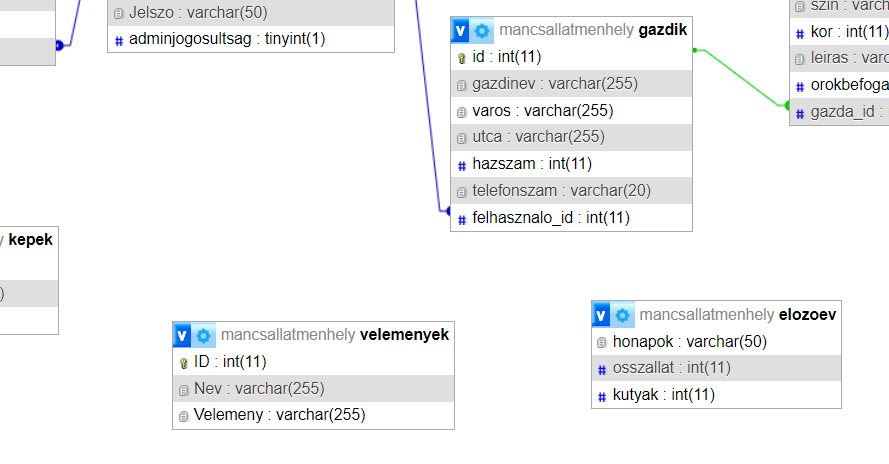
6. ábra "kepek" tábla

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

„filename”: a kép file neve (karakterek)

„filepath”: a kép file útja (karakterek)

„velemenyek” tábla

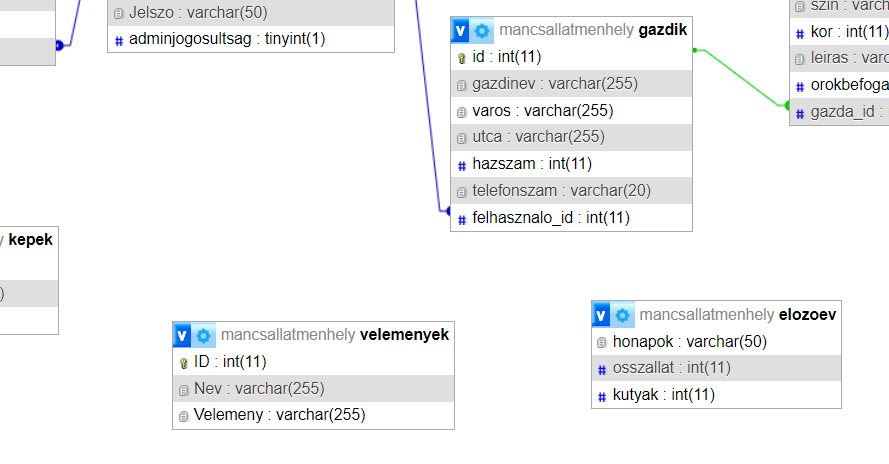
„id”: a vélemény azonosítója (szám, automatikusan generálódik, PK)

7. ábra "velemenyek" tábla

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

„Velemeny”: a beküldött vélemény (karakterek, 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"

    })

}

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

    const connection = kapcsolat();

    connection.connect()

    connection.query("show tables",(error,results,fields)=>{

        if(error)

            res.send({"error": "Hiba a lekérés során"})

        else

            res.send(results);

    });

    connection.end();

});

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

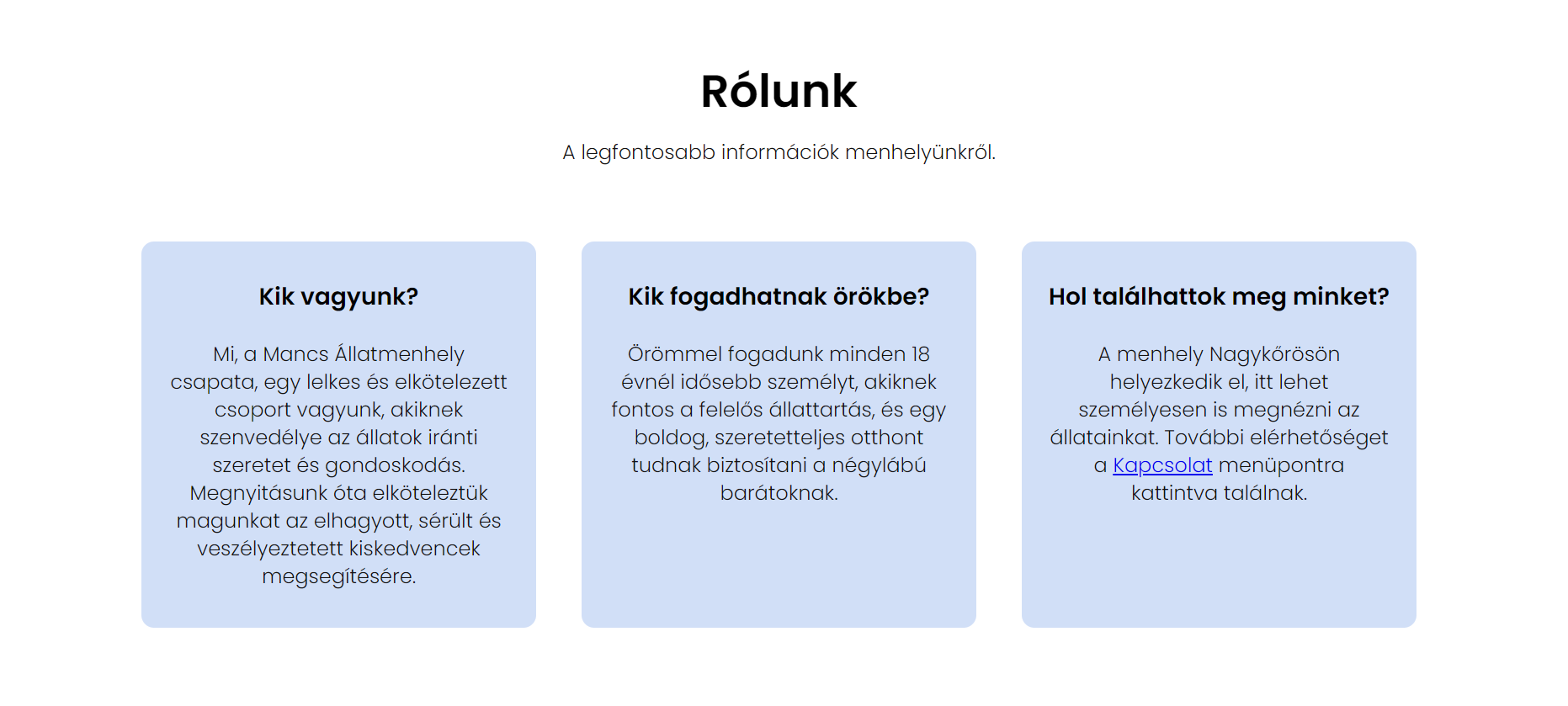
Weblapunk a Mancs Állatmenhelyet és annak működését, állatait, tevékenységeit mutatja be. A készítés során törekedtünk arra, hogy mindenki számára áttekinthető, felhasználóbarát felületet készítsünk.

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.

### 4.1 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

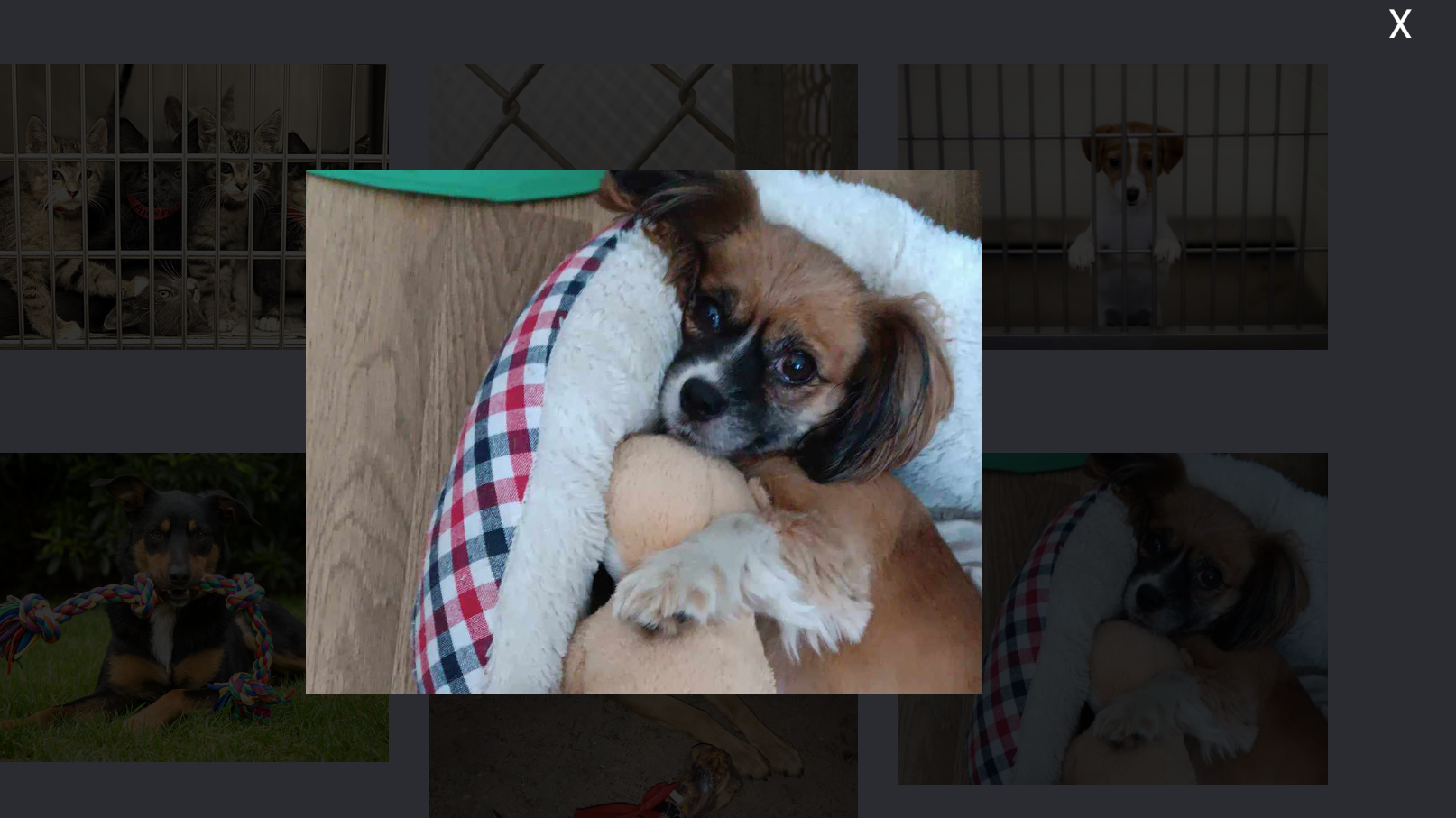
### Információk

Ezen az oldalon található minden, az örökbefogadáshoz szükséges információ.

### Galéria

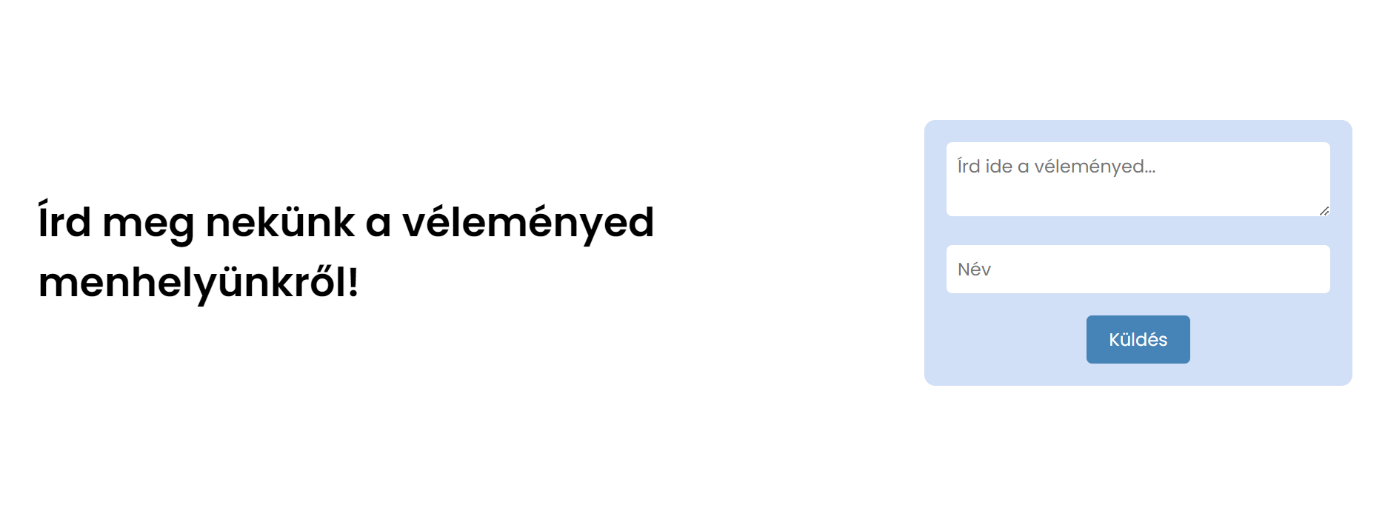
A galéria oldalon az állatokról találhatók képek. A képekre kattintva nagyobb méretben is meg lehet tekinteni őket.





### Kapcsolat és vélemények

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

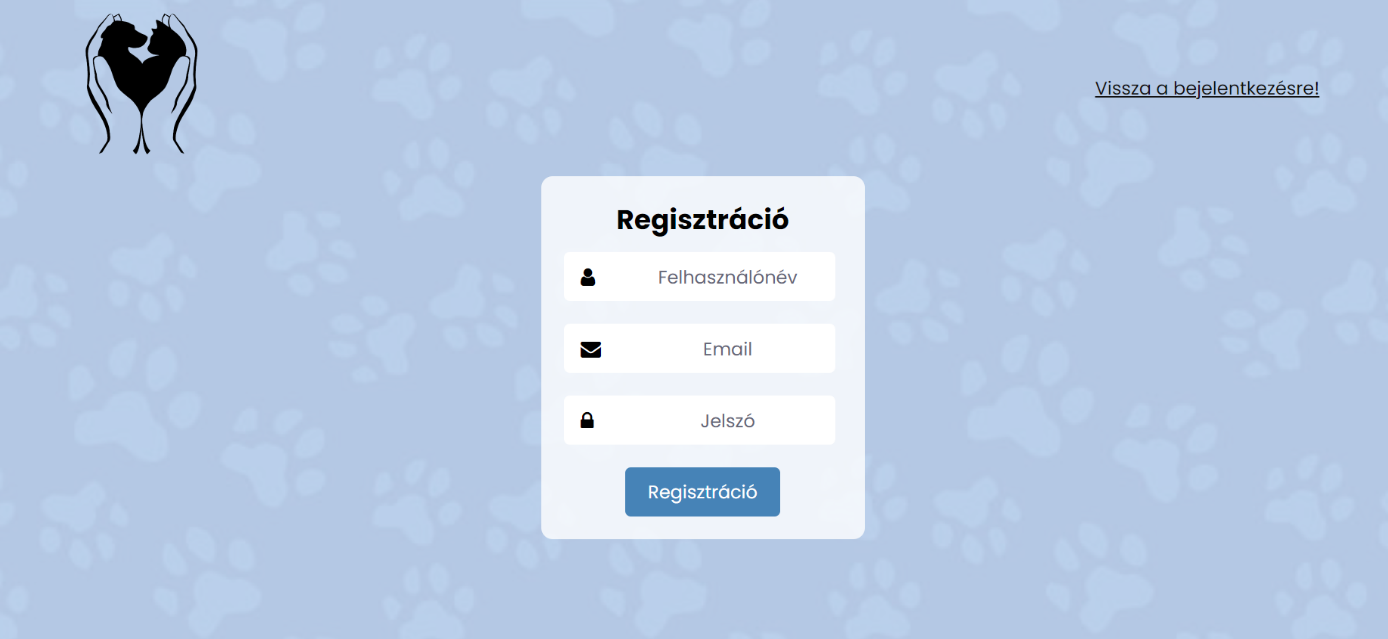
A másik részén lehetőség van véleményt írni, és a három legfrissebb bejegyzés meg is található az oldal alján.

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.

### 4.6 Belépés és regisztráció

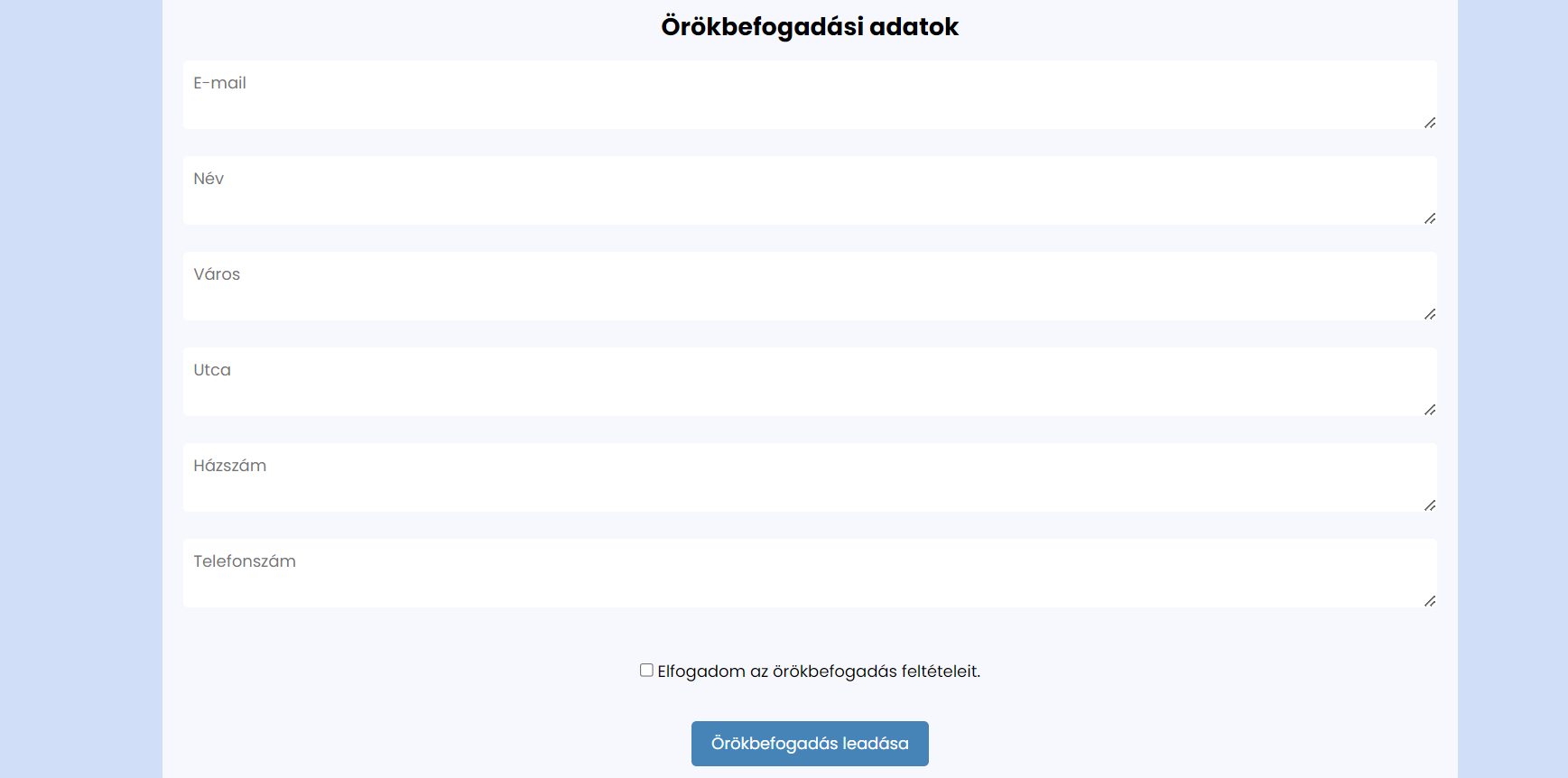
A navigációs menü „Bejelentkezés” gombjára kattintva nyílik meg a belépési felület.

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

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.



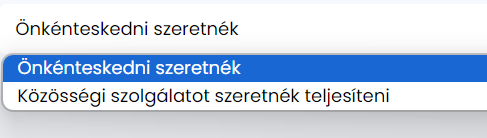
Sikeres örökbefogadás esetén a következő üzenet jelenik meg a felhasználónak:

### 4.8 Segítenék

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.



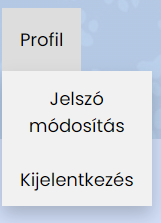
Az utolsó mezőben lehet kiválasztani, hogy ki az, aki önkéntes munkát és ki az, aki közösségi szolgálatot szeretne teljesíteni:



Sikeres jelentkezés után, itt is kap üzenetet a felhasználó:

### 4.9 Profil

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



A jelszómódosításra kattintva ez az űrlap jelenik meg:



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.

### 5.1 Galéria szerkesztés

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

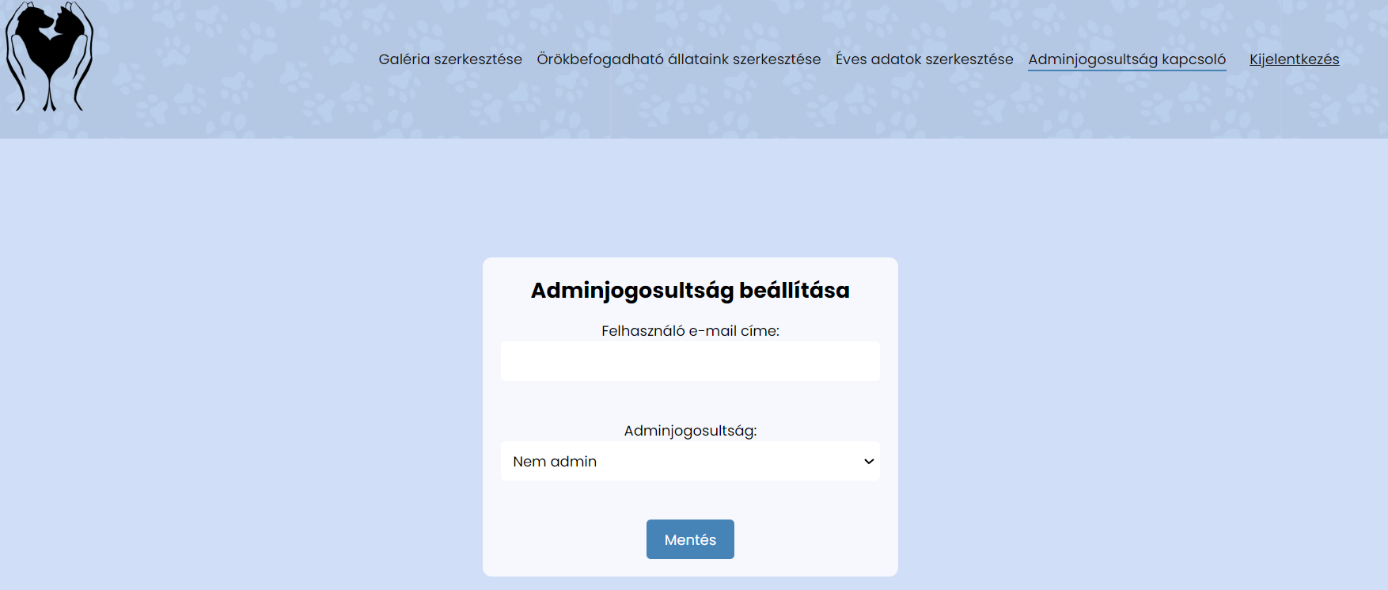
### Éves adatok szerkesztése

Ezen az űrlapon lehet a diagram adatait szerkeszteni.



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.

### 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.

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.

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

### 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>

            <p> Mi, a Mancs Állatmenhely csapata, egy lelkes és elkötelezett csoport vagyunk, akiknek szenvedélye az állatok iránti szeretet és gondoskodás. Megnyitásunk óta elköteleztük magunkat az elhagyott, sérült és veszélyeztetett kiskedvencek megsegítésére.</p>

        </div>

<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 a useEffect hook segítségével 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.

Példák:

* az „about” class beállításai

.about{

    width: 80%;

    margin:auto;

    text-align: center;

    padding-top: 55px;

    padding-bottom: 60px;

}

* „about-col” class beállításai

.about-col{

    flex-basis: 31%;

    background: rgba(141, 174, 235, 0.4);

    border-radius: 10px;

    margin-bottom: 5%;

    padding: 20px 12px;

    box-sizing: border-box;

    transition: 0.5s;

}

* 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

### 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 megjelenítésének egy része 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 azt teszi lehetővé CSS beállításokkal, hogy a képekre kattintva nagyban is megnézhetik a felhasználók az állatokat, az X-re kattintva pedig „be tudják zárni” őket, azaz kikapcsolni a megjelenítést.

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

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

    const connection = kapcsolat();

    connection.connect();

    const query = `SELECT \* FROM kepek`;

    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 images = results.map(image => {

                return {

                    filename: image.filename,

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

                };

        });

            console.log('Képek sikeresen lekérdezve');

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

        }

        connection.end();

    });

});

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

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

Ez azt eredményezte, hogy a ’public/images’ könyvtárban található összes fájl elérhető lett az alkalmazásból.

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. A függvény mindig lefut, mikor az oldal újratöltődik.

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);

        });

    } catch (error) {

        console.error('Hiba történt a képek lekérése során:', error);

    }

}

window.onload = fetchImages;

### kapcsolat.html felépítése

A közös elemeken kívül tartalmaz:

<div class="container">

    <h1>Kapcsolat</h1>

    <h2>Telefonszám: +36 31 415 9265</h2>

    <h2>E-mail cím: mancsallatmenhely@gmail.com</h2>

    <h2>Nyitvatartás: Hétfőtől szombatig 10 és 17 óra között.

    <h2>Cím: 2750 Nagykőrös</h2>

    <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d187821.7789590195!2d19.609593775111215!3d47.01807578143839!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x474162db142a76e9%3A0x400c4290c1e13a0!2zTmFneWvFkXLDtnMsIDI3NTA!5e0!3m2!1shu!2shu!4v1712520961020!5m2!1shu!2shu" width="600" height="450" style="border:0;" allowfullscreen="" loading="lazy" referrerpolicy="no-referrer-when-downgrade"></iframe>

</div>

<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>

Egy containert, amiben szövegek találhatók, illetve egy linket a Google Mapsről, egy fiktív helyszínről.

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();

}

async function LegfrissebbVelemenyek() {

    try {

        const response = await fetch('http://localhost:8080/legfrissebb\_velemenyek');

        const velemenyek = await response.json();

        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);

        });

    } catch (error) {

        console.error('Hiba történt a legfrissebb vélemények betöltésekor:', error);

    }

}

A „/legfrissebb\_velemenyek” végpontú szerver oldali kód a következő:

app.get('/legfrissebb\_velemenyek', (req, res) => {

    const connection = kapcsolat();

    connection.connect();

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

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

      if (error) {

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

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

      } else {

        console.log('Legfrissebb vélemények sikeresen lekérdezve');

        res.status(200).json(result);

      }

      connection.end();

    });

});

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. Ez a függvény aszinkron módon küld egy POST kérést a szerver felé a /velemenyek végponton keresztül, JSON formátumban elküldve a felhasználó nevét és véleményét.

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 })

        });

        if (response.ok) {

            alert('Vélemény sikeresen elküldve.');

            LegfrissebbVelemenyek();

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

        } else {

            const data = await response.json();

            alert(data.error || 'Hiba történt a küldés során.');

        }

    } catch (error) {

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

        alert('Hiba történt a kérés során.');

    }

}

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

    const { nev, velemeny } = req.body;

    const connection = kapcsolat();

    connection.connect();

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

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

      if (error) {

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

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

      } else {

        console.log('Vélemény sikeresen mentve');

        res.status(200).json({result});

      }

      connection.end();

    });

});

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:

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

    const { email, felhasznalonev, password } = req.body;

    const connection = kapcsolat();

    connection.connect();

    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" });

            }

        }

        connection.end();

    });

});

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

    const { FelhasznaloNev, Email, Jelszo } = req.body;

    const connection = kapcsolat();

    connection.connect();

    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 ezek a HTML elemek találhatók meg a kódban:

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

        <h1>Örökbefogadás</h1>

        <p>Ezen az oldalon lehetőséged van örökbefogadási igényt benyújtani. Az állatokról az <a href="allataink.html">Örökbefogadható állataink</a> menüpont alatt találsz leírást.</p>

        <p>Kérlek, mielőtt kitöltenéd ezt az űrlapot, olvasd át alaposan a tudnivalókat az <a href="informaciok.html">Információk</a> oldalon, és bizonyosodj meg róla, hogy teljesítetted a feltételeket.</p>

        <section class="felhasznalok-wrapper">

            <div class="admin-form">

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

                <select id="allatok">

                </select>

            </div>

        </section>

        <section class="allatokwrapper">

            <div class="admin-form">

            <h2>Örökbefogadási adatok</h2>

                <form id="orokbefogadasForm">

                    <textarea id="gazdiemail" placeholder="E-mail" required></textarea>

                    <textarea id="gazdinev" placeholder="Név" required></textarea>

                    <textarea id="varos" placeholder="Város" required></textarea>

                    <textarea id="utca" placeholder="Utca" required></textarea>

                    <textarea id="hazszam" placeholder="Házszám" required></textarea>

                    <textarea id="telefonszam" placeholder="Telefonszám" required></textarea>

                    <input type="checkbox" id="feltetelek" >

                    <label for="feltetelek">Elfogadom az örökbefogadás feltételeit.</label>

                    <button type="submit" class="btn" onclick='orokbefogadas(event)' >Örökbefogadás leadása</button>

                </form>

            </div>

        </section>

</section>

Az oldal első részében lehet állatot választani, a második részében pedig egy űrlap van, ennek kitöltésével lehet örökbefogadni. A gombra kattintva aktiválódik az „orokbefogadas” függvény, aminek a kódja az „orokbefogadas-script.js”-ben található.

Két részből áll a kód:

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 })

        });

        const data = await response.json();

        console.log(data);

        if (response.ok) {

            console.log('Sikeres beküldés');

            alert("Sikeres örökbefogadás.");

            window.location.reload();

        } else {

            alert("A beküldés sikertelen.");

            window.location.reload();

        }

}

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 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.

async function jelentkezes(event) {

    event.preventDefault();

    const onkentesNev = document.getElementById('onkentesnev').value;

    const onkentesEmail = document.getElementById('onkentesemail').value;

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

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

    if (onkentesNev === '' || onkentesEmail === '' || telefonszam === '' || kozepiskolas === '') {

        alert("Minden mező kitöltése kötelező.");

        return;

    }

    if (!onkentesEmail.includes('@')) {

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

        return;

    }

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

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify({ onkentesnev:onkentesNev, onkentesemail:onkentesEmail, telszam:telefonszam, kozepiskolas:kozepiskolas })

    });

    const data = await response.json();

    console.log(data);

    if (response.ok) {

        console.log('Sikeres beküldés');

        alert("Sikeres beküldés.");

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

    } else {

        alert("A beküldés sikertelen.");

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

    }

}

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:

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