

ELTŰNTÁLLAT-KERESŐ

Cserhádi Máté, Kosik Alex

Nógrád Vármegyei Szakképzési Centrum Szent-Györgyi Albert Technikum

Szoftverfejlesztő és -tesztelő technikus (szakma azonosító: 5-0613-12-03)

2024. április .17.

Tartalom

Bevezetés.....	1
1. Felhasznált technológiák	2
1.1. PHP	2
1.2. HTML	2
1.3. CSS	2
1.4. JavaScript.....	2
1.5. Node.js	3
1.6. Composer.....	3
1.7. Tailwind CSS.....	3
1.8. Vite	4
1.9. Visual Studio Code	4
1.10. XAMPP.....	4
1.11. MySQL	5
1.12. GitHub	5
1.13. WinSCP	5
1.14. Dbdiagram	5
1.15. Figma	6
1.16. .NET Keretrendszer	6
1.17. .NET C#.....	6
1.18. Microsoft Word	7
1.19. Teams.....	7
2. Felhasználói dokumentáció	8
2.1. Kezdőoldal.....	8
2.2. Publikus posztok.....	9
2.2.1. Lapozó.....	11
2.2.2. Publikus kártyák.....	11
2.2.3. Állat adatlapja	12
2.2.4. Szűrés	12

2.3. Regisztráció	13
2.4. Bejelentkezés	15
2.5. Menü	16
2.5.1. Új poszt	16
2.5.2. Saját posztjaim	17
2.5.3. Szerkesztés	18
2.5.4. Állat törlése	19
2.6. Chat	20
2.6.1. Chat beállítások	21
2.7. Saját profil módosítása	22
2.8. Kijelentkezés	23
3. Fejlesztői dokumentáció	24
3.1. Laravel	24
3.1.1. Főbb jellemzők	25
3.1.2. Előnyök a használatában	25
3.2. Weboldal elérése	26
3.3. Projekt telepítése és a webalkalmazás elindítása	26
3.4. Környezeti változók	27
3.5. Útvonalak (Routing)	29
3.5.1. Kontrollerek	29
3.5.2. Middleware	30
3.6. Middleware	31
3.7. Modellek (Model)	31
3.8. CSRF Token	32
3.9. Kontrollerek (Controller)	33
3.9.1. Felhasználóhoz tartozó controller	33
3.9.2. Controller létrehozása	34
3.9.3. Állathoz tartozó controller	35
3.9.4. Admin controller	39

3.9.5. Városok kontroller	40
3.10. Nézetek (View).....	40
3.10.1. Publikus posztok	43
3.10.2. Poszt feltöltése	43
3.11. API.....	44
3.12. Városok és megyék forrása.....	45
3.13. Tesztelés	45
3.13.1. Frontend	45
3.13.2. Backend.....	46
4. Az adatbázis célja.....	48
4.1. Tervezés megkezdése	48
4.2. Tervezési lépések.....	48
4.3. Egyed típusok/egyedek meghatározása	48
4.4. Kapcsolatok meghatározása.....	49
4.5. Saját táblák.....	49
4.5.1. Users.....	49
4.5.2. Cities	50
4.5.3. Counties	50
4.5.4. Animals	50
4.6. Keretrendszer táblái	51
4.6.1. Ch_favorites (chat kedvencek).....	51
4.6.2. Ch_messages (üzenetek)	51
4.6.3. Password_reset_tokens	51
4.7. Adatbázis tervezése a dbdiagram felületen.....	52
4.8. Adatbázis megvalósítása a phpmyadmin felületen.....	52
4.9. Adatbázis-továbbfejlesztési lehetőségek	53
5. Összefoglalás.....	54
Források.....	55
Ábrajegyzék.....	57

Bevezetés

Az Eltűntállat-kereső webalkalmazás azt a célt kívánja szolgálni, hogy akinek eltűnik a kisállata, netán szilveszterkor elszalad a tűzijáték hallatán, vagy nyitva marad a kapu a kertben és kiszalad rajta, akkor könnyen meg tudjuk találni. Az emberek nagy része valamilyen közösségi média felületén keresgéli az állatait, viszont ez nehezzé válik, hisz csoportról csoportra kell ugrálni. A mi webalkalmazásunk ezt leegyszerűsíti, mindenki könnyedén fel tudja tölteni az állatát, és ha az emberek megtalálják, akkor a beépített üzenetküldő rendszernek köszönhetően egyszerűen fel tudjuk venni az illetővel a kapcsolatot.

A csapatunk 2 főből áll, sok tudást szereztünk a munkánk során. Jobban megismertük a technológiákat, amiket használtunk. Nagyobb belátást nyertünk abba, hogy a nagyobb weboldalak szerkezete, működése hogyan is épül fel. Cserhádi Máté a háttérrendszer (backend) fejlesztésével foglalkozott, az adatbázis megvalósítását ketten végeztük, Kosik Alex a megjelenésért (frontend) volt felelős, illetve a megjelenés megtervezéséért. Az ötletelést december környékén kezdtük el. Először egy másik ötlettel indultunk, eltűnt személyeket kereső webalkalmazást szeretnénk volna készíteni, majd szilveszter után jött az ötlet az eltűnt állatokról. A mentorainktól kapott motiválás hatalmas segítséget nyújtott számunkra a nehezebb időszakokban. Sok segítséget és támogatást kaptunk tőlük. A tanulmányaink ideje alatt megszerzett tudást nagyban fel tudtuk használni a projekt elkészítéséhez, ezáltal felgyorsult az elkészítési folyamat.

Az oldal bárki számára elérhető a <https://eltuntallatkereso.hu/> címen. A projektmunkát végig a Git verziókezelővel követtük nyomon, és a Github oldalán megtalálható egy publikus „repository”-ban. A <https://github.com/mtcsrht/lostanimal-project> címen érhető el ez a repository, itt minden változtatás nyomon követhető és egy kisebb angol nyelvű dokumentáció is található a telepítéséhez, illetve, hogy mi szükséges a projekt elindításához.

1. Felhasznált technológiák

1.1. PHP

A PHP (Hypertext Preprocessor) egy szerveroldali szkriptnyelv, amelyet főként dinamikus weboldalak és webalkalmazások létrehozására használnak. A PHP-t arra tervezték, hogy lehetővé tegye a programozók számára, hogy dinamikusan generáljanak tartalmakat, interakciókat és adatbázis-kapcsolatokat hozzanak létre webes környezetben. A PHP egy rugalmas és erőteljes eszköz a webfejlesztés terén, amely lehetővé teszi a fejlesztők számára a dinamikus és interaktív weboldalak készítését.

1.2. HTML

A HTML (Hypertext Markup Language) egy olyan kódolási nyelv, amelyet a weboldalak strukturálására és tartalom formázására használnak. A HTML egy alapvető eszköz a webfejlesztésben, és lehetővé teszi a fejlesztők számára, hogy létrehozzanak statikus és dinamikus weboldalakat. A HTML egyszerű és intuitív nyelv, amely lehetővé teszi a fejlesztők számára a weboldalak létrehozását és formázását. Összességében a HTML a webes tartalom alapját képezi, és nélkülözhetetlen eszköz a webfejlesztésben.

1.3. CSS

A CSS (Cascading Style Sheets) egy stíluslapnyelv, amelyet a weboldalak megjelenésének formázására és stílusának meghatározására használnak. A CSS segítségével a fejlesztők különállóan tudják kezelni egy weboldal megjelenését és stílusát, így lehetővé teszik a tartalom és a formázás elkülönítését a HTML-től. A CSS nagyban hozzájárul a weboldalak esztétikus megjelenéséhez és a felhasználói élmény javításához. Összességében a CSS egy elengedhetetlen eszköz a webfejlesztésben, amely lehetővé teszi a fejlesztők számára a weboldalak kreatív és egyedi megjelenésének létrehozását.

1.4. JavaScript

JavaScript egy programozási nyelv, melyet elsősorban weboldalak és webalkalmazások interaktív funkcióinak létrehozására használnak. Elsődlegesen a

klients-oldalon fut, ami azt jelenti, hogy a böngészőben futó kód formájában érkezik a felhasználók eszközeire, és a böngésző motorja dolgozza fel. JavaScript nélkül a weboldalak statikusak és kevésbé interaktívak lennének. A JavaScript kulcsfontosságú szerepet játszik a webfejlesztésben, lehetővé téve az élő, interaktív és dinamikus felhasználói élmények létrehozását.

1.5. Node.js

A Node.js egy nyílt forráskódú, szerveroldali JavaScript futtatókörnyezet, amely lehetővé teszi a JavaScript kód futtatását a szerver oldalán. A Node.js alapvetően egy V8 JavaScript motorra épül, amelyet a Google fejlesztett ki, és lehetővé teszi a JavaScript kód gyors és hatékony futtatását. A Node.js-nél a JavaScript nyelvet a szerver-oldalon is használhatjuk, ami egységes kódolási környezetet biztosít a fejlesztők számára. Ezzel a hatékony és skálázható megoldással a Node.js széles körben elterjedt a webfejlesztők körében, és sokféle alkalmazás-fejlesztési feladatra használják.

1.6. Composer

A Composer egy PHP csomagkezelő és függőségkezelő eszköz, amelyet a PHP alkalmazások fejlesztéséhez és kezeléséhez használnak. A Composer lehetővé teszi a külső könyvtárak és csomagok könnyű telepítését és kezelését a PHP projektjeinkben. A Composer segítségével a PHP fejlesztők könnyedén és hatékonyan kezelhetik a projektjeik függőségeit és külső csomagjait, ami lehetővé teszi a gyorsabb és hatékonyabb fejlesztést. Ezáltal a Composer fontos eszköz a PHP közösségben, és széles körben használják a PHP projektek kezeléséhez és fejlesztéséhez.

1.7. Tailwind CSS

A Tailwind CSS egy modern, kiterjeszthető és testreszabható CSS keretrendszer, amely segítségével könnyen és gyorsan lehet készíteni reszponzív webes felhasználói felületeket. A Tailwind CSS eltér a hagyományos CSS keretrendszerektől, mivel nem előre definiált stílusosztályokat kínál, hanem egy készletet alacsony szintű építőelemekkel és alapvető stílusokkal, amelyeket kombinálhatunk és testreszabhatunk az egyedi igényeink szerint. A Tailwind CSS egy hatékony és kreatív eszköz a modern

webfejlesztéshez, amely lehetővé teszi a gyors és hatékony felhasználói felületek készítését, testreszabását és optimalizálását a projekt igényei szerint.

1.8. Vite

A Vite egy modern, gyors és kiterjeszthető fejlesztői eszközrendszer, amelyet főként webalkalmazások és weboldalak fejlesztéséhez használnak. A Vite célja, hogy egyszerűsítse és felgyorsítsa a modern JavaScript projektek fejlesztését és kezelését, különösen azokat, amelyek a Vue.js, React vagy más JavaScript keretrendszereken alapulnak. A Vite egy hatékony és rugalmas fejlesztői eszköz, amely lehetővé teszi a modern webalkalmazások és weboldalak fejlesztését és kezelését. A fejlesztők számára gyors és hatékony megoldást kínál a kódszerkesztésre, tesztelésre és optimalizálásra a projektjeikben.

1.9. Visual Studio Code

A Visual Studio Code (VS Code) egy ingyenes, nyílt forráskódú, könnyen testre szabható és kiterjeszthető fejlesztői környezet (IDE), amelyet elsősorban szoftverfejlesztők használnak különböző programozási nyelvekben készült alkalmazások fejlesztésére. A VS Code sokoldalú eszköz, amely rengeteg funkciót és szolgáltatást kínál, hogy megkönnyítse a kódolást és a projektmenedzsmentet. A VS Code egy nagyon népszerű és sokoldalú fejlesztői környezet, amelyet széles körben használnak szoftverfejlesztők a kódolás, szerkesztés, debuggolás és projektmenedzsment céljából. Ez az eszköz segítséget nyújt a hatékonyabb és produktívabb fejlesztéshez különböző programozási nyelvekben és projekteken.

1.10. XAMPP

A XAMPP egy ingyenes, nyílt forráskódú szoftvercsomag, amelyet főként webfejlesztők használnak lokális fejlesztői környezet kialakításához. A neve rövidítése a "X" a Cross-platform, az "Apache" webservert, a "MySQL" adatbázist, a "PHP" programnyelvet, és a "Perl" programnyelvet. A XAMPP egy hatékony eszköz a webfejlesztők számára, amely lehetővé teszi a lokális fejlesztői környezet gyors és egyszerű kialakítását, és lehetőséget nyújt a PHP és MySQL alapú webalkalmazások és weboldalak fejlesztésére és tesztelésére.

1.11. MySQL

A MySQL egy nyílt forráskódú, relációs adatbázis-kezelő rendszer, amelyet gyakran használnak webalkalmazások és weboldalak adatbázisának tárolására és kezelésére. A MySQL széles körben elterjedt, és sokan preferálják a könnyű telepítésének, kezelésének és megbízhatóságának köszönhetően. A MySQL egy megbízható és kifinomult adatbázis-kezelő rendszer, amely lehetővé teszi a hatékony adatbázis-kezelést és adatok tárolását webalkalmazásokban és weboldalakban. Összességében a MySQL egy elterjedt és népszerű választás a relációs adatbázis-kezelő rendszerek között a webfejlesztés terén.

1.12. GitHub

A GitHub egy webes alapú verziókezelő platform és szolgáltatás, amelyet főként szoftverfejlesztők használnak a projektmenedzsment, a kódmegosztás és a verziókezelés céljából. A GitHub lehetővé teszi a fejlesztők számára, hogy együttműködjenek a különböző projektjeiken, nyomon kövessék a változásokat és egyeztessék a munkájukat. A GitHub egy fontos eszköz a szoftverfejlesztés terén, amely lehetővé teszi a hatékonyabb együttműködést, verziókezelést és projektmenedzsmentet a fejlesztők számára. Sok fejlesztő és szervezet számára alapvető eszköz a projektjeik kezelésében és nyomon követésében.

1.13. WinSCP

A WinSCP egy ingyenes, nyílt forráskódú FTP, SFTP, SCP és WebDAV kliens Windows rendszerekhez. Ez a program lehetővé teszi a fájlok biztonságos és gyors átvitelét a számítógép és egy távoli szerver között, és támogatja a különböző protokollokat a fájlok eléréséhez és kezeléséhez. A WinSCP egy hatékony és megbízható eszköz a fájlok távoli kezeléséhez és átviteléhez, amely gyakran használatos webfejlesztők és rendszergazdák által. Segítségével könnyen és biztonságosan lehet fájlokat kezelni és átvinni a távoli szerverek és a felhasználók között.

1.14. Dbdiagram

A "Dbdiagram" egy online eszköz, amelyet relációs adatbázisok sématervezésére és modellezésére használnak. Ez az eszköz lehetővé teszi a fejlesztők és adatbázis-

tervezők számára, hogy könnyen és intuitívan hozzanak létre adatbázis-sémákat, és vizuális eszközöket biztosítanak az adatbázis struktúrájának megértéséhez és dokumentálásához. Összességében a Dbdiagram egy hatékony eszköz az adatbázisok sématervezéséhez és modellezéséhez, amely segít a fejlesztőknek és tervezőknek a struktúra pontos meghatározásában és dokumentálásában a fejlesztési folyamat során.

1.15. Figma

A Figma egy online alapú tervezőeszköz, amelyet főként interfész- és terméktervezők használnak. Ez az eszköz lehetővé teszi a tervezők számára, hogy együtt dolgozzanak, megosszák és egyeztessék a terveiket és projektjeiket azáltal, hogy közösen dolgozhatnak ugyanazon dokumentumokon és fájlokon. A Figma széles körben elterjedt és népszerű a tervezők körében, mivel lehetővé teszi a hatékony együttműködést és az interaktív prototípusok készítését. Az összefoglalóban elmondható, hogy a Figma egy kiterjedt és sokoldalú tervezőeszköz, amely lehetővé teszi a hatékony és együttműködő interfésztervezést és prototípuskészítést. Sok tervező és tervezőcsapat használja a Figma-t a digitális termékek tervezéséhez és fejlesztéséhez.

1.16. .NET Keretrendszer

A .NET Keretrendszer egy szoftverfejlesztési platform és keretrendszer, amelyet a Microsoft fejlesztett ki, és különböző programozási nyelveket és technológiákat kínál a szoftveralkalmazások fejlesztéséhez. A .NET Keretrendszer széles körben elterjedt és nagyon népszerű a vállalati környezetben és az ipari alkalmazások fejlesztésében. Összességében a .NET Keretrendszer egy kiterjedt és sokoldalú platform, amely lehetővé teszi a fejlesztők számára a szoftveralkalmazások gyors és hatékony fejlesztését különböző célokra és környezetekben. A .NET keretrendszer egyik fő előnye a fejlett eszközök, nyelvi támogatás és keresztplatformos képességek biztosítása a fejlesztők számára.

1.17. .NET C#

A .NET C# a Microsoft által fejlesztett C# programozási nyelvnek a .NET keretrendszeren belüli implementációja. A C# egy modern, általános célú, típusbiztos, objektumorientált programozási nyelv, amely erősen kapcsolódik a .NET

keretrendszerhez, és széles körben használják a szoftverfejlesztésben. Az összefoglalóban elmondható, hogy a C# egy erőteljes és rugalmas programozási nyelv, amely lehetővé teszi a fejlesztők számára a különböző típusú szoftveralkalmazások hatékony és produktív fejlesztését. A C# egyik fő előnye a gazdag nyelvi funkciók és az erős integráció a .NET keretrendszerrel, amely lehetővé teszi a széles körű alkalmazások fejlesztését különböző platformokon.

1.18. Microsoft Word

A Microsoft Word egy népszerű és széles körben használt szövegszerkesztő program, amely része a Microsoft Office csomagnak. Ez az alkalmazás lehetővé teszi a felhasználók számára, hogy különböző típusú dokumentumokat hozzanak létre, szerkesszenek és formázzanak, például leveleket, beszámolókat, prezentációkat, önéletrajzokat és még sok más. Az összefoglalóban elmondható, hogy a Microsoft Word egy sokoldalú szövegszerkesztő program, amely számos funkciót és eszközt kínál a felhasználóknak a dokumentumok írásához, szerkesztéséhez és formázásához. A Word népszerűsége annak köszönhető, hogy intuitív felhasználói felülettel rendelkezik és széles körben elérhető különböző platformokon.

1.19. Teams

A "Teams" a Microsoft Teams nevű alkalmazásra utal, ami egy összekapcsolt munkahelyi kommunikációs és együttműködési platform. Az alkalmazás lehetővé teszi a csapatoknak, hogy közvetlenül kommunikáljanak egymással, együtt dolgozzanak dokumentumokon, megoszthassák információikat és együttműködjenek projektjeiken.

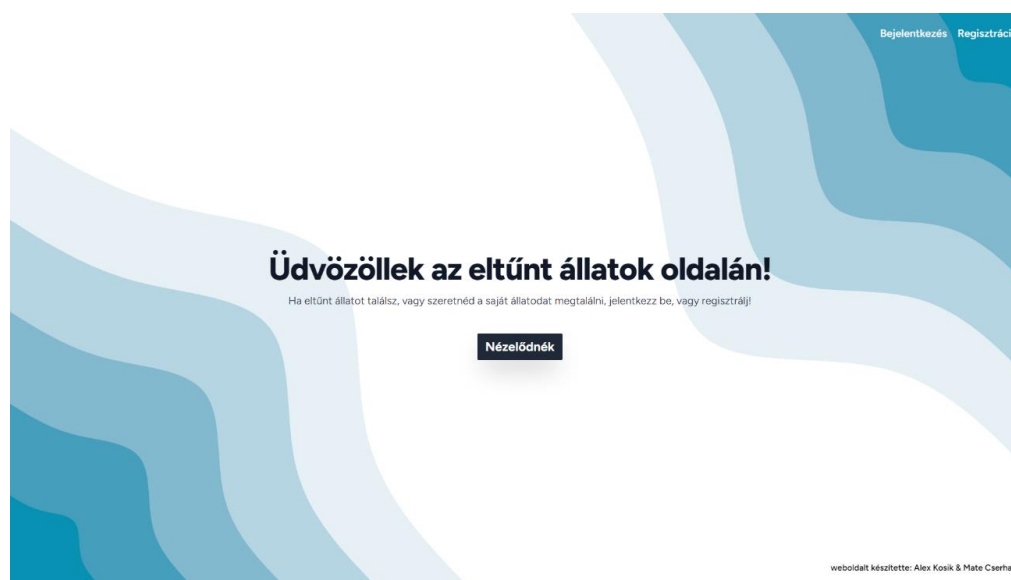
A Microsoft Teams egy hatékony eszköz a csapatmunka támogatására és a virtuális együttműködés elősegítésére. Az alkalmazás széles körben használt a vállalati környezetben és egyre népszerűbb a távoli munkavégzés és a rugalmas munkahelyi környezet növekvő igényei miatt.

2. Felhasználói dokumentáció

Az adatbázist és a háttérrendszert Cserhádi Máté tervezte és fejlesztette, míg Kosik Alex volt felelős a weboldal megjelenéséért és további fejlesztéséért. Az oldalunk minden részén egységesen ugyanazokat a sablon színeket alkalmazzuk, ezáltal biztosítva az egységes megjelenést és felhasználói élményt. Ez a szoros együttműködés és egységesített megközelítés garantálja, hogy minden oldal harmonikusan illeszkedjen a weboldal designjához, és egységes képet mutasson a felhasználók számára.

2.1. Kezdőoldal

A főoldal megtervezése során különös figyelmet fordítottam arra, hogy minden elem reszponzív módon legyen elhelyezve és megjelenjen, így biztosítva, hogy az oldal bármilyen eszközről, legyen az táblagép vagy okostelefon, kényelmesen és átláthatóan legyen olvasható és használható. A főoldalon elhelyeztem egy köszöntő szöveget, amely középre van igazítva, valamint egy "Nézelődnék" gombot, amelyre az egérmutató rávittelekor látványos effekt jelenik meg, hogy felhívja a figyelmet a gomb funkciójára. A regisztráció és bejelentkezés lehetőségei a főoldal tetején, jobb oldalon találhatóak. A tervezés során külön figyelmet fordítottunk arra, hogy a telefonos nézetben minden elem tökéletesen látható és olvasható legyen. Emellett az összes oldalon elérhető a sötét mód lehetősége is, melyet fontosnak tartottunk a szemek védelme érdekében. Így biztosítva van, hogy az oldalunk minden felhasználója számára kényelmes és kellemes élményt nyújtson bármilyen eszközön és környezetben.



1. ábra Főoldal



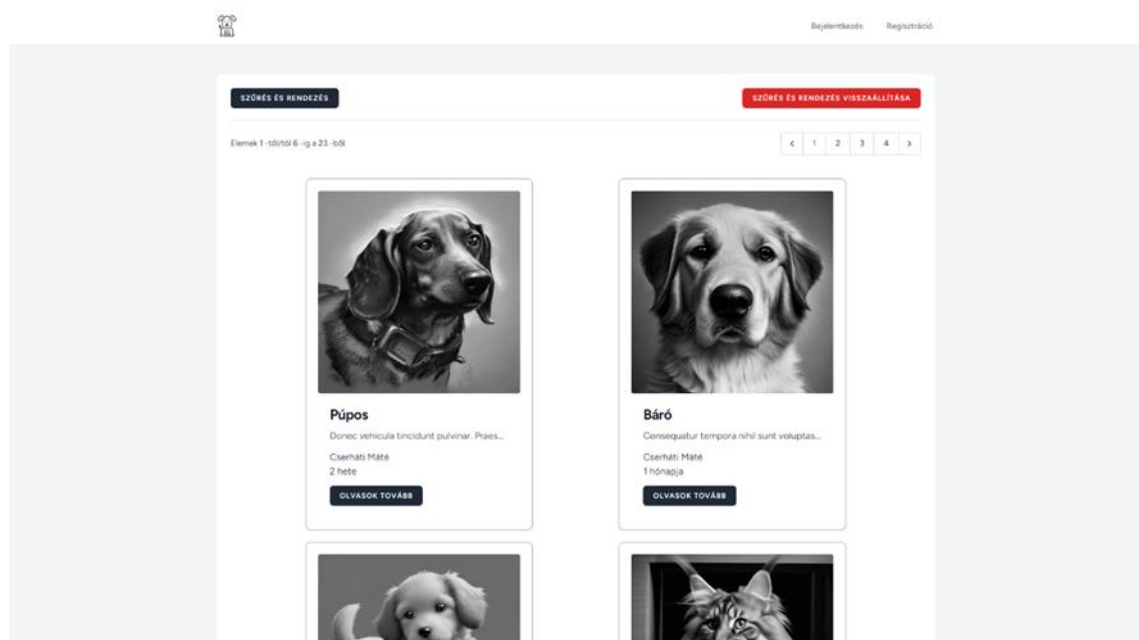
2. ábra Mobil nézet és sötét mód

2.2. Publikus posztok

A főoldalon található "Nézelődnék" gomb egy olyan oldalra vezet minket, ahol az összes felhasználó által megosztott posztot láthatjuk. Ennek az oldalnak a megtekintéséhez nem szükséges regisztrálni vagy bejelentkezni, így bárki könnyedén böngészheti az ott található tartalmat. Azért választottuk ezt a megoldást, hogy mindenki számára hozzáférhető legyen a tartalom, még regisztráció nélkül is. Így a látogatók előzetes kötelezettségek nélkül megismerhetik az oldalunkat, és csak akkor regisztrálnak, ha valóban érdekli őket a tartalom és a további lehetőségek a weboldalon. Ezáltal lehetőséget biztosítunk a látogatóknak arra, hogy maguk dönthessenek a szabad böngészésre, mielőtt a regisztráció mellett döntenek.



3. ábra Publikus posztok mobil nézetben



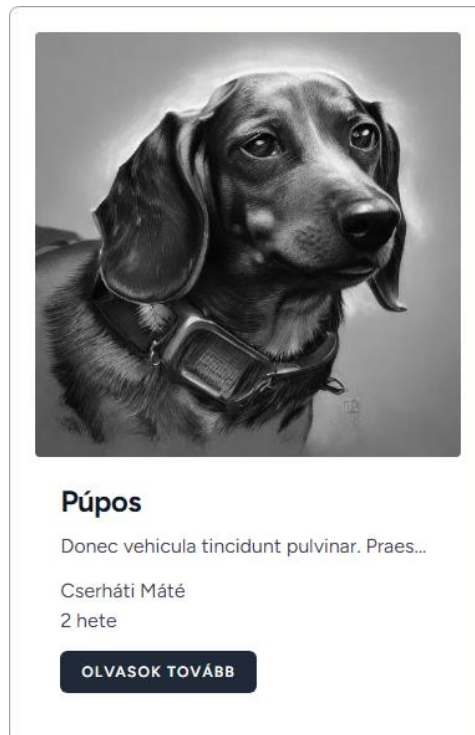
4. ábra Publikus posztok asztali nézetben

2.2.1. Lapozó

Az oldal tetején elhelyezett lapozó segítségével könnyedén tudunk lapozni az oldalon, mivel egyszerre csak hat posztot jelenítünk meg. Ez a megoldás lehetővé teszi számunkra, hogy a felhasználók könnyen és gyorsan áttekinthessék a tartalmat, miközben elkerüljük a végtelen görgetést az oldalon. A hat poszt egyidejű megjelenítése segít megőrizni az oldal átláthatóságát és könnyű navigálhatóságát, anélkül, hogy túlterhelnénk a felhasználót felesleges információkkal. Ezáltal javítjuk az oldal felhasználói élményét, és segítünk a látogatóknak hatékonyabban böngészni és megtalálni azt, ami érdekli őket.

2.2.2. Publikus kártyák

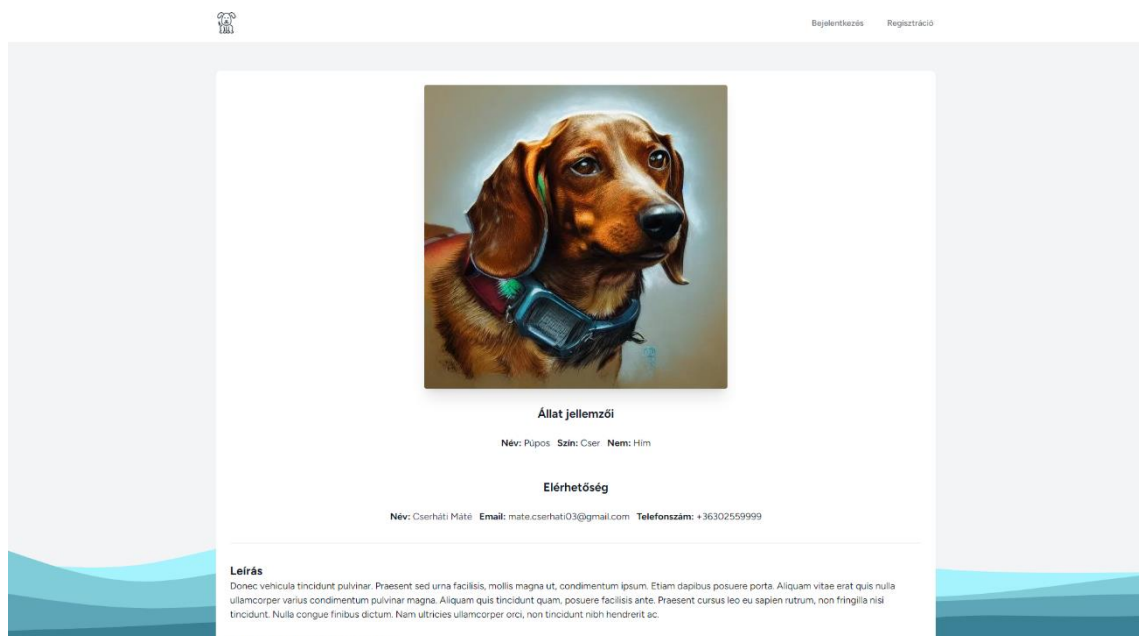
Egy feltöltött poszton számos információval találkozhatunk, de leginkább egy nagy méretű, figyelemfelkeltő kép tűnik szembe velünk. Ez a kép az eltűnt állatot ábrázolja, és egy filter segítségével fekete-fehéren jelenik meg. Azonban ha az egérmutatót a kép fölé helyezzük, láthatjuk az eredeti színét is. A kép alatt az állat neve található, melynek mérete nagyobb, hogy kiemelkedjen, és könnyen olvasható legyen. A név alatt rövid leírás található az állatról, bár csak egy részletét látjuk annak. A leírás alatt láthatjuk a feltöltő nevét, valamint azt, hogy mikor került feltöltésre a poszt. Ezt követően egy gomb található, melyre kattintva egy új oldalon megjelenik az állat teljes adatlapja, biztosítva ezzel az érdeklődők számára, hogy több információt tudjanak meg az állatról.



5. ábra Publikus kártya

2.2.3. Állat adatlapja

Az oldalon minden lényeges információt megtalálunk az adott állatról. Először is, láthatjuk az állat képét, melyre kattintva megjelenik a teljes méretű változat. A kép alatt szerepelnek az állat jellemzői, mint például a neve, színe és neme. Ezt követően található az elérhetőség rész, ahol megtekinthetjük a feltöltő adatait, mint például a felhasználó nevét, e-mail címét és telefonszámát. Ezek az információk további funkciókat is tartalmaznak. Ha a névre kattintunk, azonnal átirányít minket a chat ablakba, ahol üzenetet küldhetünk a feltöltőnek. Ha az e-mail címre kattintunk, megjelenik az e-mail küldés lehetősége, míg a telefonszámra kattintva telefonon is felvehetjük a kapcsolatot a feltöltővel. Ezáltal a felhasználók könnyen és gyorsan kapcsolatba léphetnek egymással, az állatokkal kapcsolatos további információk és egyéb ügyek intézése céljából.



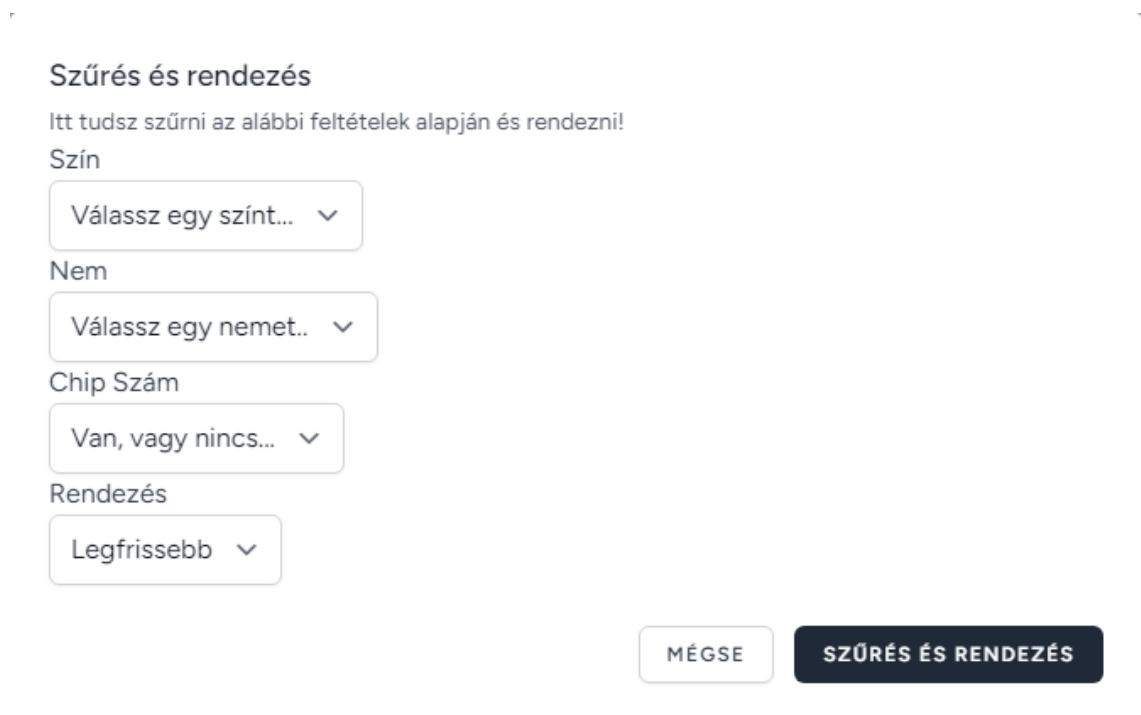
6. ábra Állat adatlapja

2.2.4. Szűrés

Amikor visszatérünk a publikus posztokhoz az oldalon, észrevehetjük, hogy a felső részen található egy "Szűrés és rendezés" gomb. Ennek segítségével könnyen beállíthatunk szűrőfeltételeket az oldalon található tartalmakra. Amikor rákattintunk erre a gombra, megjelenik egy felugró ablak, amelyben több lehetőségünk van a szűrésre. Például színek alapján szűrhetünk, vagy arra is van lehetőségünk, hogy csak bizonyos típusú tartalmakat lássunk. Továbbá, chip szám alapján is keresni tudunk. Emellett lehetőségünk van rendezésre is, például azt kérhetjük, hogy a legfrissebb posztok jelenjenek meg először. A szűrés indításához válasszuk ki a megfelelő opciókat a felugró

ablakban, majd kattintsunk a "Szűrés és rendezés" gombra az ablak alján. Ha mégsem szeretnénk szűrni, egyszerűen válasszuk a "Mégse" gombot a felugró ablak bezárásához.

Amennyiben már beállítottunk szűrőket, de szeretnénk visszatérni az eredeti, összes posztot tartalmazó nézethez, akkor egyszerűen válasszuk a "Szűrés és rendezés visszavonása" lehetőséget. Ezzel visszavonhatjuk az összes korábban beállított szűrőt, így újra láthatjuk az összes posztot az oldalon, anélkül hogy valamilyen kritériumnak megfelelően lennének szűrve vagy rendezve. Ezáltal visszaállíthatjuk az alapértelmezett megjelenítést.



7. ábra Szűrés felugró ablaka

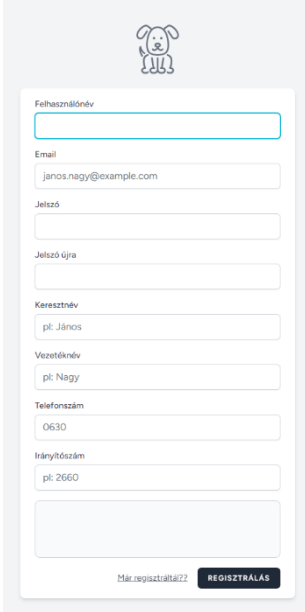
2.3. Regisztráció

Az oldalunk felső részén elhelyezett opciók között megtalálható a "Regisztráció" és a "Bejelentkezés" lehetősége. Ha még nincs fiókunk, akkor kattintsunk a "Regisztráció" gombra. Miután erre a gombra kattintottunk, megjelenik a regisztrációs űrlap. Itt először meg kell adnunk egy felhasználónevet. A felhasználónév tetszőleges, bármit választhatunk. Ezután következik az email cím megadása. Fontos, hogy olyan email címet adjunk meg, amit rendszeresen használunk, mivel a regisztrációt követően egy megerősítő emailt fogunk kapni, amelyben vissza kell igazolnunk a regisztrációt. Miután megadtuk az email címet, következik a jelszó megadása. Fontos, hogy a jelszó tartalmazzon legalább egy nagybetűt és egy számot, valamint minimum nyolc karakter

hosszúnak kell lennie. A biztonságos jelszó kiválasztása érdekében figyeljünk erre a követelményre. Ezt a jelszót a következő opcióban is meg kell adnunk, hogy megerősítsük a helyes megadást. Miután megadtuk a jelszót, következik a kereszt- és vezetéknév megadása. Ezután sor kerül a telefonszámunk és végül az irányítószámunk megadására. Amikor elkezdjük az irányítószámunkat, a címünket, a rendszer automatikusan fel fogja dobni a lehetséges választási lehetőségeket. Amikor megtaláltuk a saját lakóhelyünk nevét és irányítószámát, egyszerűen kattintsunk rá, és a rendszer automatikusan beilleszti az irányítószám megadási lehetőségéhez. Ezáltal egyszerűbbé válik a címünk pontos megadása és elkerülhetők a helyesírási hibák.

Miután kitöltöttük az összes szükséges mezőt, kattintsunk a "Regisztráció" gombra. Amennyiben nem kapunk hibaüzenetet, egy köszöntő üzenet fog megjelenni. Az üzenet alján található egy "Megerősítő email újraküldése" gomb, amire akkor van szükségünk, ha nem kaptunk meg azonnal egy megerősítő emailt a regisztrációhoz. Ez a lehetőség segít abban, hogy újra küldjék az emailt, amelyben megerősíthetjük a regisztrációt, ha esetleg valami okból kifolyólag nem érkezett meg az eredeti email.

Ellenőrizzük az email fiókunkat, hogy megkaptuk-e az állat-keresőtől érkezett emailt. Ha igen, kattintsunk rá. Az emailben található egy üzenet. Az üzenet alatt egy gomb lesz, amire kattintanunk kell annak érdekében, hogy megerősítsük az email címünket. Ha ez sikeres, a gombra kattintás után visszairányít minket az oldalunkra, ahol már számos új lehetőség vár ránk.

A screenshot of a registration form on a light gray background. At the top center is a small icon of a dog's head. The form contains several input fields: 'Felhasználónév' (empty), 'Email' (filled with 'janos.nagy@example.com'), 'Jelszó' (empty), 'Jelszó újra' (empty), 'Keresztnév' (filled with 'pl: János'), 'Vezetéknév' (filled with 'pl: Nagy'), 'Telefonszám' (filled with '0630'), and 'Irányítószám' (filled with 'pl: 2660'). Below these fields is a checkbox labeled 'Már regisztráltam?' and a dark gray button labeled 'REGISZTRÁCIÓ'.

8. ábra Regisztráció

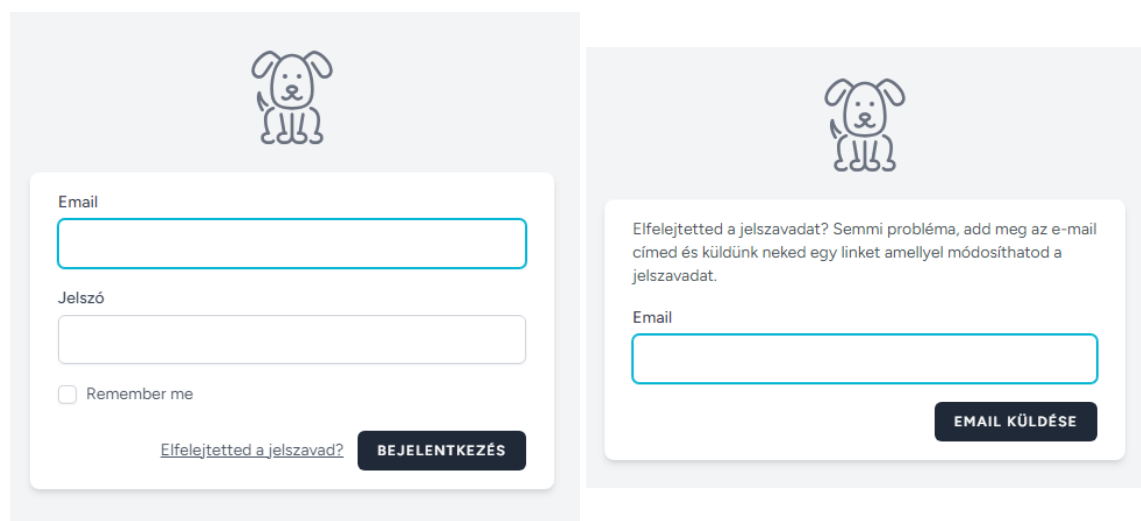
2.4. Bejelentkezés

Amennyiben már van regisztrált fiókunk, kattintsunk a "Bejelentkezés" lehetőségre. Ezután megjelenik előttünk a bejelentkezési űrlap, ahol meg kell adnunk a felhasználónevet és a jelszót, amelyet a regisztrációkor rögzítettünk. Ha ezeket az adatokat megadtuk, kattintsunk a "Bejelentkezés" gombra. Ezzel beléphetünk a fiókunkba, és hozzáférhetünk az oldalon elérhető funkciókhoz és szolgáltatásokhoz.

Ha elfelejtettük a jelszavunkat, akkor kattintsunk az "Elfelejtetted a jelszavad?" lehetőségre. Ekkor megjelenik egy űrlap, ahol meg kell adnunk az email címünket, amit a regisztráció során rögzítettünk. Miután beírtuk az email címet, kattintsunk az "Email küldése" gombra. Ezzel a rendszer elküld egy emailt az általunk megadott címre, amelyben útmutatást és lehetőséget kapunk arra, hogy visszaállítsuk a jelszavunkat.

Amint megkapjuk az emailt, észrevehetjük benne a "Jelszó visszaállítása" gombot. Ez a gomb lehetőséget biztosít arra, hogy új jelszót állítsunk be a fiókunkhoz.

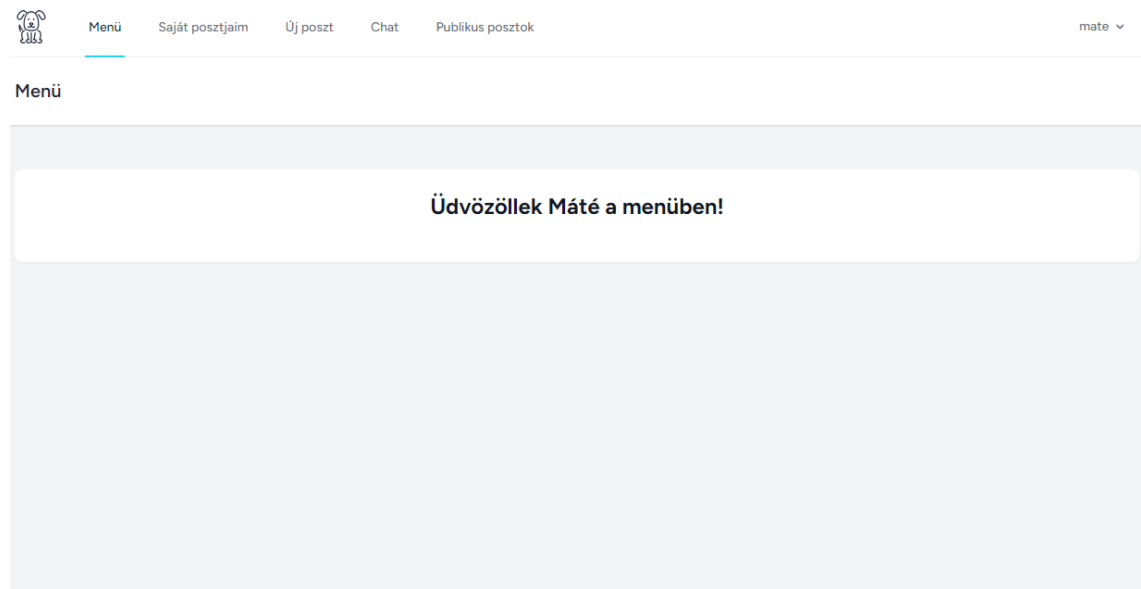
Miután rákattintottunk a "Jelszó visszaállítása" gombra az emailben, egy űrlap jelenik meg előttünk. Az email mezőben már automatikusan megjelenik a korábban megadott email címünk. Ezután meg kell adnunk az új jelszót a megfelelő mezőbe, majd meg kell ismételni azt a következő mezőben. Fontos megjegyezni, hogy az új jelszónak tartalmaznia kell legalább egy nagybetűt és egy számot, valamint legalább nyolc karakter hosszúnak kell lennie. Ezután kattintsunk a "Jelszó módosítása" gombra az új jelszó beállításához.



9. ábra Bejelentkezés és elfelejtett jelszó

2.5. Menü


Miután sikeresen regisztráltunk vagy bejelentkeztünk, az első oldal, amit látni fogunk, egy személyre szabott üdvözlő üzenetet tartalmaz. Az oldal tetején található egy menüsáv, amely számos lehetőséget kínál az oldalon. A jobb felső sarokban megjelenik a nevünk, és ha erre kattintunk, további két lehetőség jelenik meg előttünk.



10. ábra Főmenü

2.5.1. Új poszt

A menüsávban kattintsunk az "Új poszt" lehetőségre. Ezután egy új űrlapot láthatunk. Az első opcióban meg kell adnunk az állat nevét, majd a következő opcióban az állat nemét (hím vagy nőstény). Ezt követően lehetőségünk van feltölteni egy képet az állatunkról, amelynek ideális mérete 512 * 512 pixel. Ehhez kattintsunk a "Kép feltöltése" gombra, majd válasszuk ki a megfelelő képet. A következő opció a chip szám megadása, amely opcionális, de ha rendelkezünk vele, érdemes megadni. Ezután választhatjuk az állat színét egy legördülő menüből. Végül a leírás mezőben írhatunk bármilyen további információt az állatról, például ismertető jegyeket vagy, hogy hol és mikor tűnt el. Miután kitöltöttük az összes opciót, kattintsunk a "Feltöltés" gombra, hogy elküldjük a posztot.

MenüSaját posztjaimÚj posztChatPublikus posztokmate ▾

Új poszt

Állat feltöltése

Töltsd ki az állatod adataival, hogy feltöltsd a rendszerbe.

Állat neve

Chip száma (amennyiben az állat rendelkezik vele)

Állat neme

Hím ▾

Állat színe

Barna ▾

Kép az állatról (512x512)

Choose FileNo file chosen

Leírás

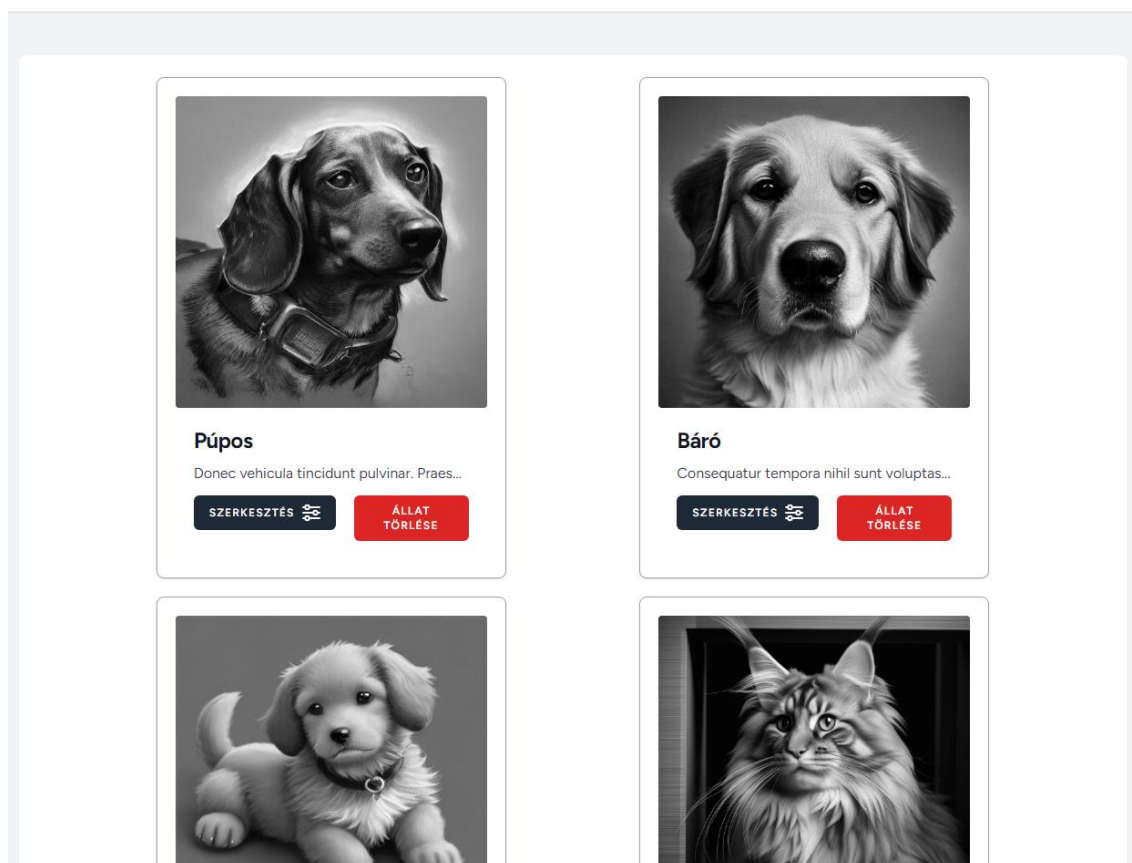
Írj az állatodról.

FELTÖLTÉS

11. ábra Állat feltöltése

2.5.2. Saját posztjaim

Miután sikeresen feltöltöttük az állatunkkal kapcsolatos posztot, kattintsunk a "Saját posztjaim" lehetőségre. Itt láthatjuk a korábban feltöltött állatot. A posztban megjelenik az állatról a feltöltött kép, egy rövid leírás, valamint két gomb: az első a "Szerkesztés", amelyre kattintva módosíthatjuk a posztunk tartalmát, ha szükséges. A másik gomb az "Állat törlése", amelyet akkor válasszunk, ha az állatunk előkerült, és már nem szükséges továbbra is a keresőlistán szerepelnie. Ezáltal törölhetjük az adott posztot az oldalról.



12. ábra Saját posztok

2.5.3. Szerkesztés

Kattintsunk a "Szerkesztés" gombra. Ezt követően megjelenik egy űrlap, amely nagyon hasonlít a feltöltési űrlaphoz. Az űrlapon automatikusan megjelennek az előre megadott információk, például az állat neve, neme, a feltöltött kép, a chip szám, a szín és a leírás. Ha nem kívánunk minden adatot módosítani, akkor nem kötelező azokat megváltoztatni. Csak hagyjuk azokat úgy, ahogy vannak, és nem kell hozzájuk nyúlni a szerkesztés során. Ezáltal csak azokat az információkat módosíthatjuk, amelyeket valóban szükségesnek tartunk. Ha végeztünk a módosításokkal, kattintsunk a "Mentés" gombra. Ezáltal az újonnan beállított vagy módosított adatok el lesznek mentve a posztban, és azok azonnal érvénybe lépnek az oldalon.



Poszt szerkesztése

Állat szerkesztése

Módosítsd az állatot ahogy csak szeretnéd!

Állat neve

Püpos

Chip száma (amennyiben az állat rendelkezik vele)

657452349565432

Állat neme

Hím ▾

Állat színe

Cser ▾


Kép az állatról (512x512)

Choose File No file chosen

Leírás

Donec vehicula tincidunt pulvinar. Praesent sed urna facilisis, mollis magna ut, condimentum ipsum. Etiam dapibus posuere porta. Aliquam vitae erat quis nulla ullamcorper varius condimentum pulvinar magna. Aliquam quis tincidunt quam, posuere facilisis ante. Praesent cursus leo eu sapien rutrum, non fringilla nisi tincidunt. Nulla congue finibus dictum. Nam ultricies ullamcorper orci, non tincidunt nibh hendrerit ac.

Jelenlegi kép (ha nem szeretnél változtatni ne tölts fel semmit!)

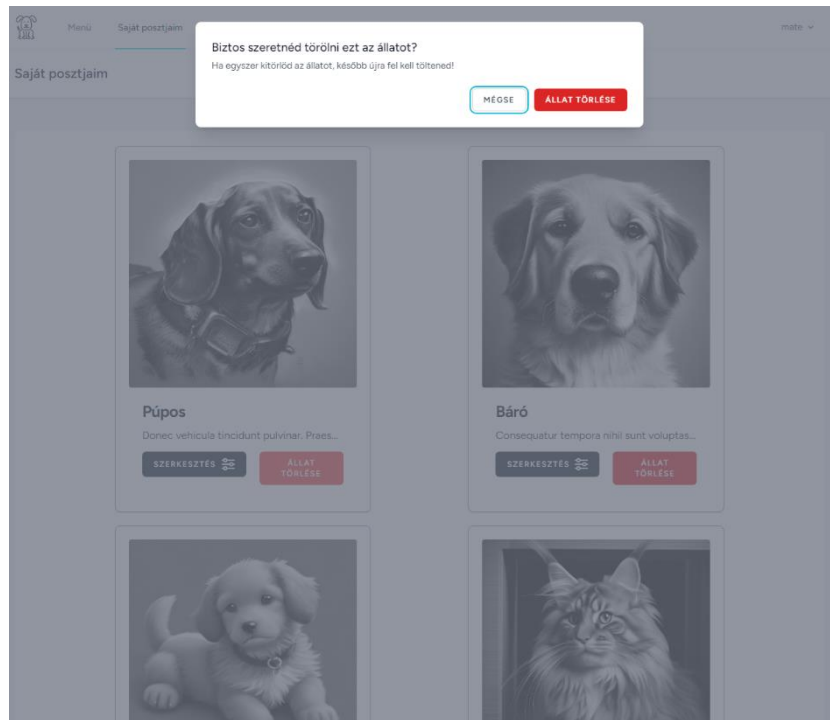


MENTÉS

13. ábra Poszt szerkesztése

2.5.4. Állat törlése

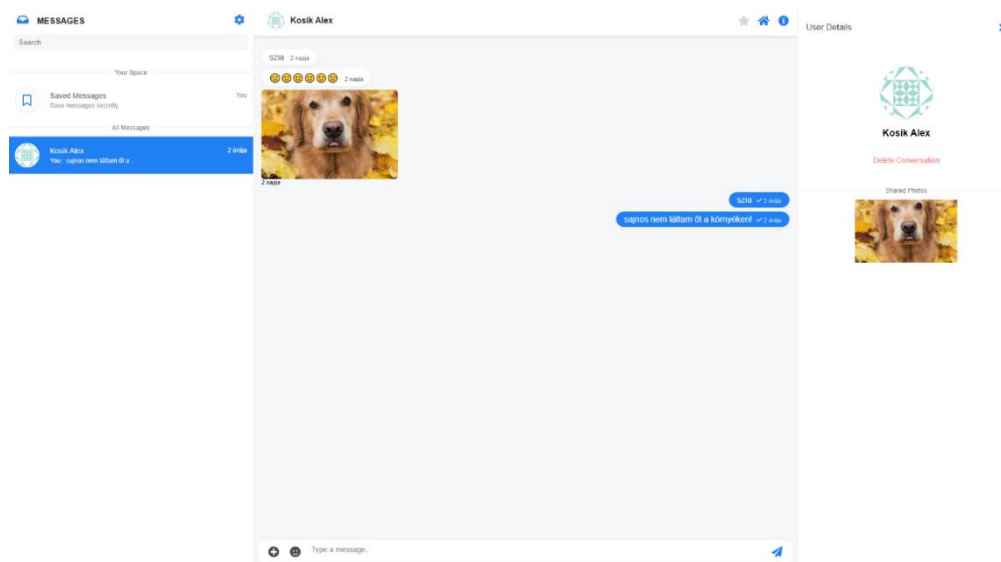
Az "Állat törlése" gombra kattintva megjelenik egy felugró ablak, amelyben megerősítést kérnek tőlünk arról, hogy törölni szeretnénk-e az állat posztját. Ha mégsem szeretnénk törölni az állatot, akkor válasszuk a "Mégse" lehetőséget, ami visszaléptet minket a saját posztjainkhoz, így nem lesz törlés. Ha viszont valóban törölni szeretnénk az állatot, akkor az "Állat törlése" gombra kattintva erősítsük meg a törlést. Ezzel véglegesen eltávolítjuk az állat posztját az oldalról.



14. ábra Állat törlése előtt felugró ablak

2.6. Chat

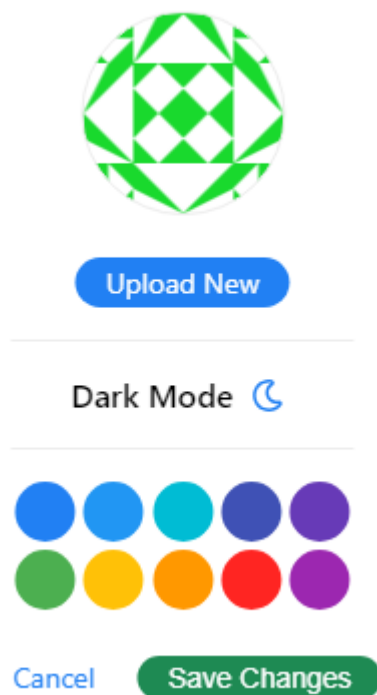
A menüsávban található "Chat" lehetőségre kattintva egy új oldalra jutunk, ahol üzenetet küldhetünk más felhasználóknak. Az oldal jobb oldalán elhelyezett kereső segítségével megtalálhatjuk az adott felhasználót, akinek üzenetet szeretnénk küldeni. Miután megtaláltuk a kívánt felhasználót, kattintsunk a nevére. Ezután az oldal alján található egy üzenetíró ablak, ahol megírhatjuk az üzenetünket. Itt található egy emoji ikon is, amire kattintva számos emoji-t választhatunk ki és használhatunk az üzenetben. Emellett látható egy plusz jel ikon is, amely segítségével fájlokat és képeket küldhetünk a felhasználónak. Az elküldött üzenet mellett láthatjuk, hogy a felhasználó már megtekintette-e, valamint azt is, hogy hány perce küldtük az üzenetet. Az oldal jobb oldalán látható a felhasználó neve, alatta pedig az eddig elküldött képek láthatók.



15. ábra Chat

2.6.1. Chat beállítások

Az oldal jobb felső sarkában található fogaskerék ikonra kattintva egy felugró ablak jelenik meg előttünk. Ebben az ablakban lehetőségünk van számos beállítás módosítására. Például beállíthatjuk a sötét módot, amely a sötét témát aktiválja az oldalon. Emellett van lehetőségünk a chat színét is állítani az egyéni preferenciák szerint. Továbbá ezen az ablakon keresztül tudunk profilképet is feltölteni, ha szeretnénk egy személyre szabottabb megjelenést biztosítani a profilunknak. Ha beállítottuk az összes preferenciánkat, amit szeretnénk, kattintsunk a "Save changes" gombra az ablak alján. Ezzel mentjük el az összes módosítást. Ha azonban mégsem szeretnénk módosítani semmit, akkor kattintsunk a "Cancel" gombra az ablak alján. Ezáltal bezárjuk az ablakot, és nem lesznek mentve semmilyen változtatások.



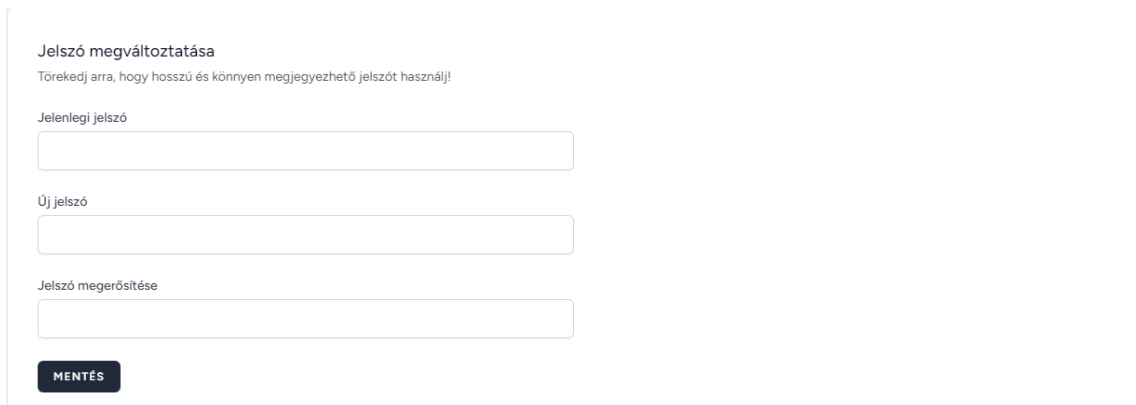
16. ábra Chat stílus módosítás

2.7. Saját profil módosítása

A főoldalon a menüsáv jobb oldalán, a nevünk alatt két opció található. Ha az első lehetőségre, a "Profil módosítás" opcióra kattintunk, akkor megnyílik egy új oldal, ahol lehetőségünk van módosítani a profilunkat. Itt minden adatot módosíthatunk, beleértve a nevünket, email címünket, valamint a jelszót is. A jelszó módosításához azonban szükség van a jelenlegi jelszavunk megadására a biztonság érdekében.

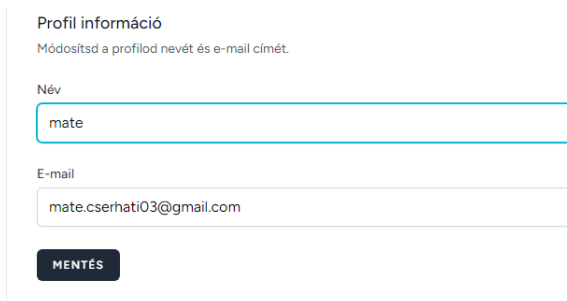
A második lehetőség a "Felhasználó törlése". Mielőtt azonban véglegesen törölnénk a fiókot, először meg kell adnunk a jelszavunkat egy felugró ablakban. Csak a jelszó megadásával tudjuk véglegesen törölni a fiókunkat. Amikor beírtuk a jelszót, kattintsunk a "Felhasználó törlése" gombra a fiók végleges törléséhez. Fontos

megjegyezni, hogy ez a folyamat visszavonhatatlan, és minden adatunkat véglegesen eltávolítja az oldalról.



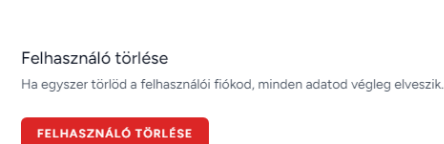
The screenshot shows a web form titled "Jelszó megváltoztatása" (Change password). Below the title is a subtitle: "Törekedj arra, hogy hosszú és könnyen megjegyezhető jelszót használj!" (Try to use a long and easy-to-remember password!). There are three input fields: "Jelenlegi jelszó" (Current password), "Új jelszó" (New password), and "Jelszó megerősítése" (Confirm password). At the bottom of the form is a dark button labeled "MENTÉS" (Save).

17. ábra Jelszó újítás



The screenshot shows a web form titled "Profil információ" (Profile information). Below the title is a subtitle: "Módosítsd a profilod nevét és e-mail címét." (Edit your profile name and email address). There are two input fields: "Név" (Name) with the value "mate" and "E-mail" with the value "mate.cserhati03@gmail.com". At the bottom of the form is a dark button labeled "MENTÉS" (Save).

18. ábra Név és email módosítása



The screenshot shows a web form titled "Felhasználó törlése" (Delete user). Below the title is a subtitle: "Ha egyszer törölsz a felhasználói fiókod, minden adatod végleg elveszik." (If you ever delete your user account, all your data will be permanently lost). At the bottom of the form is a red button labeled "FELHASZNÁLÓ TÖRLÉSE" (Delete user).

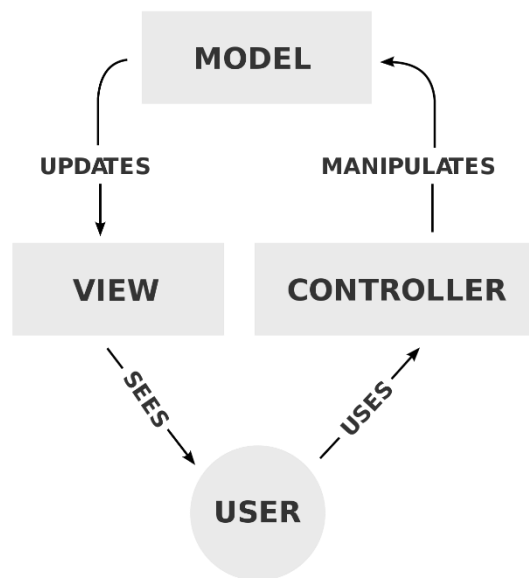
19. ábra Felhasználó törlése

2.8. Kijelentkezés

Ha a nevünkre kattintva a kijelentkezés lehetőséget választjuk, akkor kijelentkezünk az oldalról, és nem leszünk többé bejelentkezve. Ez azt jelenti, hogy átmenetileg elveszítjük az oldalon elérhető funkciókhoz való hozzáférést. Azonban bármikor újra be tudunk jelentkezni az oldalra, és visszanyerjük a teljes hozzáférést az összes funkcióhoz és szolgáltatáshoz. Ez lehetővé teszi számunkra, hogy kényelmesen elérjük az oldal tartalmát és funkcióit újra, amikor úgy döntünk, hogy újra belépünk.

3. Fejlesztői dokumentáció

Januárban tanulmányaink során megismerkedtünk a Laravel keretrendszerrel, egyből nagyon megtetszett számomra, így a csapattársammal együtt úgy döntöttünk, hogy eme keretrendszer segítségével fogjuk elkészíteni a vizsgamunkánkat. A dokumentáció írásának pillanatában a legutóbbi verzió a 11-es. A projektünkben a 10-es verziót használjuk, mert mikor elkezdtük, az volt a legfrissebb. Nem szerettünk volna áttérni a 11-es verzióra, hisz nem biztos, hogy a kód minden komponense helyesen működött volna az újítások miatt. A keretrendszer az MVC architektúrát használja.



20. ábra MVC architektúra

3.1. Laravel

A Laravel egy nyílt forráskódú PHP keretrendszer webes alkalmazások fejlesztéséhez. Taylor Otwell fejlesztette ki 2011-ben, miután a CodeIgniter keretrendszerrel szerzett tapasztalatai alapján úgy érezte, hogy szükség van egy modernebb és rugalmasabb megoldásra.

Az első Laravel verzió (Laravel 1) még alapvető funkciókkal rendelkezett, de gyorsan népszerűvé vált a fejlesztők körében. A keretrendszer igazi áttörése a Laravel 4-gyel jött el, amely számos új funkciót és fejlesztést hozott, mint például az Eloquent ORM-et, a Blade sablonrendszert és a Task Scheduler-t. A Laravel 4 jelentősen javította

a keretrendszer teljesítményét és hatékonyságát, és tovább növelte a Laravel népszerűségét a fejlesztői közösségben.

Azóta a Laravelnek számos újabb verziója jelent meg, amelyek további funkciókkal és fejlesztésekkel bővítették a keretrendszert. A Laravel 5 bevezetett egy új routing rendszert, a Laravel 6 pedig bevezette a Laravel Mix-et, egy Webpack integrációt a Javascript-eszközök egyszerűbb kezeléséhez. A Laravel 7 a Blade sablonrendszer továbbfejlesztéseit hozta, a Laravel 8 pedig bevezette a Laravel Scout-ot, egy Elasticsearch integrációt a teljes szöveges kereséshez.

A Laravel a mai napig folyamatosan fejlődik és frissül, hogy megfeleljen az egyre növekvő fejlesztői igényeknek. A keretrendszer mögött álló csapat elkötelezett amellett, hogy a Laravelt a lehető legjobb PHP keretrendszerré tegyék.

3.1.1. Főbb jellemzők

Teljes csomag: A Laravel mindent biztosít, amire egy modern webes alkalmazáshoz szükség van, beleértve a routingot, a lekérdezéseket, a hitelesítést, a jogosultságkezelést, a sablonkezelést és a beépített tesztelést.

Moduláris: A Laravel moduláris felépítésű, így könnyen bővíthető és testreszabható. Számos hivatalos és harmadik féltől származó modul áll rendelkezésre, amellyel bővíthetjük az alkalmazásunk funkcionalitását.

Könnyen tanulható: A Laravel dokumentációja kiterjedt és könnyen követhető, a beépített Artisan parancssori eszköz pedig gyorsan segít elvégezni a repetitív feladatokat.

Aktív közösség: A Laravelnek hatalmas és aktív fejlesztői közössége van, akik folyamatosan bővítik a keretrendszert és segítenek a fejlesztőknek a problémák megoldásában.

3.1.2. Előnyök a használatában

Gyors fejlesztési sebesség: A Laravel beépített funkciói és moduljai gyorsan segítenek elindítani a projektet, és minimalizálják a fejlesztési időt.

Könnyű karbantarthatóság: A Laravel moduláris felépítése és kódolási stílusa megkönnyíti a kód karbantartását és bővítését.

Biztonságos: A Laravel beépített biztonsági funkciókkal rendelkezik, amelyek megvédik az alkalmazást a támadásoktól.

Skálázható: A Laravel könnyen skálázható, így alkalmas kis és nagy projektekhez egyaránt.

3.2. Weboldal elérése

A weboldalt a <https://eltuntallatkereso.hu> oldalon tudjuk elérni. Ahhoz, hogy az oldalt bárhonnán elérjük, szükségünk volt egy webszerverre. Az Apache webszervert használjuk, mert egyszerű és gyorsan telepíthető. Ezt a webszervert egy virtuális szerveren futtatjuk, amit a <https://www.digitalocean.com/> hosztingról bérelünk és Ubuntu Linux disztribúció fut rajta. Jelenleg megosztott processzor-erőforráson fut a virtuális szerver, mi 2 virtuális processzort használunk mellé 4 GB rendszermemóriával és 25 GB tárhellyel. Túl van méretezve a biztonság kedvéért, hogy ha a későbbiekben több látogató érkezik az oldalra, akkor könnyen kezelje azt a rendszer és ne omoljon össze. A domainünket a <https://www.rackhost.hu/> oldalról béreljük saját költségvetésből, ahogyan a virtuális szervert is. A HTTPS protokoll biztosítása érdekében a CloudFlare proxy-ját használjuk, teljesen bérmentve kapunk hozzáférést ehhez és még minimális védelmet is nyújt számunkra a cég. Ezt a <https://www.cloudflare.com/> weboldalon tehetjük meg, regisztráció után, regisztrálnunk kell a domainünket, majd ezután az oldal ellenőrzi, hogy a megfelelő névszervereket használjuk-e, ha nem, akkor felszólít minket, hogy a titkosítás érdekében váltsunk az ő névszervereikre. Ezt könnyen módosítani tudjuk ott, ahonnan a domainünket béreljük, a mi esetünkben a rackhost oldalán.

3.3. Projekt telepítése és a webalkalmazás elindítása

Ahhoz, hogy telepíteni tudjuk a webalkalmazásunkat szükségünk lesz Git-re, PHP-ra, Composer-re, Node.js-re, az npm package manager-re és egy MySQL szerverre.

Ha mindent telepítettünk, akkor először is klónozzuk a git repository-t a Github oldaláról.

Ehhez nincs más dolgunk, mint a következő parancs futtatása cmd-ben, gith bash-ben vagy bármilyen terminálban, ami képes kezelni a parancsokat.

```
git clone https://github.com/mtcsrht/lostanimal-project.git
```

Ekkor az éppen aktív mappába letöltésre kerül a git repository, benne minden szükséges fájlal.

Következőként telepíteni kell az összes Composer modul-t, ezt a következő paranccsal tudjuk megtenni. Fontos hogy ezt a parancsot ott futtasuk, ahol a local repository található, tehát a fájlok a webalkalmazáshoz.

```
composer install
```

Ezután telepítenünk kell az összes Node modult ezt a következőképpen tehetjük meg.

```
npm install
```

Ha ezzel elkészültünk, létrehozzuk a környezeti változókat tartalmazó fájlt.

```
cp .env.example .env
```

Itt a 3.4-es alfejezet szerint megadjuk a megfelelő adatokat az újonnan létrehozott .env fájlban.

Migráljuk az adatbázist. Ehhez a projekt főkönyvtárában kell lennünk.

```
php artisan migrate
```

Generálunk egy kulcsot a webalkalmazáshoz, ami a titkosítás miatt kell.

```
php artisan key:generate
```

Majd pedig 2 lehetőségünk van. Lokális szerveren érdemes egy fejlesztői szervert futtatni, viszont távoli szerveren pedig érdemes az alkalmazást buildelni.

Fejlesztői szerver

```
npm run dev
```

Webalkalmazás buildelése

```
npm run build
```

3.4. Környezeti változók

A keretrendszer alapvető beállításait egy környezeti fájlból olvassa be, ami nem más, mint egy rejtett .env fájl.

Ezen beállítások a webalkalmazás különböző paraméterei, például a neve, hogy milyen környezetben fut, helyi szerveren (local), vagy futó (production) szerveren, egy kulcs, amit egyedileg kell legenerálni minden indításkor, hogy írjon-e ki debug üzeneteket, és az url, amit fontos mindig jól beállítani.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_URL=http://localhost
```

A következőben az adatbázishoz való csatlakozást tudjuk beállítani, először is, hogy milyen adatbázis csatlakozó driver-t használunk, a szerver címe, a port, ami a MySQL esetében alapvetően 3306, az adatbázis neve, illetve az adatbázis bejelentkező adatai.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Az email szerver megfelelő működéséhez be kell állítanunk, hogy milyen email szervert használunk, pl. SMTP (Simple Mail Transfer Protocol). Az email szerver címe, portja, a bejelentkezési adatok, a titkosítás, hogy milyen email címről kerüljön küldésre, illetve ki legyen a címző, ami alapvetően a webalkalmazás nevére van állítva

```
MAIL_MAILER=smtp
MAIL_HOST=mailpit
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="${APP_NAME}"
```

Ezen kívül a chat működéséhez szükségünk van a Pusher API-hoz való csatlakozáshoz szükséges adatok beállítására, amit a <https://pusher.com/> oldalon

regisztráció után kaphatunk meg, ha létrehozunk egy app-ot az oldalon a webalkalmazásunknak. Ide csak be kell másolnunk az adatokat az oldalról.

```
PUSHER_APP_ID=  
PUSHER_APP_KEY=  
PUSHER_APP_SECRET=  
PUSHER_HOST=  
PUSHER_PORT=443  
PUSHER_SCHEME=https  
PUSHER_APP_CLUSTER=mt1
```

3.5. Útvonalak (Routing)

A weboldalak routingja alapvetően az a folyamat, amely meghatározza, hogy a felhasználói kérések hogyan irányítódnak át a megfelelő kontrollerekre vagy funkciókra egy webalkalmazásban. A routing rendszere tehát összeköti a webes címeket (URL-eket) a mögöttük lévő tartalommal vagy logikával. Laravelben ez a folyamat kiemelkedően rugalmas és könnyen kezelhető, lehetővé téve a fejlesztők számára, hogy egyszerűen irányítsák az alkalmazásuk forgalmát és szabályozzák az útválasztást.

Laravelben a routing definíciókat általában a routes mappában található fájlok tartalmazzák. Ez a mappa több különböző route fájlt is tartalmazhat, mint például a web.php a webes kérésekhez és az api.php az API kérésekhez. Ezen fájlokban definiálhatóak az útvonalak és a hozzájuk tartozó logika.

A landing page-hez tartozó útvonal egy GET kérést definiál, amikor beírjuk a böngésző címsorába, hogy <http://eltuntallatkereso.hu> akkor a böngésző a webszervernek küld egy GET kérést a böngésző pedig megjeleníti a welcome.blade.php oldalt.

```
Route::get('/', function () {  
    return view('welcome');  
});
```

3.5.1. Kontrollerek

A bonyolultabb logikák esetében a Laravel lehetőséget biztosít arra, hogy a route-kat kontrollerekhez kössük, amelyek osztályok, és szervezettebben kezelhetővé teszik a kódunkat.

Amikor szeretnénk megnézni a posztjainkat, a főoldalról átvezet minket egy gomb a <https://eltuntallatkereso.hu/posts> oldalra, ekkor a böngésző egy GET kérést küld a webszervernek, ami az előre meghatározott útvonalból tudja, hogy ekkor a PostContoller show metódusát kell meghívnia, amit a Kontrollerek alfejezetnél fejtek ki. A posts.show egy egyszerűsített neve az útvonalnak, a kódban való könnyebb meghívás és olvashatóság érdekében.

```
<a href="{route('posts.show')}}" class="..." /a>  
Route::get('/posts', [PostController::class, 'show'])->name('posts.show');
```

Van lehetőségünk paramétert is átadni a böngésző sorában, amikor a saját posztok oldalán rámegyünk, hogy biztosan szeretnénk törölni az állatot, egy form DELETE kérést küld a <https://eltuntallatkereso.hu/myposts/> oldalnak, paraméterként átadva az állat objektumot, ezt a PostController destroy metódusa veszi fel, majd pedig kezeli a később leírt módon.

```
Route::delete('/myposts/{animal}', [PostController::class, 'destroy'])->name('myposts.destroy');
```

Itt a poszt létrehozásakor a form elküld egy POST kérést a <https://eltuntallatkereso.hu/createpost> útvonalnak, amit kezel a webszerver, ezután a PostController store metódusa kezeli a kérést a később leírt módon.

```
Route::post('/createpost', [PostController::class, 'store'])->name('uploadpost');
```

3.5.2. Middleware

A Laravel routing rendszerébe integrálva vannak a middleware-ek, amelyek lehetővé teszik, hogy kérések feldolgozása előtt vagy után közbelépjenek. Ezeket a köztes rétegeket használhatjuk például a hitelesítéshez, naplózáshoz vagy kérés előfeldolgozás céljából.

Amikor az admin oldalakat szeretnénk elérni akkor az admin middleware előtte ellenőrzi, hogy a felhasználó be van-e jelentkezve és admin-e.

```
Route::middleware('admin')->group(function () {  
    //adminhoz tartozó útvonalak  
});
```

A Laravel routing rendszere rendkívül kifinomult, lehetővé téve a fejlesztők számára, hogy rugalmasan és hatékonyan kezeljék az alkalmazások útvonalait. A jól definiált route-ok, a kontrollerek használata, a middleware integráció, valamint az

útvonalak csoportosításának képessége mind hozzájárul egy tiszta, karbantartható és jól szervezett kódalap kialakításához.

3.6. Middleware

A middleware-k olyan fontos elemei a Laravelnek, amelyek segítségével könnyen kezelhetjük a HTTP kéréseket a kliens és a szerver között.

A middleware egy olyan köztes réteg, amelynek célja a HTTP kérések feldolgozása és manipulálása, a keretrendszerben. A segítségével például képesek vagyunk ellenőrizni, hogy a felhasználó be van-e jelentkezve, különben nem éri el a kért oldalakat, amelyeket a middleware ellenőrzés alá von. Ezeket a middleware-eket a routingnál határozzuk meg, hogy mely útvonalakat vonja ellenőrzés alá. Erre példát is találhatunk az Útvonalak fejezet Middleware alfejezet részében, ahol az admin oldalak elérése során kerül ellenőrzésre, hogy a felhasználó admin-e.

Ez úgy működik, hogy amikor a felhasználó a böngésző sorába beírja, hogy <https://eltuntallatkereso.hu/admin> akkor ez a kérés mielőtt továbbjut az útvonalhoz, átmegy az admin middleware-n. Ez ellenőrzi először, hogy a user be van-e jelentkezve, majd, hogy az admin tábla tartalmazza-e a bejelentkezett user email címét, ha nem akkor egy 403-as HTTP választ kapunk vissza, aminek jelentése, hogy nincs jogunk hozzáférni az adott oldalhoz. Ha minden egyezik, akkor pedig továbbadja az adatokat az útvonalkezelő osztálynak, ami kezeli tovább a kérést.

```
public function handle(Request $request, Closure $next)
{
    if (!auth()->check() || !Admin::where('email', auth()->user()->email)->exists()) {
        return abort(403, 'Hozzáférés megtagadva!');
    }
    return $next($request);
}
```

3.7. Modellek (Model)

A Laravel modellek adatbázis táblák objektumorientált reprezentációi. Segítségükkel egyszerűsíthetjük és karbantarthatóbbá tehetjük kódunkat. A modellek objektumokban tárolják az adatokat, így könnyebben kezelhetők és manipulálhatók. A modellek elrejtik az adatbázis-specifikus kódot, így a kódunk hordozhatóbbá válik. Könnyebben tudjuk validálni a bemeneti adatokat.

Egy új modellt a **php artisan make:model** paranccsal hozhatunk létre. A parancsnak meg kell adnunk a modell nevét. A parancs létrehoz egy új fájlt az **app/Models** mappában. Ez a fájl fogja tartalmazni a modell osztályát. Itt definiálhatjuk a modell mezőit, kapcsolatait és validációs szabályait.

A modell mezőit **\$fillable** és **\$guarded** tulajdonságok segítségével definiálhatjuk. A **\$fillable** tulajdonság megadja azokat a mezőket, amelyeket szeretnénk kitölteni, míg a **\$guarded** azokat, amelyeket nem szeretnénk kitölteni, ide általában generált szöveg, szám stb. jön, például jelszó.

A modellek közötti kapcsolatokat a **belongsToMany**, **belongsTo**, **hasOne** és **hasMany** metódusok segítségével definiálhatjuk.

A modellek validációs szabályait a **\$rules** tulajdonság segítségével definiálhatjuk. A **\$rules** tulajdonság egy tömb, amely a mezőnevekhez rendelt validációs szabályokat tartalmazza.

A lekérdezéseket az Eloquent ORM segítségével egyszerűen végrehajthatjuk. Az Eloquent ORM egy objektumorientált interfész az adatbázis lekérdezéséhez. Alap lekérdezések például:

Itt az összes user model lekérdezésre kerül, majd eltárolásra a \$users tömbben.

```
$users = User::all();
```

Itt egy állat uuid-je alapján keresünk egy állatot.

```
$animal = Animal::where('uuid', $animal->uuid)->first();
```

A modelleket lehetőségünk van módosítani vagy törölni is. A módosítást például az állatok szerkesztése után alkalmazzuk a **save()** metódus segítségével. Létezik még ezen kívül az **update()** metódus is. Ha törölni szeretnénk egy állatot, akkor a **delete()** metódus lesz segítségünkre.

```
$animal->save();  
$animal->delete();
```

3.8. CSRF Token

A CSRF (Cross-Site Request Forgery) egy egyedi titkos token amelyet a Laravel generál minden egyes aktív user session-ben. Ez a token arra szolgál, hogy az oldal

ellenőrizni tudja, hogy valóban a user küld kéréseket az alkalmazásnak és megakadályozzon minden kéretlen kérést, amelyet nem a bejelentkezett user küld.

Annak érdekében, hogy használni tudjuk a keretrendszer CSRF védelmét, egy hidden input mezőt kell a HTML form-ban generálnunk, amelynek az értéke egy hosszú karaktersorozat lesz, amely a tokenünk. Ehhez csupán annyit kell beírunk a kódba, hogy @csrf, ez le fogja nekünk automatikusan generálni a rejtett mezőt.

A CSRF token validálása a keretrendszer által történik, ha hiányzik ez a token, vagy nem megfelelő, akkor a keretrendszer elutasítja a kérést egy 419-es http válaszkóddal. Ennek köszönhetően tudjuk ilyesfajta támadásoktól megvédeni az oldalunk.

3.9. Kontrollerek (Controller)

A kontrollerek olyan PHP osztályok, amelyek a webes kérések feldolgozásáért felelősek. Segítségükkel strukturáltabbá és jobban karbantarthatóvá tehetjük a Laravel webalkalmazásunkat. A kontrollerekhez útvonalakat rendelhetünk, ahogy az Útvonalak fejezetben is olvashatjuk. Ezek a kontrollerek metódusokat definiálnak, amelyek a HTTP kérések típusától függően (GET, POST, PUT, DELETE) hajtódnak végre. Ezek a kontrollerek különböző válaszokat adhatnak vissza a felhasználóknak pl. nézetet (view) vagy JSON-t. A keretrendszer rendelkezik saját kontrollerekkel, mint például az autentikációért felelős kontrollerek, amiket mi a projekt szempontjából kicsit átalakítottunk, például a regisztrációért felelős kontrollert.

3.9.1. Felhasználóhoz tartozó controller

Amit módosítanunk kellett rajta, hogy új mezőket vittünk fel a nézeten, ezáltal több adatot kellett validálni a felhasználó létrehozása előtt. Minden olyan metódust, ami új példányt hoz létre “store”-nak nevezünk el. A Laravelben egy beépített backend validáció van, ahol kulcs-érték párokat adunk meg az első paraméterében a validate metódusnak. A kulcs a request-ben szereplő paraméter neve, az érték pedig egy tömb a validációs szabályokkal, ami meghatározhatja, hogy szükséges-e kitölteni egy mezőt, a mező típusa, például karakterlánc, vagy szám, illetve, hogy mekkora legyen a maximum karakterszám és/vagy minimum karakterszám. Ezek után a második paraméterben definiálhatunk fordításokat, de ez opcionális.

```
public function store(Request $request): RedirectResponse
{
```

```

$request->validate([
    'name' => ['required', 'string', 'max:255'],
    'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:'.User::class],
    'password' => ['required', 'confirmed', Rules\Password::defaults()],
    'firstname' => ['required', 'string', 'max:255'],
    'lastname' => ['required', 'string', 'max:255'],
    'phonenumber' => ['required', 'numeric', 'min:11'],
    'postalCode' => ['required', 'string', 'min:4', 'max:4'],
], [
    'name.required' => 'A név megadása kötelező.',
    'name.string' => 'A név csak szöveg lehet.',

```

A Hash osztály meghívásának segítségével képesek vagyunk létrehozni az új jelszót. A jelszó titkosítására bcrypt-et használunk. Ez egy elég megbízható közepes erőforrás-igényű titkosítási eljárás. A create metódus pedig létrehozza az adatbázisban az új felhasználót.

```

$user = User::create([
    'name' => $request->name,
    'email' => $request->email,
    'password' => Hash::make($request->password),
    'firstname' => $request->firstname,
    'lastname' => $request->lastname,
    'phonenumber' => $request->phonenumber,
    'IRSZ_Id' => $request->postalCode,
]);

```

Mikor egy GET kérést kapunk - például a regisztrációhoz -, akkor az útvonalkezelő meghív egy metódust (létrehozásnál a create nevet adjuk ennek a metódusnak, szerkesztésnél: edit) ami elvezeti a felhasználót a megfelelő nézetre.

```

public function create(): View
{
    return view('auth.register');
}

```

3.9.2. Kontroller létrehozása

Létrehozhatunk saját kontrollereket, erre a legegyszerűbb mód, ha belépünk egy parancssorral a főkönyvtárba, majd beírjuk a következő parancsot:

```
php artisan make:controller PeldaController
```

A PeldaController helyett a saját kontrollerünk nevét adhatjuk meg.

Ahhoz, hogy műveleteket hajtsunk végre az állatokkal, amiket posztként kezelünk, minden esetben a PostController osztályt használjuk.

3.9.3. Állathoz tartozó kontroller

A user saját állatait a myposts nézetben jelenítjük meg, az útvonal-kezelő hívja meg GET kérés esetén a self metódust, ahol a bejelentkezett user id-ja alapján kérjük le az állatait, majd visszatérünk a myposts nézettel, és átadjuk neki a felhasználóhoz tartozó állatokat.

```
public function self(Request $request) : View
{
    $animals = Animal::where("userId", auth()->user()->id)->get();
    return view("myposts", compact("animals"));
}
```

Amikor egy állat adatlapjára vagyunk kíváncsiak, akkor az “about-animal” nézet jelenik meg, amelynek átadjuk magát az állatot, az user-t, akihez tartozik, illetve az állat színét. Ezt úgy tudjuk megtenni, hogy lekérdezzük az állathoz tartozó user id alapján, hogy ki a pontos user, illetve a szín id alapján, hogy milyen színű az állat. Mivel a paraméter már tartalmazza az állatot, így azt csak egyszerűen továbbadjuk a nézetnek.

```
public function index(Animal $animal): View
{
    $user = User::where('id', $animal->userId)->first();
    $color = Color::where('id', $animal->colorId)->first();
    return view("about-animal", compact("animal", "user", "color"));
}
```

Amikor az összes állatot szeretnénk látni, akkor meghívja az útvonalkezelő a show metódust, aminek egyetlenegy paramétere van a “request”, ami nem más, mint a kérés összes eleme. Amikor szűrünk, a form-ban átadott colorId is ide kerül, így a metódus kezelni tudja ezt az átadott értéket. A szűrés nagyon egyszerűen van megoldva, minden egyes szűrőfeltételnél van egy when metódus, ami elsősorban ellenőrzi, hogy a request tartalmaz-e olyan szűrési feltételt (pl. color_id, a HTML form-ban a name attribútum határozza meg a kérésben szereplő nevet), ha tartalmaz, akkor lefut a callback metódus, és a \$query változóba belekerül az összes olyan állat, amelyik az adott feltételnek megfelel. Ezután ezt továbbadjuk a következő when metódusnak egészen addig, míg az utolsó megadott szűréshez ér. A rendezés során, amikor átadjuk a kérésben a sort_by feltételt, akkor 4 lehetőségünk van: vagy feltöltés dátuma szerint növekvő, vagy csökkenő sorrendbe, vagy név szerint növekvő, vagy csökkenő sorrendbe tehetjük. Egy listába szétbontjuk az explode metódussal a \$sort_by értékét. Az első az oszlop neve a táblában (pl. name) a második pedig a rendezés iránya (pl. “asc” azaz növekvő). Majd visszatér a

callback funkció a sorbarendezt állatokkal, ezután átadjuk az összes leszűrt állatot a paginate függvénynek, ami egy paramétert vesz fel, hogy hány állat jelenjen meg oldalanként. Erre a paginate függvényre azért van szükségünk, hogy a beépített lapozó megfelelően működjön.

```
public function show(Request $request): View
{
    $animals = Animal::when($request->color_id, function ($query, $color_id) {
        return $query->where('colorId', $color_id);
    })->when($request->gender, function ($query, $gender) {
        return $query->where('gender', $gender);
    })->when($request->chipNumber, function ($query, $chipNumber) {
        if($chipNumber == "has"){
            return $query->whereNotNull('chipNumber');
        }else{
            return $query->whereNull('chipNumber');
        }
    })->when($request->query('sort_by'), function($query, $sort_by){
        if($sort_by){
            list($column, $direction) = explode('-', $sort_by);
            return $query->orderBy($column, $direction);
        }else{
            return $query;
        }
    })->paginate(6);

    $users = User::all();
    $colors = Color::all();
    return view("posts", compact("animals", "users", "colors"));
}
```

Mikor állatot szeretnénk feltölteni, először megjelenítjük a createpost nézetet, aminek át kell adnunk a színeket, hogy a legördülő menüben megfelelően meg tudjuk jeleníteni.

```
public function create() : View
{
    $colors = Color::all();
    return view("createpost", compact('colors'));
}
```

Amikor feltöltjük az állatot, a createpost nézetről jön egy POST kérés, amit az útvonalkezelő kezel, majd meghívja a kontroller store metódusát. A store metódus ezután, ahogy a felhasználó regisztrálásánál validálja a megfelelő adatokat a megfelelő validációs szabályokkal, ha ez nem felel meg akkor a hibákat visszadobja az oldalnak, ahonnan

érkezett a kérés, az pedig megjeleníti őket. Ahhoz, hogy képet is tudjunk feltölteni, a beépített fájlkezelőt használjuk, és végül az adatbázisba az állat képének elérési útját adjuk meg.

```
$request->validate([
    'name' => ['required', 'string'],
    'chip' => ['nullable', 'string', 'max:16'],
    'gender' => ['required'],
    'color' => ['required'],
    'description' => ['required', 'string', 'max:1000'],
    'picture' => [
        'required',
        'image',
        'mimes:jpg,png,jpeg,svg',
        'max:2048'
    ],
],
[
    'name.required' => 'A név mezőt kötelező kitölteni',
    'name.string' => 'A név mezőnek karakterláncnak kell lennie',
    többi hibaüzenet...
]);

$imagePath = $request->file('picture')->store(
    'animal-pictures',
    'public'
);
```

A példány létrehozása:

```
Animal::create([
    'uuid' => Uuid::uuid4()->toString(),
    'userId' => $request->user()->id,
    'name' => $request->name,
    'chipNumber' => $request->chip,
    'gender' => $request->gender,
    'colorId' => $request->color,
    'description' => $request->description,
    'image' => $imagePath,
]);
```

Majd a végén visszatérés a nézettel, illetve egy státuszüzenettel.

```
return Redirect::route('myposts.index')->with('status','animal-uploaded');
```

Amikor szerkesztünk egy állatot ugyanez történik, 2 dolog módosul, a paraméterben átadjuk az állatot, amit változtatni szeretnénk, az validálásra kerül ugyan azon szabályok szerint, mint a feltöltésnél, kivéve, hogy a kép nullable lesz, azaz nem

muszáj feltölteni, mert már alaphól fel van töltve egy. Ha validáltuk, az eredeti állat példánynak módosítjuk a tulajdonságait.

```
$animal->name = $request->name;
$animal->chipNumber = $request->chip;
$animal->gender = $request->gender;
$animal->colorId = $request->color;
$animal->description = $request->description;
```

Majd a képet ellenőrizzük, hogy a kérésben szerepel-e fájl, ha igen, akkor eltároljuk ezt a képet és az új kép elérési útját is módosítjuk a példányban.

```
if ($request->hasFile('picture')) {
    $imagePath = $request->file('picture')->store(
        'animal-pictures',
        'public'
    );
    $animal->image = $imagePath;
}
```

Ha ez is megvan, akkor pedig elmentjük az állatot és visszatérünk a saját állataink oldalra, egy státuszüzenettel.

```
$animal->save();
return Redirect::route('myposts.index')->with('status','animal-updated');
```

Amikor állatot törölünk, a DELETE kérést kezeli az útvonalkezelő majd, meghívja a destroy metódust, aminek paraméterként átadásra kerül az állat. Ellenőrizzük először is, hogy létezik-e az állat képe a tárhelyünkben (hibakezelés), ha létezik, akkor az unlink metódussal töröljük azt, ezután az állat példány delete metódusát meghívjuk és az állat törlésre kerül az adatbázisból. Ezután visszairányít minket a saját posztjaink oldalra egy státuszüzenettel.

```
public function destroy(Animal $animal)
{
    // Az állat képének törlése, ha egyáltalán létezik az a kép(hibakezelés).
    $imagePath = $animal->image;
    if(file_exists(public_path('storage/'.$imagePath)))
    {
        unlink(public_path('storage/'.$imagePath));
    }
    // Az állat törlése.
    $animal->delete();
    // A myposts.blade.php oldalra irányítás. Egy animal-deleted státuszüzenettel.
    return Redirect::route('myposts.index')->with('status','animal-deleted');
}
```

3.9.4. Admin kontroller

Az admin kontroller az admin oldal létrehozásához volt szükséges. Először is a megjelenésért a show metódus felel, ami átadja az összes várost és user-t az admin nézetnek.

```
public function show()
{
    // Az összes felhasználót és várost lekéri az adatbázisból.
    $users = User::all();
    $cities = City::all();
    // Az admin.blade.php oldalon jeleníti meg a felhasználókat és a városokat.
    return view('admin', compact('users', 'cities'));
}
```

A törlés pont mint az állatoknál a destroy metódussal történik, csak itt nem adjuk át paraméterbe az user-t, hanem az id-t adjuk át kérésben, ami alapján lekérdezzük a usert a when metódussal, majd a visszaadott egyedet töröljük és visszatérünk az admin oldalra egy státusz üzenettel.

```
public function destroy(Request $request)
{
    // A felhasználót törli az adatbázisból.
    User::where('id', $request->user)->delete();
    // Az admin oldalra visszatér egy státuszűzenettel.
    return redirect()->route('admin.show')->with('status', 'user-deleted');
}
```

A felhasználó admin által való szerkesztése egyszerű, mert csak a telefonszámot, lakhelyet, vezetéknévet, keresztnévet tudjuk módosítani, ezeket validáljuk. Ezeket a kérésből szedjük ki.

```
$request->validate([
    'firstname' => 'required',
    'lastname' => 'required',
    'phone' => 'required|min:11',
    'irsz' => 'required',
],
[
    'firstname.required' => 'Keresztnév megadása kötelező!',
    'lastname.required' => 'Vezetéknév megadása kötelező!',
    'többi hibaüzenet....'
])
```

Ha ezzel megvagyunk, csak akkor kérjük le az user-t, ha minden validálva lett és működik. A lekért példánynak módosítjuk a tulajdonságait, majd elmentjük és visszatérünk egy státusz üzenettel.

```

$user = User::where('id', $request->user)->first();
// A felhasználó adatait frissíti.
$user->firstname = $request->firstname;
$user->lastname = $request->lastname;
$user->phonenumber = $request->phone;
$user->irsz_id = $request->irsz;
$user->save();
return redirect()->route('admin.show')->with('status', 'user-updated');

```

3.9.5. Városok kontroller

A városok kontrollert csak az API miatt kellett elkészíteni. Amikor szeretnénk lekérdezni az összes várost, az index metódust hívja meg az API útvonalkezelő. Ez a metódus lekérdezi az irányítószámot, majd átnevezi postal_code-ra, a város nevét city_name-re és a megyét county-ra, hogy amikor kérést küldünk az apinak olvashatóbb legyen, ezután a két táblát össze kell kötnünk a város tábla County_Id idegen kulcsa és a megye tábla id elsődleges kulcsa segítségével.

```

public function index()
{
    return City::select(['irsz_id as postal_code', 'counties.name as county', 'cities.name as city'])->join('counties', 'cities.county_id', '=', 'counties.id')->get();
}

```

Amikor egy város név részletére keresünk rá, akkor a LIKE függvényt használjuk a WHERE után.

```

public function show($name)
{
    return City::where('cities.name', 'like', $name . '%')->join('counties', 'cities.county_id', '=', 'counties.id')->select(['irsz_id as postal_code', 'counties.name as county', 'cities.name as city'])->get();
}

```

3.10. Nézetek (View)

A megjelenéshez a Blade sablonmotorját használjuk, amely könnyű, hatékony, biztonságos sablonmotor. Könnyen tudunk HTML kódban PHP kódot megjeleníteni, akár közvetlenül is. Minden nézetet a **resources/views** mappában tárolunk és a kiterjesztése a fájloknak a .blade.php.

A nézetek azt a célt szolgálják, hogy a felhasználó láthassa magát az oldal tartalmát. A nézetek layout-ra épülnek. 3 Layoutot különböztetünk meg a projektünkben. A public layout, arra szolgál, hogy a publikus posztokat jelenítsük meg.

```
<x-public-layout>
<!-- Tartalom -->
</x-public-layout>
```

A guest layout a bejelentkezéshez és a regisztrációhoz kell.

```
<x-guest-layout>
<!-- Tartalom -->
</x-guest-layout>
```

Az app layout pedig minden olyan funkcióhoz kell, ami bejelentkezést igényel.

```
<x-public-layout>
<!-- Tartalom -->
</x-public-layout>
```

Ahol a tartalom komment van, oda írjuk meg a HTML kódunkat, de akár scriptet is írhatunk bele. Az app layout fő fájljában, ahova később az imént leírt kódokat illeszti be a Blade motor, meghatározunk egy header részt, amit előtte egy szelekcióval ellenőrzünk, hogy van-e. A Blade sablonmotorba úgy tudunk adatokat megjeleníteni, hogy {{ }} jelek közé tesszük, ezek megvédik az oldalt az XSS támadásoktól, tehát ha valaki például egy állatnak azt a nevet adja, hogy <script> alert('HACKED') </script> akkor ez nem fog lefutni, ellentétben a PHP echo módszerével.

```
@if (isset($header))
    <header class="bg-white dark:bg-gray-800 shadow">
        <div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">
            {{ $header }}
        </div>
    </header>
@endif
```

A headert a beillesztett fájlban határozzuk meg.

```
<x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">
        {{ __('Új poszt') }}
    </h2>
</x-slot>
```

Minden kód, ami ezen kívül kerül, a \$slot változóba kerül betöltésre egy main tagen belül.

```
<main>
    {{ $slot }}
</main>
```

A TailwindCSS stílus keretrendszert használjuk, így nagyban megkönnyítettük a munkánkat. Így csak az előre elkészített osztályokat kellett felhasználnunk és nem kellett

megírunk a stíluslapot. Az app.css-ben csak importáljuk azokat a tailwind modulokat, amiket szeretnénk használni. Majd a Vite segítségével importálni tudjuk az oldalunkba.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
@vite(['resources/css/app.css', 'resources/js/app.js'])
```

Minden oldalon a hátteret a body-ra töltjük be az előre definiált osztályokkal.

```
<body class="font-sans antialiased bg-mobile-waves-lightmode sm:bg-waves-lightmode
sm:dark:bg-waves-darkmode dark:bg-mobile-waves-darkmode bg-no-repeat bg-cover">
```

Ahhoz, hogy pl. a bg-waves-darkmode megjelenítse az általunk beállított képet, a tailwind configba meg kellett határoznunk egy saját osztályt. Illetve itt határozzuk meg az alap betűtípust is, ami nem más, mint a Figtree, ezt a google font api-jának köszönhetően kapjuk meg.

```
theme: {
  extend: {
    fontFamily: {
      sans: ['Figtree', ...defaultTheme.fontFamily.sans],
    },
    backgroundImage: {
      'lightmode': "url(/resources/images/bg-white.svg)",
    },
  },
}
```

Ahhoz, hogy a betűtípus működjön, előre be kell töltenünk a google fontot, hogy ne legyen csúnya a megjelenés.

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link
href="https://fonts.googleapis.com/css2?family=Figtree:ital,wght@0,300..900;1,300..900&displ
ay=swap" rel="stylesheet">
```

A title-t a környezeti változókat tartalmazó .env fájlból olvassa ki.

```
<title>{{ config('app.name', 'Laravel') }}</title>
```

A biztonság érdekében szükségünk van egy csrf tokenre is, amit a meta tag-ben tudunk meghatározni.

```
<meta name="csrf-token" content="{{ csrf_token() }}">
```

A public és az app layoutoknál külön navigációt használunk, hisz a publicnál nem biztos, hogy be vagyunk jelentkezve, ezáltal más jelenik meg, mint mikor be vagyunk jelentkezve. Az app layoutnál nem ellenőrizzük, hogy be van-e jelentkezve az user.

Az include segítségével tudjuk a navigációt beszúrni.

```
@include('layouts.navigation')
```

3.10.1. Publikus posztok

A publikus posztokban, hogy meg tudjuk jeleníteni az állatokat, a PostControllertől megkaptuk őket, egy PHP tömb típusú változó formájában, ezután ezen végigmegyünk a beépített @forelse ciklussal.

```
@forelse ($animals as $animal)
```

Azért használunk forelse-t, mert ha üres a tömbünk, akkor van egy hamis águnk (igaz a „feltétel”, ha van akár egy eleme is tömbnek, ekkor lefut a ciklus), ahol meg tudjuk jeleníteni, hogy „Nincs megjeleníthető állat”, ugyanez történik akkor is, ha nincs találat a szűrésekre.

A kártyákba az adatokat a fentebb említett módon tudjuk beilleszteni, pl. az állat nevét a {{ \$animal->name }} kódrészlet segítségével.

A paginátor egy beépített metódus segítségével hozható létre, ez a metódus automatikusan létrehozza a léptetőhöz szükséges HTML kódot.

```
{{ $animals->links() }}
```

Ehhez viszont szükséges a 3.9.3-as alfejezetben szükséges show metódusban leírt paginate() metódus hozzáadása a lekérdezéshez.

3.10.2. Poszt feltöltése

A Blade sablonmotor alapvetően létrehoz számunkra komponenseket a resources/views/components mappában, ilyen az x-input-label illetve x-text-input is. Ezek előre meghatározott stílusú, illetve attribútumokkal lekódolt új tag-ek, amiket be tudunk szúrni az oldalunkra, így elősegítve az egységesítést. Képesek hiba esetén a régi értéket újra megjeleníteni, hogy ne kelljen újra begépelnünk az old (*name attribútum*) metódus segítségével. Hibaüzeneteket is meg tudunk jeleníteni, amik a validálás során keletkeznek a kontrollerben az x-error tag segítségével, itt a message attribútumba az \$errors tömbből lekérdezzük az adott mező hibáit.

```
<x-input-label for="name" :value="__('Állat neve')"/>
<x-text-input id="name" class="block mt-1 w-full" type="text" name="name"
:value="old('name')"/>
<x-input-error :messages="$errors->get('name')" class="mt-2"/>
```

Az összes input mezőt egy form-ba helyeztük el, amely submit (elküldés) után az uploadpost útvonalnak küld POST kérést. Ezen kívül meghatározzuk az encoding type-ot, hogy tudjunk képet is feltölteni a multipart/form-data érték segítségével. A következő sorba beillesztjük a csrf token mezőt, aminek szükségességét a 3.8-as alfejezetben fejték ki.

```
<form action="{{ route('uploadpost') }}" method="POST" enctype="multipart/form-data"
class="Tailwind osztályok...">
@csrf
```

A végén pedig szintén egy előre elkészített komponens lesz a submit gombunk, ami az x-primary-button.

3.11. API

Az API, vagy Alkalmazásprogramozási felület, egy olyan szoftverszerkezet, amely lehetővé teszi, hogy különböző alkalmazások kommunikáljanak egymással. A Laravel keretrendszer kiváló eszközökkel szolgál az API-k fejlesztéséhez, többek között beépített funkciókkal az autentikációhoz, a routinghoz, a kivételkezeléshez és a válaszok kezeléséhez.

A mi projektünkben 3 féle végpont van, aminek kérést küldve egy JSON response (választ) kapunk vissza

Minden <https://eltuntallatkereso.hu/api> lekérdezés az api.php-ban meghatározott routeokon (útvonalakon) keresztül történik. Itt meghatározzuk, hogy melyik elérési úton, melyik kontroller melyik metódusa fusson le.

Ezek közül az egyik a <https://eltuntallatkereso.hu/api/cities>. Erre azért volt szükség, mert a regisztráció során innen kérdezi le a frontend JavaScript kódja a városokat. Mikor GET kérést küldünk erre a végpontra, akkor visszatér egy JSON válasszal, amit később fel tudunk dolgozni.

```
Route::get('cities', [CityController::class, 'index']);
```

Ezen kívül lehetőségünk van a város neve alapján is keresni az api-ban. Itt a cityname paraméter átadásra kerül a CityController show metódusának, amit kezel a Város kontrollerben leírt módon.

```
Route::get('cities/{cityname}', [CityController::class, 'show']);
```


Például, ha szeretnénk a “bal” szórészlettel kezdődő városok nevét lekérdezni, akkor egyszerűen csak GET kérést küldünk az apinak: <https://eltuntallatkereso.hu/api/cities/bal> erre a következő lesz a válasz

```
[
  {
    "postal_code": 2660,
    "county": "Nógrád",
    "city": "Balassagyarmat"
  },
  {
    .... többi válasz pl.: minden balatoni város
```

Ezen kívül lehetőségünk van lekérdezni az összes állatot is a <https://eltuntallatkereso.hu/api/animals> végponton.

3.12. Városok és megyék forrása

Mivel szükségünk volt a projekthez olyan táblára, ami tartalmazza a megyéket és egy másikra, ami pedig a városokat, így segítségül kellett hívni az internetes szakmai oldalakat. Meg is találtam ezt az oldalt https://webdraft.eu/orszagok_varosok/ ahol igaz SQL dump is van, de nekem ennél egy különlegesebb szerkezetre volt szükségem. Ezért letöltöttem az excel verziót, készítettem egy egyedi táblát benne, majd pedig C# segítségével írtam egy programot, amely szétválogatta a városokat, illetve megyéket. A megyék mellé id-t rendelt, amelyeket a városok táblában kicserélt a megfelelő helyen, majd 2 csv fájlt hozott létre, amelyeket be tudtam importálni a MySQL szerverre.

3.13. Tesztelés

3.13.1. Frontend

Az oldal frontend tesztelésére a Google által kiadott Lighthouse böngésző bővítményt használtuk. Ez az eszköz arra való, hogy teszteljük az oldal teljesítményét, sebességét, és helyességét. Miután feltelepítjük a bővítményt, egyszerűen csak belépünk arra az oldalra, amit szeretnénk tesztelni, ezek lehetnek aloldalak is. Mi a legfontosabbat teszteltük először, a publikus posztokat, hisz a legtöbb látogató itt fog tevékenykedni. Ha beléptünk a tetszőleges oldalra, a Google Chrome beépített fejlesztői eszköztárát megnyitjuk és a Lighthouse fülre kattintunk, majd rámegyünk az „Analyze Page Load” gombra. El fog indulni az oldal betöltése, közben a bővítmény folyamatosan ellenőrzi az oldalt. Ha kész, egy 5 elemű eredményt fogunk kapni. Mindenre maximum 100 pontot

kaphatunk. A **performance** indikátor azt mutatja meg, hogy milyen gyorsan tölt be az oldalunk, majd részletezi, hogy mikén tudunk javítani, hogy ez gyorsabb legyen. Mi 97 pontot kaptunk, amin javítanunk kellett, így mostmár nem a Google API segítségével kérjük le a betűkészletet, hanem a saját szerverünkről, így nem kell megvárni, hogy a Google API választ küldjön nekünk. A képeink JPG, vagy PNG formátumban is feltölthetőek, viszont ezek nem tartoznak a „Next-gen” formathoz, mert nem jó a tömöríthetőségük. A bővítmény ajánlotta, hogy csökkentsük a felhasználatlan JavaScript méretét, amire azért nincs lehetőségünk, mert minden oldal ugyanazt a JavaScript fájlt használja.

Az **accessibility** indikátor azt jelenti, hogy az oldalunk mennyire egyértelműen használható. Itt 90 pontot kaptunk, a probléma ott volt, hogy a kártyáknál a nevek, amelyek kattintható linkek, nem elég kitűnőek. Ezen azért nem javítunk, mert nem is szeretnénk design szempontjából, hogy kitűnjenek. A másik probléma, amit a jelentésben kaptunk, hogy a kis kutyus ikon a bal felső sarokban nem rendelkezik semmilyen egyértelműsítővel, hogy visszavezet a főoldalra, ez főleg azoknál jelent problémát, akik képernyőolvasót használnak az oldal tartalmának megtekintésére.

A best practices indikátor azt mutatja, hogy az oldal a megfelelő technológiákat és megoldásokat használja-e. Mi erre 100 pontot kaptunk. Ebben a tesztben ellenőrzi például a bővítmény, hogy HTTPS-t használunk, mellőzzük az elavult API-kat és nincsenek a konzolban error-ok, illetve issue-k.

A SEO (Search Engine Optimization) azt ellenőrzi, hogy az oldalunk megfelel-e a keresőmotorok által optimálisnak tartott weboldalak követelményeinek. A mi esetünkben 91 pontot kaptunk. Egyetlen egy hibát kaptunk, hogy nincs meta leírásunk a HTML fájlokban, mivel nem szeretnénk egyelőre, hogy megjelenjen a böngésző keresési eredményei között az oldal, így nem töltöttük ki ezt a mezőt.

3.13.2. Backend

A backend tesztelésre a Laravel tökéletes megoldást biztosít, hisz, ha bármilyen probléma van, azonnal jelzi számunkra a beépített debuggerrel. Manuálisan is meg tudjuk jeleníteni a debug felületet, erre 2 lehetőségünk is van, egy kevésbé részletes és egy részletes leírással rendelkező menüvel. Ahhoz hogy manuálisan előhozzuk a minimális debug felületet, beírjuk a `dd()`; parancsot, aminek paraméterébe átadunk egy változót, amit szeretnénk vizsgálni. Például így teszteltük a validálás előtt a `$request` változókat is,

hogy megfelelő-e minden adat, ami a kérésben érkezett. A `ddd();` parancs a részletesebb debug menüt nyitja meg, itt is ugyanúgy kell felparamétereznünk.

A backend teszteléséhez a PHPUnit-ot használjuk. Készítettünk 2 feature tesztet, ami a `/admin` végpont elérését teszteli. Az egyik teszt esetben generálunk egy email-t a FakerPHP osztály segítségével, majd létrehozunk az admin táblában egy egyedet ezzel az email címmel illetve egy user-t is. Ezek után megpróbáljuk elérni a végpontot a user-ként, akinek joga van elérni, és a 200-as HTTP választ várjuk.

```
public function testIfUserCanUseAdminPanel(): void
{
    $faker = \Faker\Factory::create();
    $email = $faker->email;
    Admin::create(['email' => $email]);
    $user = User::factory()->create(['email' => $email]);
    $this->actingAs($user);
    $response = $this->get('/admin');
    $response->assertStatus(200);
}
```

A másik esetben a fordítottját teszteljük, egy sima user-t hozunk létre és megnézzük, hogy elérjük-e így a `/admin` végpontot. Mivel nem lesz tagja az admin táblának így a 403-as HTTP válasz kódot várjuk, mivel nincs hozzáférése a `/admin` végponthoz.

```
public function testIfUserCannotUseAdminPanel(): void
{
    $user = User::factory()->create();
    $this->actingAs($user);
    $response = $this->get('/admin');
    $response->assertStatus(403);
}
```

Ezeket a teszteket a `php artisan test` paranccsal tudjuk lefuttatni, és ha minden teszt megfelel, akkor kapunk egy visszajelző üzenetet.

```
PS C:\xampp\htdocs\lostanimalproject> php artisan test

PASS Tests\Feature\AdminTest
✓ if user can use admin panel
✓ if user can not use admin panel

Tests: 2 passed (2 assertions)
Duration: 1.46s
```

21. ábra Teszt lefutása konzolban

4. Az adatbázis célja

A webalkalmazás egyik fő alkotóeleme az adatbázis, minden adat itt kerül eltárolásra és mindent innen kérdezzünk le. Arra törekedtünk, hogy egy redundancia-mentes adatbázist hozzunk létre, amelyben gyorsan végrehajthatók a lekérdezések, illetve arra, hogy az adatok konzisztensek maradjanak. Fontos szempont volt a biztonság is, hisz eltárolásra kerülnek szenzitív adatok, például telefonszám, lakhely, teljes név, stb.

4.1. Tervezés megkezdése

A tervezés kezdetekor át kellett gondolni, hogy pontosan milyen táblákra is lesz szükségünk, illetve azok milyen kapcsolatban álljanak egymással. Mivel a keretrendszerben voltak előre meghatározott táblák, így abból indultunk ki, és azokat alakítottuk át, vagy fejlesztettük tovább.

4.2. Tervezési lépések

Mivel még sosem terveztünk hasonlóan komplikált adatbázisokat, így először nehezen indultunk neki. A nehézséget a táblák összekötése okozta, többször neki kellett futni 1-1 táblának, mire azt működőképesre tudtuk alakítani.

Első lépésnek találnunk kellett egy olyan felületet, ahol mindketten hozzáférhetünk és meg tudjuk tervezni az adatbázisunkat. Cserhádi Máté rátalált a <https://dbdiagram.io> oldalra, ahol egyszerűen meg tudjuk tervezni az adatbázisunkat.

4.3. Egyed típusok/egyedek meghatározása

users: A userek (felhasználók) adatait tárolja.

cities: A települések nevét, irányítószámát és megye azonosítóját tárolja.

counties: A megye nevét és azonosítóját tárolja.

animals: Egy eltűnt állat adatait tárolja.

colors: A colors tábla a színek azonosítóját és nevét tárolja.

ch_favorites: Az user kedvencnek jelölhet más userekkel való beszélgetést és ezeket tárolja.

ch_messages: Az üzenetek, amelyek elküldésre kerültek.

password_reset_tokens: Az emailben elküldött jelszó-módosításhoz tartozó egyedi tokeneket tárolja.

4.4. Kapcsolatok meghatározása

A **users** 1:N-es kapcsolatban áll az **animals** táblával, mert 1 felhasználó több állatot is feltölthet, de egy állat csak 1 felhasználóhoz tartozhat.

A **cities** tábla 1:N-es kapcsolatban áll a **users** táblával, mert 1 település több felhasználóhoz is tartozhat, de egy felhasználó csak egy településhez tartozik.

A **counties** tábla 1:N-es kapcsolatban áll a **cities** táblával, mert egy megyéhez több település is tartozhat, de egy település csak egy megyéhez tartozik.

A **colors** tábla 1:N-es kapcsolatban áll az **animals** táblával, mert egy szín több állathoz is tartozhat, de egy állathoz csak egy szín tartozik.

4.5. Saját táblák

4.5.1. Users

Mező neve	Mező típusa	Mező leírása	megjegyzés
id	bigint	user azonosító (PK)	auto increment
name	varchar(255)	bejelentkezési név	
email	varchar(255)	email cím	
email_verified_at	timestamp	email megerősítés időpontja	
password	varchar(255)	jelszó	Laravel beépített titkosítás
remember_token	varchar(100)	bejelentkezés megjegyzéséhez tartozó token	
created_at	timestamp	user rekord létrehozásának ideje	
updated_at	timestamp	user rekord módosításának ideje	
firstname	varchar(255)	keresztnév	
lastname	varchar(255)	vezetéknév	
phonenumber	varchar(255)	telefonszám	
IRSZ_ID	int	irányítószám	unsigned, index
avatar	varchar(255)	kép fájlneve	
active_status	tinyint(1)	chatben aktív-e a felhasználó	
dark_mode	tinyint(1)	chat dark mode-ba van-e állítva	

messenger_color	varchar(255)		
-----------------	--------------	--	--

4.5.2. Cities

Mező neve	Mező típusa	Mező leírása	megjegyzés
IRSZ_Id	int	település irányítószám alapú azonosítója (PK)	unsigned
name	varchar(255)	település neve	
County_Id	int	megye azonosítója (FK)	unsigned, index

4.5.3. Counties

Mező neve	Mező típusa	Mező leírása	megjegyzés
id	int	megye azonosítója (PK)	unsigned
name	varchar(255)	megye neve	

4.5.4. Animals

Mező neve	Mező típusa	Mező leírása	megjegyzés
id	char(36)	állat azonosítója (PK)	unsigned auto increment
created_at	timestamp	az állat rekord létrejötte	
updated_at	timestamp	az állat rekord módosulása	
userId	bigint	user azonosító (FK)	unsigned, index
name	varchar(255)	állat neve	
chipNumber	varchar(255)	állat chip száma	nullable
gender	tinyint	állat neme	
colorId	bigint	szín azonosítója	unsigned
description	varchar(1000)	leírás	
image	varchar(255)	állatról a kép elérési útvonala	

colors:

Mező neve	Mező típusa	Mező leírása	megjegyzés
id	int	szín azonosító (PK)	unsigned
name	varchar(255)	szín neve	

admin:

Mező neve	Mező típusa	Mező leírása	megjegyzés
id	bigint	admin jog azonosítója (PK)	
email	varchar(255)	az admin email címe	

4.6. Keretrendszer táblái

4.6.1. Ch_favorites (chat kedvencek)

Mező neve	Mező típusa	Mező leírása	megjegyzés
id	char(36)	kedvenc-nek jelölés id-je	UUID
user_id	bigint	melyik user jelölt kedvencnek	
favorite_id	bigint	melyik usert jelölte a forrás user kedvencnek	
created_at	timestamp	létrehozás ideje	
updated_at	timestamp	frissítés ideje	

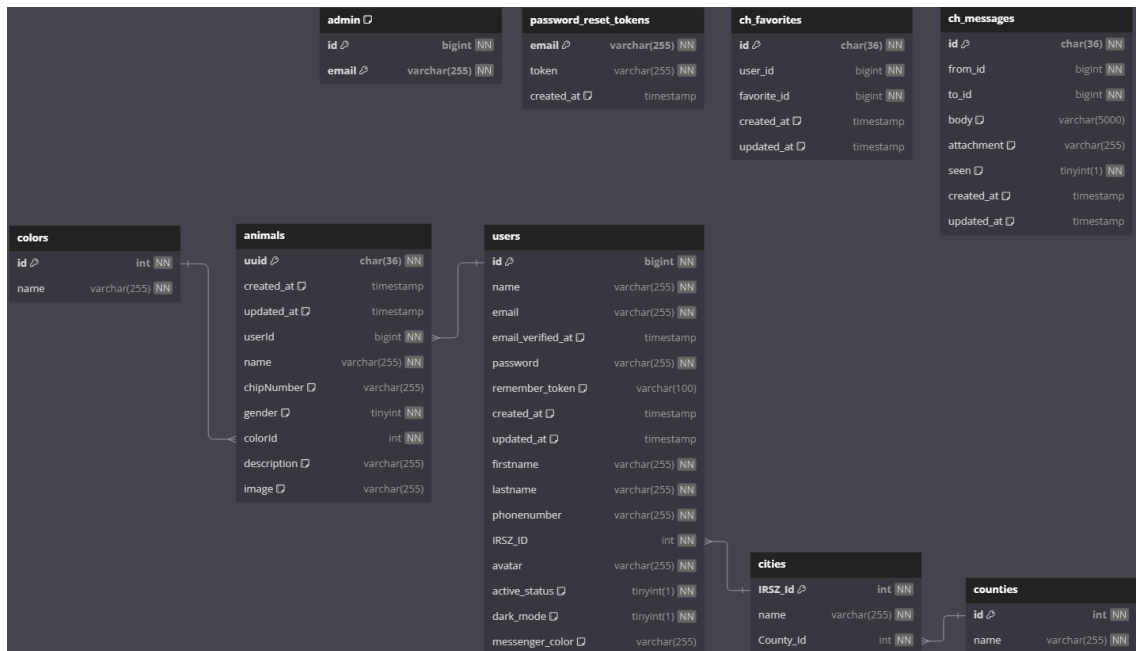
4.6.2. Ch_messages (üzenetek)

Mező neve	Mező típusa	Mező leírása	megjegyzés
id	char(36)	üzenet id-je	UUID
from_id	bigint	melyik user-től ment el az üzenet	
to_id	bigint	melyik user-nek érkezett meg az üzenet	
body	varchar	üzenet tartalma	
attachment	varchar	csatolt állományok elérési útvonala	
seen	tinyint	látták-e az üzenetet	
created_at	timestamp	üzenet elküldésének ideje	
updated_at	timestamp	üzenet frissítésének ideje	

4.6.3. Password_reset_tokens

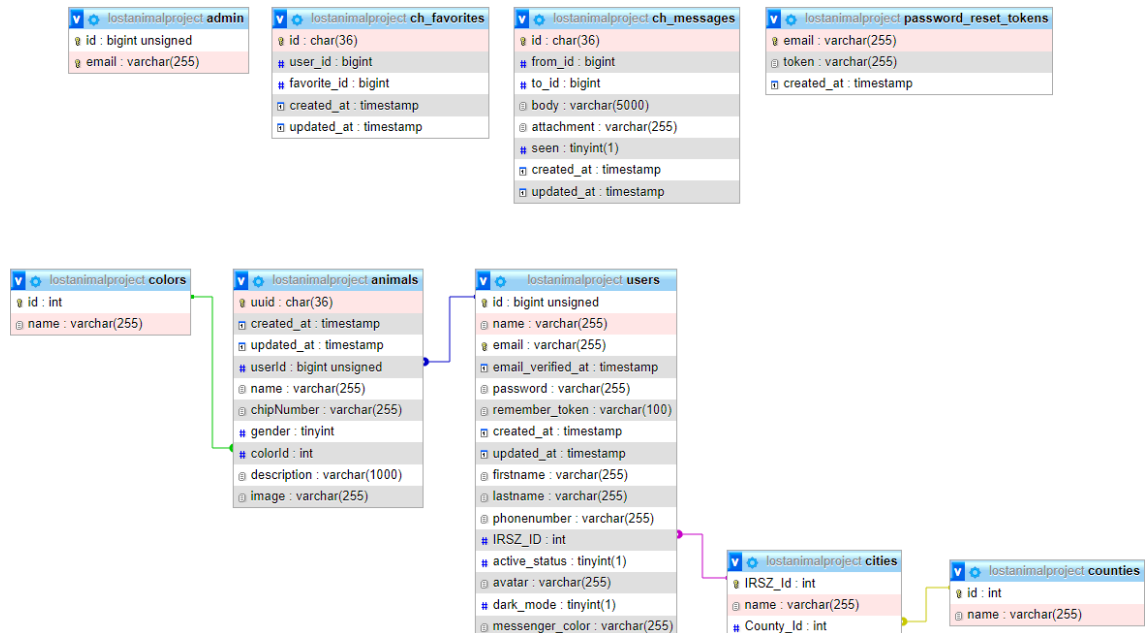
Mező neve	Mező típusa	Mező leírása	megjegyzés
email	varchar(255)	az email-t tárolja amire kérték a jelszó módosítást	
token	varchar(255)	az egyedileg generált token	
created_at	timestamp	a token létrehozásának ideje	

4.7. Adatbázis tervezése a dbdiagram felületen



22. ábra Adatbázis diagram a dbdiagram felületén

4.8. Adatbázis megvalósítása a phpmyadmin felületen



23. ábra Adatbázis diagram a phpmyadmin felületén

4.9. Adatbázis-továbbfejlesztési lehetőségek

Állatok fajának és fajtájának tárolása gyorsabb elérés és a felhasználói kényelem érdekében.

Animalimages tábla, ahol az állatról több képet is tudunk tárolni.

Animalvideos tábla, ahol az állatokról videókat is tudunk tárolni.

Településrészek kidolgozása. Jelenleg csak a fő települések vannak feltöltve, egyéb ugyanazon az irányítószámon található település nem került a rekordok közé pl. 2660 Balassagyarmat, de Nyírjes is a 2660 irányítószám alá tartozik.

5. Összefoglalás

Nagyban megkönnyítette a munkánkat, hogy az iskolában egymás mellett volt lehetőségünk párhuzamosan dolgozni, így „live coding” keretein belül könnyen megtudtuk valósítani a gyors munkát. Otthon saját gépen saját környezetben is fejlesztettünk a projekten, a kényelemnek és megszokott környezetnek köszönhetően otthon nagyobb produktivitást értünk el.

Az alkalmazás saját költségekből lett finanszírozva, mind a domain, mind pedig a virtuális szerver. Ezt szeretnénk a későbbiekben bővíteni, hogy nagyobb forgalom esetén ne legyen fennakadás, illetve egy load balancert is bevezetni, hogy levegye a terhet a távoli szerverről.

Szeretnénk az adatbázist a későbbiekben a MongoDB-re átültetni, ami egy gyorsabb, dokumentumorientált adatbázis.

Az API-t szeretnénk kibővíteni, hogy későbbiekben a natív mobil alkalmazást le tudjuk fejleszteni, ehhez szükséges átalakítani a háttérrendszert.

Szeretnénk behozni azt a lehetőséget is, hogy ne csak az állat tulajdonosa tölthesse fel az állatot, hanem az is, aki talált állatot, ezt persze elkülönítve, hogy ne keveredjenek. Ehhez át kell alakítanunk az állatokat tároló táblát.

Tervben van a globalizálás, hogy ne csak a magyaroknak legyen elérhető a szolgáltatás, hanem akár más országokban is. Ehhez szükséges a domain átregisztrálása .com-ra, a háttérrendszer átalakítása, és az adatbázis átalakítása.

Források

Web:

Fontok előtöltése: <https://stackoverflow.com/questions/4712242/wait-for-fonts-to-load-before-rendering-web-page>

XSS védelem: <https://stackoverflow.com/questions/27698977/how-and-where-can-xss-protection-be-applied-in-laravel>

Storage-ből kép törlése: <https://stackoverflow.com/questions/44710894/how-to-delete-images-from-public-images-folder-in-laravel-5-url-data>

Redirect in Laravel: <https://stackoverflow.com/questions/22950781/redirect-using-button-onclick>

Web háttér svg-k: <https://haikei.app/>

Tailwind dokumentáció: <https://tailwindcss.com/docs/installation>

Laravel dokumentáció: <https://laravel.com/docs/10.x>

JavaScript-ben PHP változó elérése: <https://stackoverflow.com/questions/4287357/access-php-variable-in-javascript>

JavaScript legördülő menü elemeinek törlése: <https://stackoverflow.com/questions/3364493/how-do-i-clear-all-options-in-a-dropdown-box>

CSRF token: <https://laravel.com/docs/10.x/csrf>

404-es kóddal manuálisan visszatérés: <https://stackoverflow.com/questions/43583906/redirect-to-404-page-automatically-at-laravel-5-4>

Laravel https kényszerítés: <https://stackoverflow.com/questions/70367815/make-laravel-use-https-by-default>

Hibaüzenetek megjelenítése: <https://stackoverflow.com/questions/26732821/displaying-the-error-messages-in-laravel-after-being-redirected-from-controller>

Laravel lokalizáció: <https://laravel.com/docs/10.x/localization>

MVC kép: <https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/1280px-MVC-Process.svg.png>

Adatbázis:

Táblák csatolási szabályai: <https://dba.stackexchange.com/questions/74627/difference-between-on-delete-cascade-on-update-cascade-in-mysql>

Városok és megyék forrása: https://webdraft.eu/orszagok_varosok/

Ábrajegyzék

1. ábra Főoldal.....	8
2. ábra Mobil nézet és sötét mód.....	9
3. ábra Publikus posztok mobil nézetben	10
4. ábra Publikus posztok asztali nézetben	10
5. ábra Publikus kártya	11
6. ábra Állat adatlapja.....	12
7. ábra Szűrés felugró ablaka	13
8. ábra Regisztráció	14
9. ábra Bejelentkezés és elfelejtett jelszó	15
10. ábra Főmenü	16
11. ábra Állat feltöltése	17
12. ábra Saját posztok.....	18
13. ábra Poszt szerkesztése.....	19
14. ábra Állat törlése előtt felugró ablak	20
15. ábra Chat.....	21
16. ábra Chat stílus módosítás	22
17. ábra Jelszó újítás.....	23
18. ábra Név és email módosítása	23
19. ábra Felhasználó törlése	23
20. ábra MVC architektúra	24
21. ábra Teszt lefutása konzolban	47
22. ábra Adatbázis diagram a dbdiagram felületén	52
23. ábra Adatbázis diagram a phpmyadmin felületén	52