Echomap

Árpás Kevin, Cservenák Dániel, Refka Bence
Nógrád Vármegyei Szakképzési Centrum Szent-Györgyi Albert Technikum
Szoftverfejlesztő és -tesztelő technikus (szakma azonosító: 5-0613-12-03)
2025. április 20.

Tartalom

1. Bevezetés

Projektünk két fő részből áll, ez magába foglal egy interaktív mobilalkalmazást, amely a Civitas Fortissima látványosságait mutatja be, valamint egy fórumszerű weboldalt az érdekeltek számára. Szülővárosunkban meghatározó szerepű Civitas Fortissima története, melyben Balassagyarmat lakói bátran kiálltak a város szabadságáért Trianont követően, s a város napjainkig is nagy hangsúlyt fektet ennek megemlékezésére, tanösvények, emlékművek és egyéb kulturális látványosságok formájában. Ennek inspirációjából létrehoztunk egy mobilalkalmazást, amely lehetőséget biztosít a felhasználóknak arra, hogy egy vizuálisan gazdag élmény keretében fedezzék fel a környezetüket egy virtuális térképen keresztül. Az interaktív ikonokra kattintva a felhasználók megtekinthetik az adott helyszín részleteit, beleértve annak nevét, leírását és fényképét.

A projekt teljes mértékben angol nyelven készült, mivel célközönségünk a Balassagyarmat történelmében érdekelt turisták. A jövőben tervezzük a projektünket magyar nyelvű fordításban is megvalósítani.

A weboldal ezzel szemben egy fórumszerű platformként működik. Így a felhasználók nemcsak információkat szerezhetnek a látványosságokról, hanem közösségi interakciókra is lehetőségük nyílik. A felhasználók könnyedén és hatékonyan tudnak interaktálni egymással, ötleteiket kiposztolni, megjegyzéseket írni ezekre az ötletekre, bekövetni egymást, s rákeresni bizonyos posztokra keresőfunkciók révén. Ezen felül a weboldalon megtekinthető a felhasználó számára az ő mobilalkalmazáson belül megtekintett látványosságainak száma egy haladási sáv keretein belül, mely zöld színűre váltva tudatja a felhasználóval, ha a mobilappon belül az összes nevezetességet meglátogatta. Továbbá, egy ranglista is található az oldalon, mely a legtöbb poszttal rendelkező felhasználók neveit listázza ki. A weboldal funkciói a napjainkban elterjedt kommunikációs közegeink mintájára készült, mint például a Reddit vagy X (korábban Twitter). Külön hangsúlyt fektettünk arra, hogy a weboldal és a mobilalkalmazás dizájnja hasonló stílusú legyen, ezzel érzékeltetve a két különálló szoftvertermék szoros kapcsolatát.

A projektben három fő működött közre, és a közös munka során számos új ismerettel gazdagodtunk. Alaposabban megismertük az általunk alkalmazott technológiai eszközöket, betekintést nyertünk a komplexebb webes rendszerek felépítésébe és működésébe, emellett gyakorlatot szereztünk a mobilalkalmazás-fejlesztésben is, különös tekintettel a Unity környezetben történő fejlesztésre. Árpás Kevin a háttérrendszer kialakítását és az adatbázis tervezését végezte, míg Cservenák Dániel a felhasználói felület megtervezéséért volt felelős. A mobilplatformra készült alkalmazást Refka Bence készítette el. A weboldalt a csapat a Laravel nevű keretrendszerben fejlesztette, a mobilalkalmazást Unity alapú megoldással valósította

meg. Fontos kiemelnünk, hogy sem a Laravel keretrendszer, sem a Unity fejlesztői környezet nem szerepel az iskolai tananyagban, ezek használatát önállóan, tanórán kívül sajátítottuk el, s ez tette ki a projektünk megvalósításának meghatározó hányadát.

A projekt webes felülete bárki számára elérhető a https://bdk.teamorange.hu/ linken keresztül, a teljes projekt forrása pedig a következő linken keresztül tekinthető meg: https://gitlab.teamorange.hu/ArpKevin/echomap. A repository tartalmaz minden releváns dokumentumot, valamint a weboldal forráskódjának mappájában egy rövid, angol nyelvű útmutatót is a rendszer telepítéséhez és üzemeléséhez.

2. Felhasznált technológiák

2.1.Google Icons

A Google Icons egy sokoldalú és széles körben használt ikonkészlet, amely letisztult, modern megjelenést kínál webes és mobilalkalmazások felhasználói felületeinek kialakításához. Ingyenes, nyílt forráskódú és könnyen integrálható megoldásként fejlesztők és designerek egyaránt használják különféle projektek vizuális egységének biztosítására. Népszerűsége abból ered, hogy széles ikonválasztékot nyújt, amely rendszeresen frissül, és jól illeszkedik a Material Design irányelveihez. A Google Icons egyszerűen használható HTML-ben, CSS-ben vagy akár különféle frontend keretrendszerekben, így gyors és hatékony eszköz bármilyen felhasználói felület fejlesztéséhez. Úgy lett kialakítva, hogy minden platformon egységes megjelenést biztosítson, reszponzív és stílusos módon. Ez a népszerű ikonkészlet alapvető vizuális elemként szolgál a modern alkalmazásfejlesztés során, és sokan választják, amikor gyorsan bevethető, esztétikus ikonokra van szükség.

2.2.Postman

Postman egy sokoldalú és széles körben használt API-kezelő eszköz, amely megkönnyíti a RESTful szolgáltatások tesztelését, fejlesztését és dokumentálását. Ingyenes, platformfüggetlen alkalmazásként elsősorban backend fejlesztők és tesztelők használják HTTP-kérések gyors és strukturált küldésére, valamint válaszok elemzésére. Népszerűsége abból ered, hogy felhasználóbarát felületet biztosít komplex API-hívások kezelésére, automatizált tesztek írására és környezetek kezelésére. A Postman lehetővé teszi kollekciók létrehozását, exportálását és megosztását, így ideális csapatmunkához és dokumentációk építéséhez. Úgy lett kialakítva, hogy átláthatóan kezelje az API-k működését és hibáit, ezáltal segíti a fejlesztőket a gyors és pontos hibakeresésben. Ez az eszköz mára szinte elengedhetetlen minden olyan webes projektben, ahol API-kommunikáció zajlik, és alapvető része a modern szoftverfejlesztési eszköztárnak.

2.3.DBDiagram

DBDiagram egy letisztult és hatékony online eszköz, amely vizuálisan jeleníti meg az adatbázis-struktúrákat, ezáltal segítve a tervezést, dokumentálást és csapatmunkát adatbázis-projektek során. Ingyenes, könnyen használható és böngészőalapú platformként ideális választás fejlesztők, adatmodellezők és projektmenedzserek számára, akik gyorsan szeretnék áttekinteni az adatkapcsolatokat. Népszerűsége annak köszönhető, hogy egyszerű szintaxis

segítségével generál ER-diagramokat, amelyeket exportálni, megosztani vagy dokumentációként használni is lehet. A DBDiagram úgy lett megtervezve, hogy intuitív felülettel és valós idejű szerkesztési lehetőségekkel támogassa a kollaboratív munkát. Ez az eszköz különösen hasznos adatbázisok elsődleges felépítése vagy refaktorálása során, mivel világos képet ad a táblák közötti kapcsolatról, így sok fejlesztő részesíti előnyben adatbázis-projektek kezdeti és karbantartási szakaszaiban.

2.4.Composer

Composer egy széles körben használt csomagkezelő rendszer PHP-hez, amely lehetővé teszi külső könyvtárak egyszerű telepítését és kezelését különböző projektekben. Ingyenes, nyílt forráskódú és könnyen használható eszközként a PHP-fejlesztők első számú választása, ha függőségek automatikus letöltéséről, frissítéséről vagy verziókezeléséről van szó. Népszerűsége abból ered, hogy egyértelműen meghatározható vele, melyik csomag melyik verziója szükséges, így biztosítva a projektek stabilitását és újratelepíthetőségét. A Composer nemcsak Laravel vagy Symfony típusú frameworkökkel működik jól, hanem bármilyen PHP-projekthez könnyen integrálható. Úgy lett kialakítva, hogy egyszerűen kezelhető legyen parancssorból, és jól skálázható maradjon nagyobb projekteknél is. Ez az eszköz ma már alapvető része a modern PHP-fejlesztésnek, és elengedhetetlen a hatékony és strukturált munkafolyamatok kialakításához.

2.5.JavaScript

A JavaScript egy sokoldalú és széles körben használt programozási nyelv, amely alapvető szerepet játszik a modern webfejlesztésben. Ingyenes, nyílt forráskódú és platformfüggetlen technológiaként lehetővé teszi interaktív és dinamikus felhasználói felületek létrehozását böngészőalapú alkalmazásokban. Népszerűsége abból ered, hogy kliensoldali és szerveroldali feladatokhoz egyaránt használható, valamint hatalmas ökoszisztémával és fejlesztői közösséggel rendelkezik. A JavaScript lehetővé teszi az események kezelését, adatok dinamikus megjelenítését, API-kommunikációt, és számos könyvtár vagy keretrendszer – mint például a React, Vue vagy Node.js – segítségével komplex alkalmazások fejlesztését is támogatja. Úgy lett kialakítva, hogy rugalmas és gyors legyen, miközben jól integrálható más webes technológiákkal, mint a HTML és a CSS. Ez a nagy teljesítményű nyelv ma már szinte minden webes fejlesztési projekt alapját képezi, és elengedhetetlen eszköz a frontend és egyre inkább a backend fejlesztésben is.

2.6.Laravel

A Laravel egy ingyenes és sokoldalú PHP keretrendszer, amely megkönnyíti a modern webalkalmazások fejlesztését. Egyszerű és elegáns szintaxisa révén gyors és hatékony munkafolyamatot biztosít a fejlesztők számára. Beépített funkciói, mint az adatbáziskezelés (Eloquent ORM), a hitelesítés, a routing és a sablonkezelés (Blade), lehetővé teszik skálázható és biztonságos alkalmazások létrehozását. A Laravel támogatja az MVC architektúrát, így jól strukturált és könnyen karbantartható kódot eredményez. Gyakran használják weboldalak, API-k és komplex rendszerek fejlesztésére, mivel rugalmassága és erőteljes ökoszisztémája miatt az egyik legnépszerűbb PHP keretrendszer a piacon.

2.7.PHPUnit

A PHPUnit egy népszerű, nyílt forráskódú tesztelési keretrendszer, amelyet a PHP nyelvhez fejlesztettek ki. Kifejezetten az automatizált egység- és funkcionális tesztek futtatására szolgál, és széles körben használják modern PHP-alapú alkalmazások minőségbiztosításában. A Laravel teljes körű támogatást nyújt a PHPUnit használatához, így egyszerűen létrehozhatók és futtathatók tesztek a php artisan test parancs segítségével. A tesztek során különféle assert metódusokkal ellenőrizhető az alkalmazás működése, például a válaszkódok, nézetek, vagy adatbázis-műveletek helyessége. A PHPUnit segítségével biztosítható, hogy a fejlesztés során bevezetett változtatások ne okozzanak hibákat más részekben, így hozzájárul a megbízható és jól karbantartható kód létrehozásához. Rendszeres használata a professzionális szoftverfejlesztési folyamatok szerves része.

2.8.Laravel Debugbar

A Laravel Debugbar egy sokoldalú és széles körben használt fejlesztői eszköz, amely valós idejű hibakeresést és teljesítményoptimalizálást tesz lehetővé Laravel alapú alkalmazásokban. Nyílt forráskódú, könnyen telepíthető csomagként elsősorban azok a fejlesztők használják, akik szeretnék átlátni az alkalmazás működését, különösen az adatbázis-lekérdezések és egyéb háttérfolyamatok szempontjából. Népszerűsége abból ered, hogy képes részletes információkat megjeleníteni az egyes requestekhez kapcsolódóan, beleértve a route-okat, a view-kat, a lekérdezéseket és a futási időket is, így jelentősen hozzájárul a hibák gyors megtalálásához és a kód optimalizálásához. A Debugbar úgy lett kialakítva, hogy valós időben nyújtson visszajelzést a fejlesztői környezetben, ezzel támogatva a hatékony munkát és a pontos hibakeresést. Ez a praktikus eszköz széles körben elterjedt a Laravel közösségben, és sokan használják komplex rendszerek fejlesztéséhez, mivel megbízható támogatást nyújt a projekt

teljesítményének és szerkezetének finomhangolásához. A nyílt forráskódú repository a következő webcímen található meg: https://github.com/barryvdh/laravel-debugbar

2.9.Mailtrap

A Mailtrap egy sokoldalú és megbízható emailtesztelő eszköz, amely ideális megoldást nyújt fejlesztési és tesztelési környezetekben történő emailküldés biztonságos kezelésére. Ingyenes verzióval is elérhető, könnyen integrálható különböző frameworkökkel, köztük a Laravel környezettel, ahol különösen hasznos lehet például regisztráció után küldött megerősítő emailek tesztelésére. Népszerűsége annak köszönhető, hogy valós emailcím használata nélkül is lehetővé teszi az üzenetek teljes tartalmának ellenőrzését, így megakadályozza a véletlenszerű kiküldéseket éles rendszerekre. A Mailtrap lehetőséget nyújt a fejlesztőknek arra, hogy részletesen átnézzék a fejléceket, HTML-tartalmat, mellékleteket és egyéb email-komponenseket. Ez az eszköz különösen jól alkalmazható Laravel projektekben, ahol a beépített SMTP konfiguráció révén gyorsan beállítható és megbízhatóan működik a regisztrációs folyamatok során, így sok fejlesztő választja a modern webalkalmazások emailkezelésének teszteléséhez.

2.10. JSONCrack

A JSON Crack egy sokoldalú és széles körben használt vizualizációs eszköz, amely számos olyan funkciót kínál, amelyek megkönnyítik a JSON-struktúrák átlátását és kezelését. Ingyenes, nyílt forráskódú és könnyen használható szerkesztőként elsősorban fejlesztők, adatmodellezők és API-tervezők használják összetettebb adatszerkezetek megjelenítésére és szerkesztésére. Népszerűsége abból ered, hogy képes egyszerűbbé és vizuálisan érthetőbbé tenni az adatkapcsolatok és hierarchiák kezelését, ezáltal növelve a hatékonyságot különböző fejlesztési és tesztelési feladatok során. A JSON Crack úgy lett megtervezve, hogy interaktív és valós idejű szerkesztési lehetőségeket kínáljon, így könnyen alkalmazható különféle munkafolyamatokban. Ez a praktikus eszköz támogatja a fejlesztőket és elemzőket az adatok gyors értelmezésében, strukturálásában és bemutatásában, ezért sokan választják a modern fejlesztői eszköztár részeként.

2.11. Uiverse

A Uiverse egy sokoldalú és széles körben használt fejlesztői platform, amely számos olyan vizuális komponenst és felhasználói felület elemet kínál, amelyek egyszerűsítik a webes alkalmazások kialakítását és fejlesztését. Ingyenes, nyílt forráskódú és nagymértékben

testreszabható eszközként elsősorban a frontend fejlesztők használják különböző projektek látványvilágának gyors és hatékony megvalósításához. Népszerűsége abból ered, hogy képes növelni a termelékenységet és a hatékonyságot azáltal, hogy előre elkészített, könnyen adaptálható UI-elemeket kínál. A Uiverse úgy lett megtervezve, hogy a fejlesztők könnyen megoszthassák és újrahasznosíthassák egymás munkáját, így rugalmasan igazítható az egyéni igényekhez. Ez a nagy teljesítményű platform támogatja a fejlesztőket a felhasználói élmény gyors kialakításában, finomhangolásában és egységesítésében, ezért a szoftverfejlesztő közösségben egyre többen választják.

2.12. Google Fonts

A Google Fonts egy ingyenesen használható, online betűtípus- és ikonkészlet-gyűjtemény, amely fejlesztők és tervezők számára kínál egyszerű megoldást modern és jól optimalizált tipográfia, valamint ikonhasználat integrálására webes és mobilprojektekben. A platform nemcsak több száz különféle betűtípust kínál, hanem a Google Fonts Icons segítségével könnyedén importálható és testreszabható ikonokat is biztosít. Ezek az ikonok skálázható vektoros formátumban érhetők el, ami biztosítja a tiszta és éles megjelenést bármilyen felbontáson. A betűtípusokhoz hasonlóan az ikonok is egyszerűen beépíthetők egyetlen HTML linkkel vagy CSS importálással, ami gyors fejlesztést és konzisztens dizájnt tesz lehetővé. A Google Fonts célja, hogy megbízható, platformfüggetlen és könnyen hozzáférhető eszközt biztosítson a vizuálisan igényes és jól strukturált webes élmények létrehozásához.

2.13. Windsurf

A Windsurf Editor a Codeium által fejlesztett sokoldalú és széles körben használható fejlesztőeszköz, amely számos olyan funkciót kínál, amelyek egyszerűsítik a kódolást és a projektmenedzsmentet. Ingyenes, nyílt forráskódú és nagymértékben testreszabható szerkesztőként elsősorban a szoftverfejlesztők használják különböző programozási nyelvek alkalmazásainak építéséhez. Népszerűsége abból ered, hogy képes növelni a termelékenységet és a hatékonyságot a kódolás, a szerkesztés, a hibakeresés és a projektek kezelése során. A Windsurf Editort úgy tervezték, hogy könnyen bővíthető legyen, így a fejlesztők saját igényeikhez igazíthatják. Ez a nagy teljesítményű környezet támogatja a fejlesztőket a kódbázisok könnyű létrehozásában, finomításában és karbantartásában, így a szoftverfejlesztő közösségben egyre többen választják.

2.14. Visual Studio Code

A Visual Studio Code (VS Code) egy sokoldalú és széles körben használt fejlesztőeszköz, amely számos olyan funkciót kínál, amelyek egyszerűsítik a kódolást és a projektmenedzsmentet. Ingyenes, nyílt forráskódú és nagymértékben testreszabható szerkesztőként elsősorban a szoftverfejlesztők használják különböző programozási nyelvek alkalmazásainak építéséhez. Népszerűsége abból ered, hogy képes növelni a termelékenységet és a hatékonyságot a kódolás, a szerkesztés, a hibakeresés és a projektek kezelése során. A VS Code-t úgy tervezték, hogy könnyen bővíthető legyen, így a fejlesztők saját igényeikhez igazíthatják. Ez a nagy teljesítményű környezet támogatja a fejlesztőket a kódbázisok könnyű létrehozásában, finomításában és karbantartásában, így a szoftverfejlesztő közösségben sokan választják.

2.15. Visual Studio 2022

A Visual Studio 2022 egy ingyenes és sokoldalú fejlesztőkörnyezet (IDE), amelyet a Microsoft fejlesztett ki, és lehetővé teszi különféle programozási nyelvek, például C#, C++, Python és JavaScript használatát. A fejlesztők hatékony eszközöket kapnak a kódíráshoz, hibakereséshez és verziókezeléshez, miközben a beépített IntelliSense és AI-alapú kódjavaslatok gyorsítják a munkát. A Visual Studio 2022 támogatja az asztali, webes, mobil- és felhőalapú alkalmazások fejlesztését, valamint a Unity integráció révén játékfejlesztésre is kiváló. Gyakran használják egyéni fejlesztők, kis csapatok és nagyvállalatok egyaránt, mivel modern, gyors és skálázható megoldást kínál bármilyen programozási projekthez.

2.16. HTML

A HTML (Hypertext Markup Language) egy egyszerű és könnyen érthető jelölőnyelv, amely a weboldalak szerkezetének és tartalmának felépítésére szolgál. A webfejlesztés alapvető eszköze, lehetővé téve a fejlesztők számára, hogy statikus és dinamikus weboldalakat hozzanak létre. A HTML nélkülözhetetlen eszköz minden webfejlesztő számára, mivel a webes tartalom alapvető építőeleme. Segítségével a fejlesztők hatékonyan formázhatják és hozhatnak létre webes tartalmakat, így kulcsszerepet játszik a modern webes projektekben.

2.17. CSS

A CSS (Cascading Style Sheets) egy stíluslapnyelv, amely a weboldalak vizuális megjelenésének finomhangolására és testreszabására szolgál. A modern webfejlesztésben elengedhetetlen eszköz, mivel lehetővé teszi a tartalom (HTML) és a dizájn szétválasztását, így

rugalmasabbá és hatékonyabbá téve a fejlesztési folyamatot. A CSS segítségével a fejlesztők kreatív és egyedi vizuális megoldásokat hozhatnak létre, hozzájárulva a weboldalak esztétikusabb és professzionálisabb megjelenéséhez. Emellett javítja a felhasználói élményt, így kulcsszerepet játszik a webes projektek sikerében.

2.18. PHP

A PHP (Hypertext Preprocessor) egy szerveroldali szkriptnyelv, amelyet főként dinamikus weboldalak és webalkalmazások létrehozására használnak. Rugalmas és hatékony eszköz, amely lehetővé teszi a fejlesztők számára, hogy interaktív funkciókat hozzanak létre, dinamikus tartalmakat generáljanak, és adatbázisokkal kommunikáljanak webes környezetben. Széles körben alkalmazott és támogatott nyelv, amely kiválóan alkalmas modern webes projektek megvalósítására. A PHP segítségével gazdag funkcionalitású, dinamikus és interaktív weboldalakat hozhatnak létre a fejlesztők, így kulcsszerepet játszik a webfejlesztésben.

2.19. XAMPP

A XAMPP egy nyílt forráskódú, ingyenes szoftvercsomag, amelyet főként webfejlesztők használnak lokális fejlesztői környezet beállításához. Ez az eszköz lehetővé teszi a fejlesztők számára, hogy PHP és MySQL alapú webalkalmazásokat és weboldalakat hozzanak létre, teszteljenek és fejlesszenek anélkül, hogy külső szerverre lenne szükségük. A XAMPP hatékony és könnyen használható megoldás, amely gyorsan és egyszerűen beállítható, így ideális választás a lokális fejlesztési folyamatok támogatására.

2.20. MySQL

A MySQL egy hatékony és robosztus, nyílt forráskódú relációs adatbázis-kezelő rendszer, amelyet széles körben használnak webalkalmazások és weboldalak adatainak kezelésére. Számos fejlesztő számára az első választás a relációs adatbázisok területén, mivel lehetővé teszi az adatok strukturált tárolását és gyors lekérdezését. A webfejlesztés során kiemelkedően népszerű megoldás, amely megbízhatóan támogatja a dinamikus webes projektek adatkezelési igényeit.

2.21. GitHub

A GitHub egy webalapú platform, amelyet szoftverfejlesztők használnak verziókezelés, kódmegosztás és projektmenedzsment céljából. Ez az eszköz lehetővé teszi a fejlesztők számára, hogy változtatásokat kövessenek nyomon, hatékonyan együttműködjenek, és munkájukat koordinálják. A szoftverfejlesztésben kulcsfontosságú, mivel egyszerűsíti a

verziókezelést, javítja a csapatmunka hatékonyságát, és átláthatóbbá teszi a projektmenedzsment folyamatait. Számos fejlesztő és szervezet számára nélkülözhetetlen eszköz a projektek hatékony kezeléséhez és nyomon követéséhez.

2.22. Github Desktop

A GitHub Desktop egy ingyenes és sokoldalú verziókezelő kliens, amely lehetővé teszi a fejlesztők számára, hogy egyszerűen kezeljék GitHub-projektjeiket egy vizuális felületen keresztül. Intuitív kezelőfelületével megkönnyíti a repositoryk klónozását, commitok létrehozását, branch-ek kezelését és pull requestek kezelését anélkül, hogy parancssort kellene használni. Ideális eszköz kezdők és tapasztalt fejlesztők számára egyaránt, hiszen gyorsítja a munkafolyamatokat és átláthatóbbá teszi a csapatmunkát. Windows és macOS rendszereken egyaránt elérhető, így széles körben használható bármilyen fejlesztési környezetben.

2.23. Discord

A Discord egy ingyenes, sokoldalú kommunikációs platform, amely lehetővé teszi a felhasználók számára, hogy hanghívásokon, szöveges üzeneteken vagy videóhívásokon keresztül kapcsolatban maradjanak. A felhasználók szerverekre csatlakozhatnak, ahol csoportos beszélgetéseket folytathatnak, játékokban vehetnek részt, vagy más közös tevékenységeket szervezhetnek. Gyakran használják játékosok és közösségek, hogy zökkenőmentesen kommunikáljanak játék közben vagy közös projektek során, de alkalmazása messze túlmutat ezen. A platform sokoldalúsága miatt bármilyen célra használható, ahol gyors és hatékony online kommunikációra van szükség.

2.24. Unity

A Unity egy ingyenes és sokoldalú játékmotor, amely lehetővé teszi a fejlesztők számára, hogy 2D és 3D játékokat, alkalmazásokat és interaktív élményeket hozzanak létre. A platform széles körben támogatja a különböző eszközöket, beleértve a PC-t, konzolokat, mobilokat és a VR/AR rendszereket. Az egyszerű kezelőfelület és a C# nyelven írt szkriptek révén kezdők és profi fejlesztők egyaránt hatékonyan dolgozhatnak vele. Gyakran használják indie fejlesztők, nagyobb stúdiók és más iparágak is, például filmkészítésben vagy építészeti vizualizációban. A Unity rugalmassága és erőteljes eszköztára miatt az egyik legnépszerűbb játékmotor a piacon.

2.25. Mapbox SDK (Unity)

A Mapbox SDK egy ingyenes és sokoldalú eszközkészlet Unity-hez, amely lehetővé teszi térképek és helyalapú alkalmazások létrehozását. A fejlesztők dinamikus, testreszabható 2D és

3D térképeket integrálhatnak projektjeikbe, támogatva a valós idejű navigációt, adatvizualizációt és interaktív földrajzi élményeket. Az SDK segítségével könnyedén kezelhetők a térképrétegek, épületmodellek, útvonaltervek és egyéb helyalapú adatok. Gyakran használják játékokban, városi szimulációkban és AR/VR alkalmazásokban. Rugalmassága és széleskörű funkcionalitása miatt ideális eszköz mind kezdők, mind tapasztalt fejlesztők számára.

2.26. .NET Keretrendszer

A Microsoft által létrehozott .NET egy olyan fejlesztői környezet, amely többféle nyelvet és technológiát egyesít, lehetővé téve különböző típusú alkalmazások létrehozását. A rendszer különösen népszerű az üzleti és ipari szférában, köszönhetően rugalmasságának és széles körű támogatottságának. A platform célja, hogy gyors és hatékony megoldásokat nyújtson különböző fejlesztési igényekre, miközben modern eszközkészletet és több operációs rendszeren való működést biztosít. A .NET egyik kiemelkedő tulajdonsága, hogy integrált támogatást nyújt különféle nyelvekhez, és elősegíti az alkalmazások széles körű elérhetőségét.

2.27. C#

A C# nyelv a Microsoft által kidolgozott, korszerű fejlesztőeszköz, amely szorosan összefonódik a .NET 7 infrastruktúrával. Ez a nyelv típusbiztonságot kínál, objektumorientált szemléletet követ, és alkalmas különféle programok, rendszerek vagy szolgáltatások megalkotására. A C# különösen népszerű a fejlesztők körében, mivel olyan funkciókat kínál, amelyek megkönnyítik a robusztus és skálázható alkalmazások létrehozását. A .NET-tel való szoros integráció révén a nyelv hatékony eszközként szolgál platformfüggetlen fejlesztési feladatokhoz is, legyen szó asztali, webes vagy mobil megoldásokról.

2.28. Microsoft Word

A Microsoft által kifejlesztett Word egy elterjedt digitális szerkesztőeszköz, amely része az Office programcsomagnak. Segítségével a felhasználók különféle írásos anyagokat készíthetnek el és alakíthatnak át, például hivatalos leveleket, jelentéseket, pályázatokat vagy más dokumentumtípusokat. A szoftver a dokumentumkezelés teljes folyamatát támogatja, beleértve a tartalom létrehozását, stilizálását és végső formába öntését. Népszerűségét felhasználóbarát felülete és a platformok széles skáláján való használhatósága is erősíti, így ideális választás a mindennapi irodai vagy tanulmányi feladatokhoz.

3. Felhasználói dokumentáció

3.1.Webes Felület

3.1.1. Főoldal

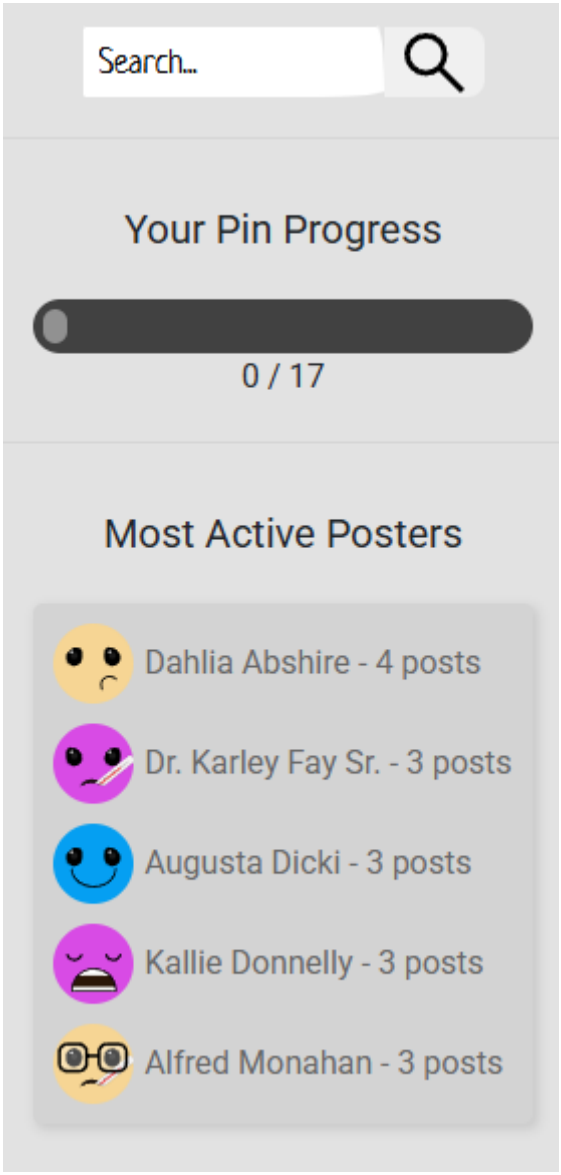
Az weboldalra való belépés után elsőnek a főoldalra érkezünk bejelentkezés állapotától függetlenül. A weboldal felső részén a felhasználó megtalálja a navigációs menüt. Ha a felhasználó nincs bejelentkezve, akkor a navigációs menü sorrendje a “Home” gomb, ami a főoldalra irányít, “Login” gomb, ami átirányít a bejelentkezési oldalra, aztán “Register” gomb, ami átirányít a regisztrációs weboldalra. Ha a felhasználó be van jelentkezve, akkor a sorrend pedig “Home” gomb, ami a főoldalra irányít, “Feed” gomb, ami a követett felhasználók posztjaiknak leszűrt weboldalára irányít, “Profile” gomb, amely a saját profilhoz irányít át, és legvégül a “Logout” gomb, amivel ki lehet jelentkezni. Amennyiben a felhasználó admin jogosultsággal rendelkezik, a “Logout” gomb után megtalálható lesz az “Admin” gomb, ami átirányít az admin panelre.

1. ábra Navigációs sáv

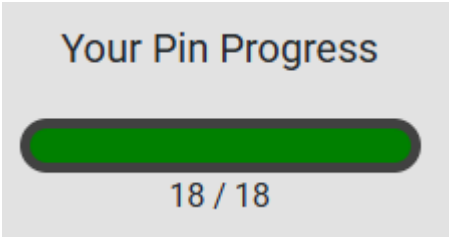
A belépési állapottól függően sok funkció elérhetetlenné válik. Ha a felhasználó nincs bejelentkezve, kizárólag a keresősáv, profilok és a posztok alatt lévő “kommentek” megtekinthetőek. A bejelentkezést követően válik elérhetővé az összes funkció a weboldalon, mint például a “like” gomb használata. Egy letiltott funkció használata, mint például egy gomb lenyomása, a belépési weboldalra való átirányítást okozza.

A weboldalon a felhasználó megtekintheti a regisztrált felhasználók által készített posztokat, illetve megtekintheti a regisztrált (“posztolók”) felhasználók profiljait, ami a weboldal közepén található. Bal oldalt találhatóak a posztok és a szövegszerkesztő, jobb oldalt talál a felhasználó egy keresősávot, ami egy szűrést fog végezni, s kizárólag azok a posztok fognak megjelenni, melyeknek szövege tartalmazza a keresett kulcsszót.

Ezen felül látható még egy lista a legtöbb posztot közzé tévő felhasználókról nevéről és a posztjaik mennyiségéről, valamint egy haladási sáv, ami azt mutatja, hogy a bejelentkezett felhasználó hány nevezetességet látogatott meg eddig. Amennyiben a felhasználó nincs bejelentkezve, a meglátogatott nevezetességek száma mindig 0. Amennyiben a felhasználó megnézte az összes elérhető pint, a sávban található világos szürke szín zölddé fog válni.

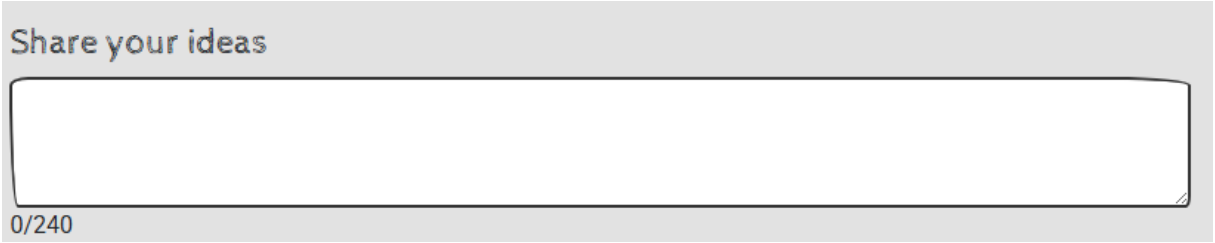


2. ábra Keresősáv, haladási sáv, ranglista



3. ábra Befejezett haladási sáv

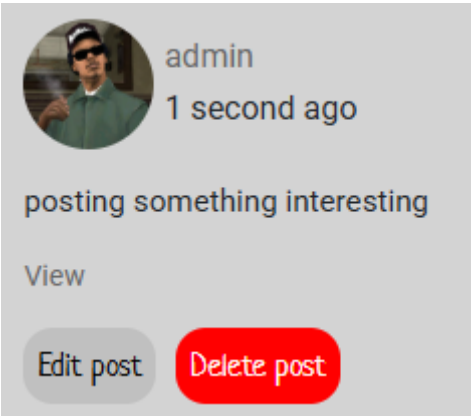
Amennyiben be vagyunk jelentkezve, a posztok felett található szövegszerkesztő mező segítségével mi is tudunk ötletet posztolni. Ellenkező esetben a weboldal azt írja ki, hogy jelentkezzünk be ötlet megosztásához (“Login to share your ideas”).



4. ábra Szövegszerkesztő mező ötlet posztolásához

5. ábra Tájékoztató szöveg kijelentkezett felhasználóknak

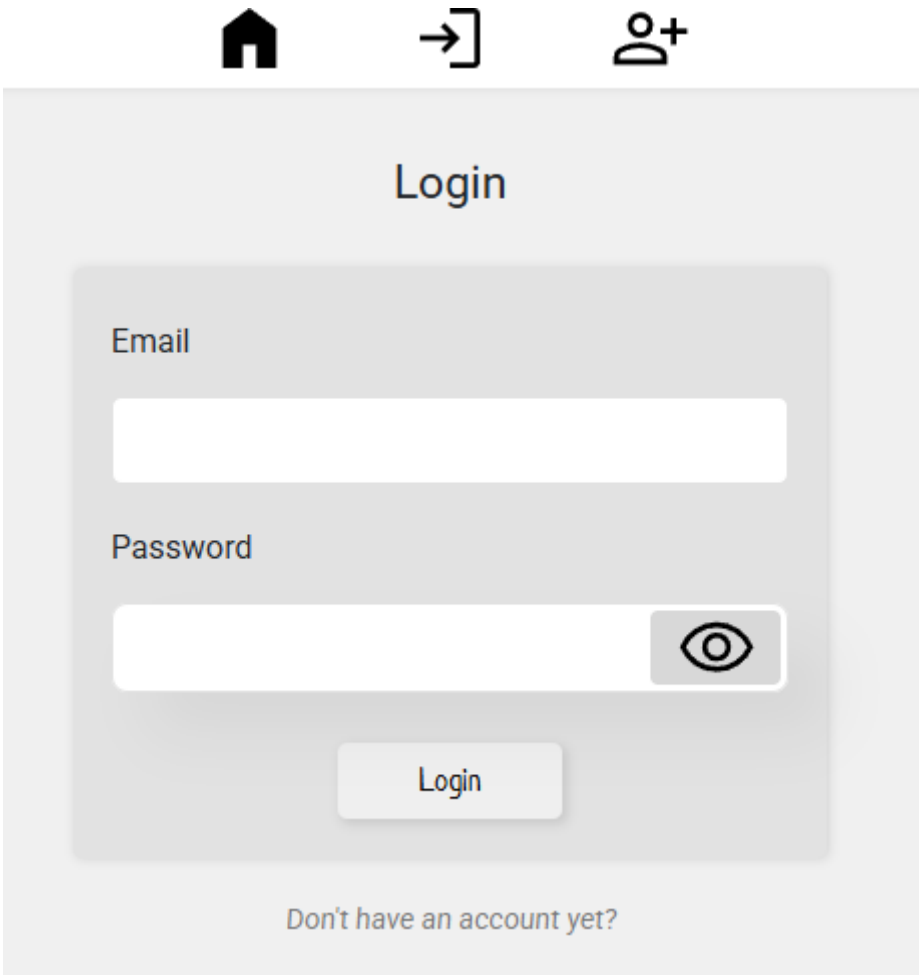
Egy posztban több információ látható. Legfelül látható a posztoló profilképe, felhasználóneve, illetve, hogy mikor tette közzé a felhasználó a posztot. Alatta található a poszt szövege. A poszt alján jobb oldalon található egy szív, amire rákattintva be tudjuk "like-olni" a posztot. A szív mellett lévő szám azt mutatja, hogy hány ember tette ezt meg előttünk. Legvégül látható egy "View" gomb, amire rákattintva a weboldal átirányít a posztra és a poszt alatt lévő kommentekre, és ezek egy külön oldalon fognak megjelenni. A felhasználó saját posztjain található még két további gomb, az "Edit post" és "Delete post" gombok. Az "Edit post" gombra kattintva a felhasználó a poszt szövegét megváltoztathatja egy beviteli mező segítségével. A mező alatt található két gomb, az "Update" és a "Cancel". Az "Update" gombot megnyomva a felhasználó sikeresen frissítette a poszt szövegét, a "Cancel" gombbal pedig minden módosítást eldobott és visszaállította az eredeti szövegre. A mentést követően a weboldal átirányít a főoldalra, ahol az adott poszt már a változtatások jegyében fog megjelenni. A "Delete post" gombra kattintva egy "popup" jelenik meg, amivel megerősítjük posztunk törlésének szándékát, majd ezt követően a poszt törlése megtörténik.



6. ábra Kiposztolt ötlet

3.1.2. Bejelentkezési weboldal

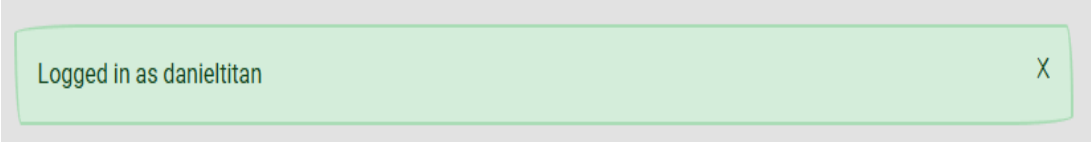
A felhasználónak szükséges a bejelentkezés a weboldal teljes szintű eléréséhez. A belépési oldalon két beviteli mező várja a felhasználót, ami kér egy emailt és egy jelszót.



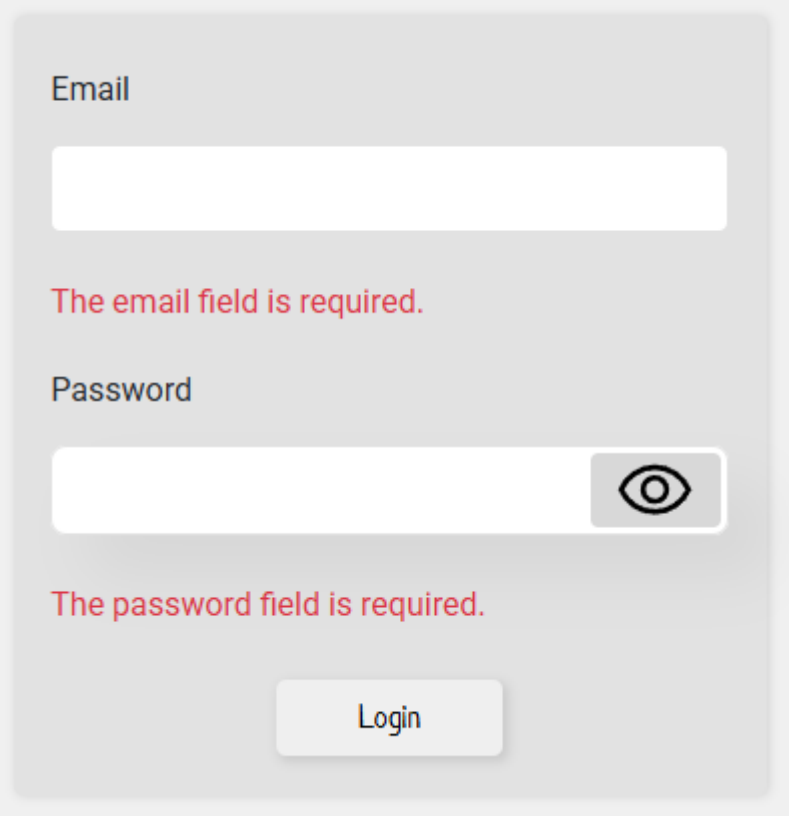
7. ábra Bejelentkezés

A jelszó mezőben található egy kis gomb, aminek az a funkcionalitása, hogy a jelszó begépelését követően megjeleníti a mezőben található szöveget, mindaddig, amíg a felhasználó nem engedi fel az egérgombot. A form alatt található egy button “Don’t have an account yet?” szöveggel, amire rákattintva a weboldal átirányítja a felhasználót a regisztrációs weboldalra. Amennyiben a felhasználó adatok felvitele nélkül próbál bejelentkezni, a weboldal hibaüzeneteket fog kiírni a következő szövegekkel: “The email field is required”, ha nem lett email cím megadva, vagy “The password field is required”, ha nem lett jelszó megadva. Ha az email cím formátuma nem megfelelő, vagy helytelen jelszót adott meg a felhasználó, szintén új hibaüzenetek jelennek meg. A jelszó 8-tól 20 karakterig terjedő hosszúságú lehet. Ha a jelszó túl rövid, vagy éppen túl hosszú, a weboldal továbbra sem engedi bejelentkezni a felhasználót. Ha a felhasználó valós adatokat adott meg, a weboldal főoldalra fogja őt átirányítani, s egy zöld hátterű tájékoztató szöveg értesíti a felhasználót arról, hogy sikeresen belépett. A konténer jobb

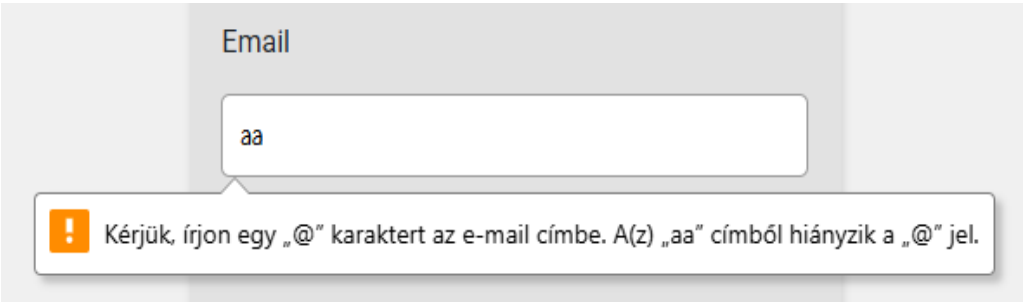
oldalán található egy kis “X” gomb, amivel a tájékoztató szöveget el lehet tüntetni. A rendszer a felhasználónak a weboldalon végrehajtott változtatásairól ennek a szövegdoboznak a segítségével ad visszajelzést, például amikor közzétesz egy posztot, vagy bekövet egy másik felhasználót.



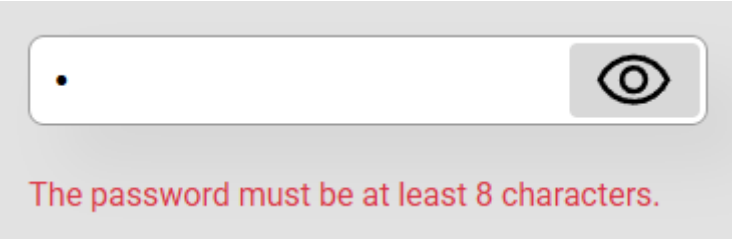
8. ábra Megerősítő szöveg bejelentkezésről



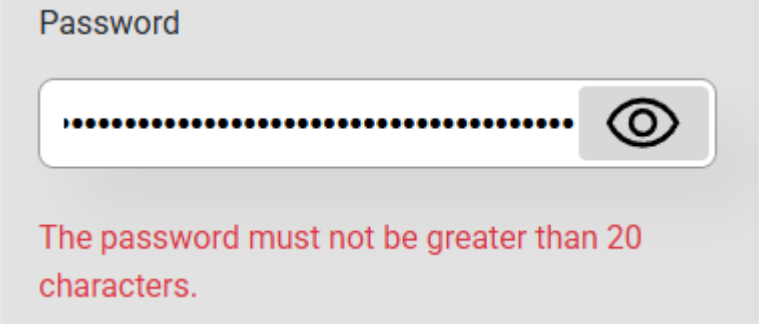
9. ábra Hibaüzenet bejelentkezéskor I.



10. ábra Hibaüzenet bejelentkezéskor II.



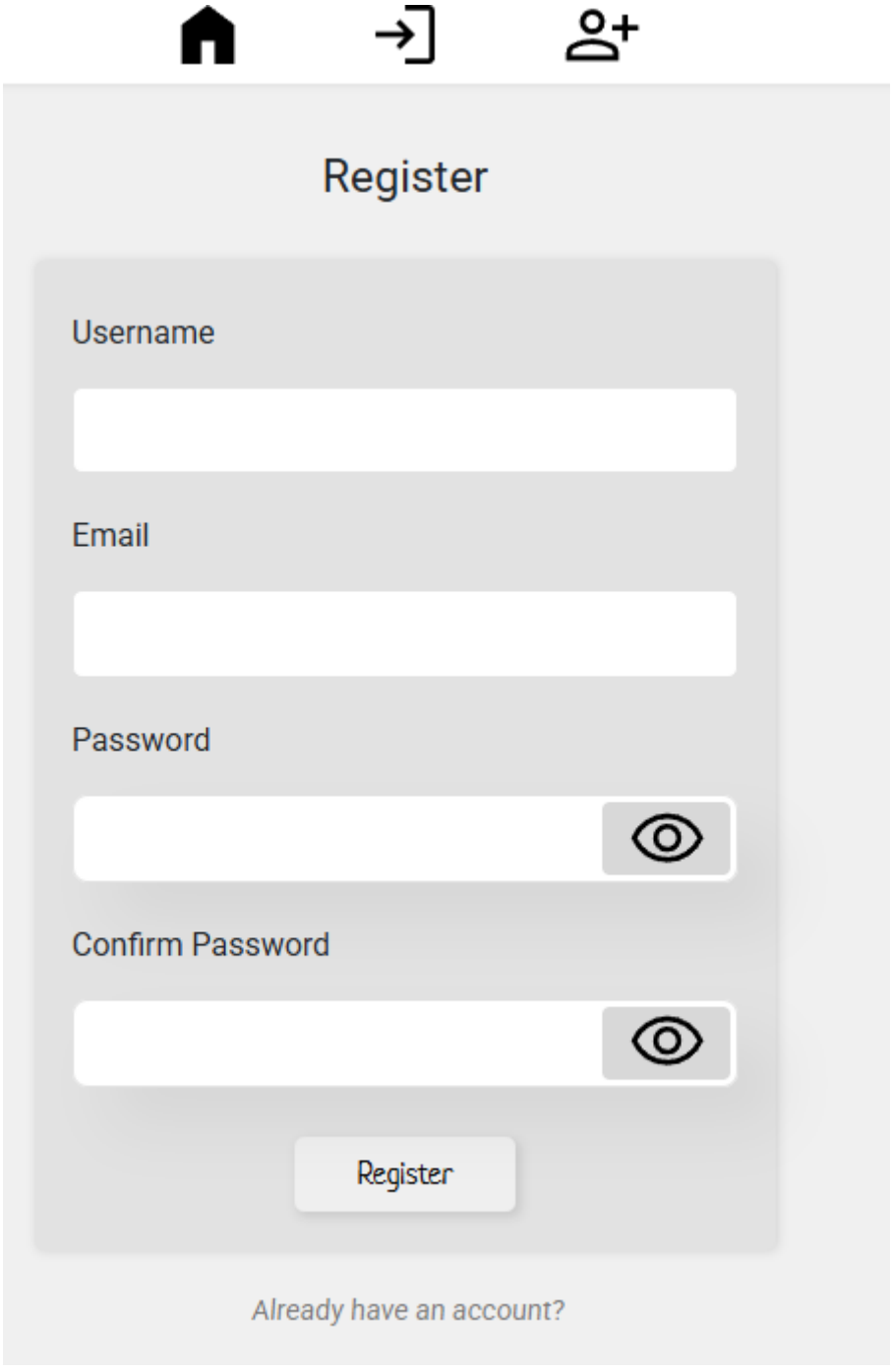
11. ábra Hibaüzenet bejelentkezéskor III.



12. ábra Hibaüzenet bejelentkezéskor IV.

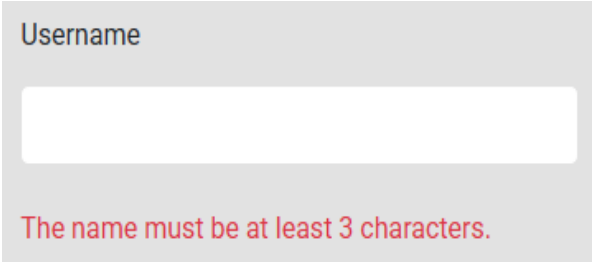
3.1.3. Regisztrációs weboldal

A regisztrációs oldal kinézetre hasonló, mint a bejelentkezési oldal, itt viszont a regisztrálni kívánt nevet és email címet kéri a felhasználótól, továbbá egy jelszót, majd végül ennek megerősítését.



13. ábra Regisztráció

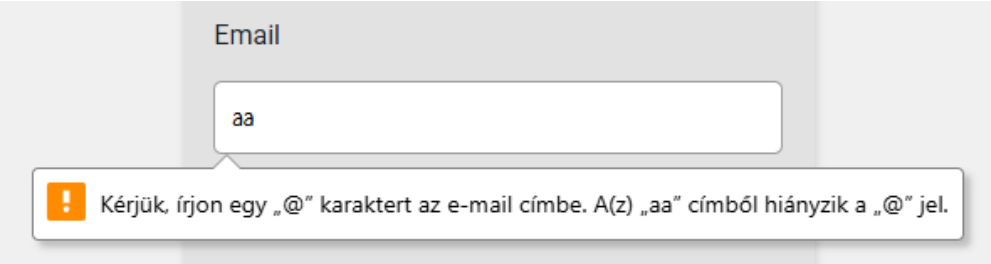
A begépelt adatokkal hasonló ellenőrzések sora megy végbe, mint a bejelentkezési oldal esetében. Ha például a felhasználó 3-nál kevesebb karaktert gépel be a név mezőbe, a “The name must be at least 3 characters.” hibaüzenet fog megjelenni. Ellentétben ezzel, ha 40 karakternél hosszabb a beírt szöveg a mezőbe, a “The name must not be greater than 40 characters.” hibaüzenet fog megjelenni.



14. ábra Hibaüzenet regisztrációkor I.

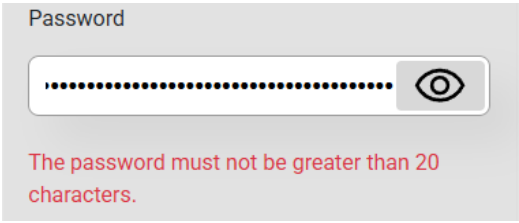
15. ábra Hibaüzenet regisztrációkor II.

Az email cím felviteli mezőjében kötelező az úgynevezett “kukac” (@) karakter az email-cím valóságának ellenőrzése miatt. Ha a kukac karakter hiányzik, egy hibaüzenet fog megjelenni, ami tájékoztatja a felhasználót erről.

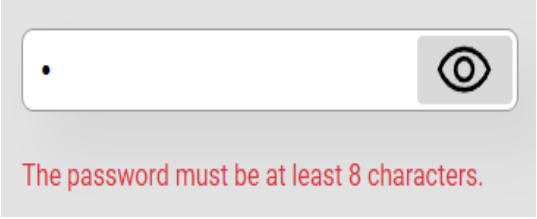


16. ábra Hibaüzenet regisztrációkor III.

Az email cím mező alatt található a jelszó mező, ami minimum 8, illetve maximum 20 karaktert követel meg.

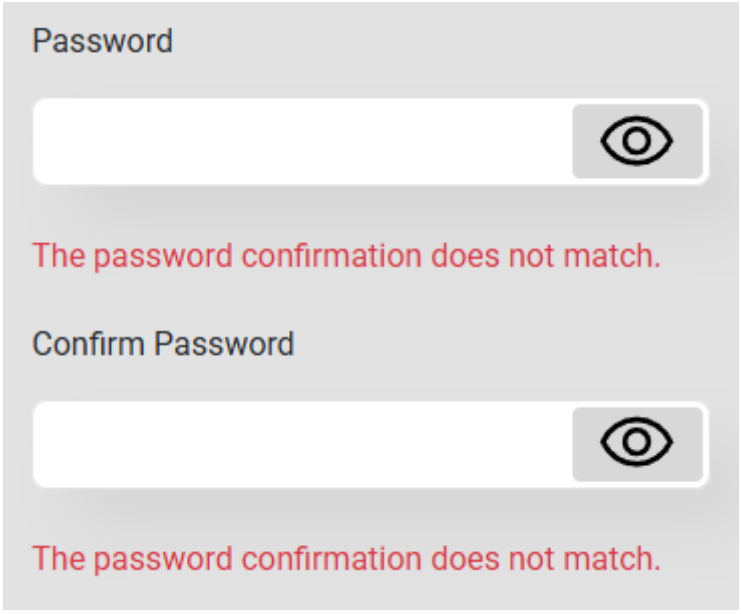


17. ábra Hibaüzenet regisztrációkor IV.



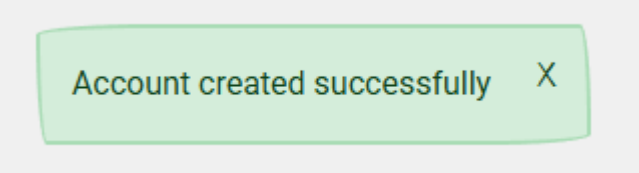
18. ábra Hibaüzenet regisztrációkor V.

A regisztrációs weboldalon is megtekinthetjük jelszónkat, mielőtt felvisszük adatként. Ha a jelszó és a jelszó megerősítése mező nem egyezik meg, akkor a weboldal ki fog írni egy hibaüzenetet.



19. ábra Hibaüzenet regisztrációkor VI.

A form alatt található egy button az “Already have an account” szöveggel, amire rákattintva a weboldal átirányít a belépési weboldalra. A felhasználó sikeres regisztrációját követően a weboldal átirányítja őt a login weboldalra egy visszajelző szövegdobozzal.



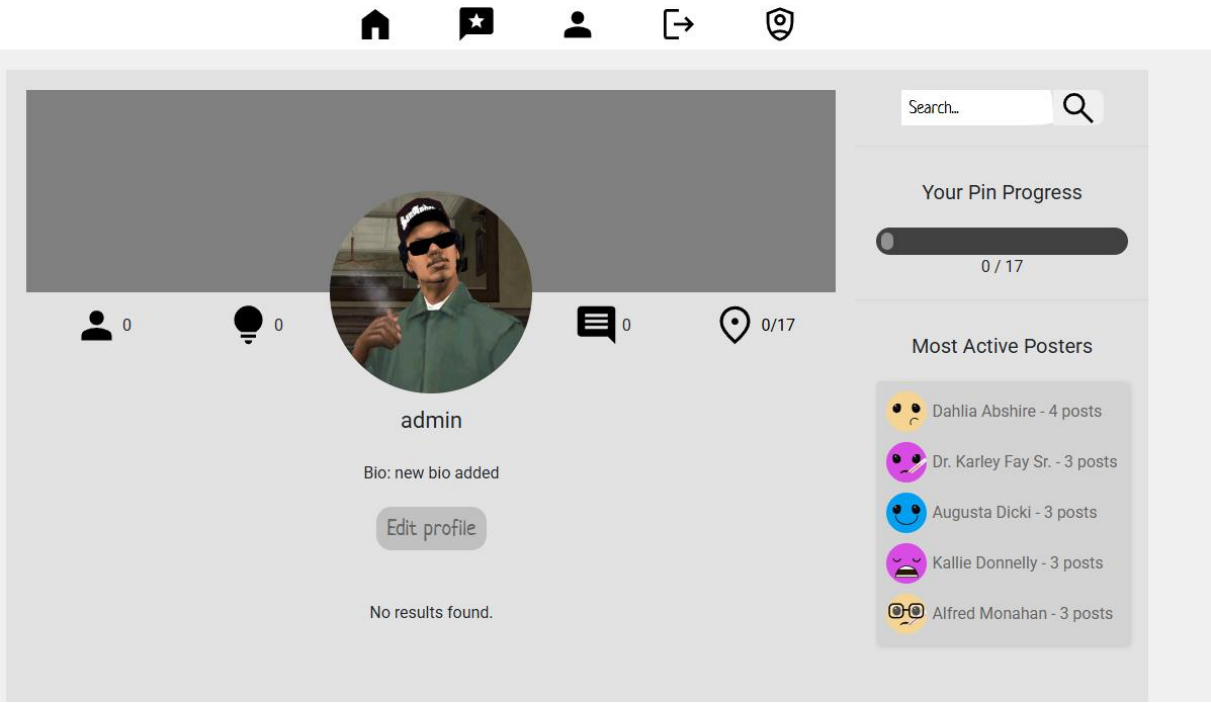
20. ábra Megerősítő szöveg fióklétrehozásról

3.1.4. Profil weboldal

A felhasználónak a bejelentkezés után elérhetővé válik a profiluk megtekintése a megújult navigációs menü segítségével, ahol a következő ikon kattintására átirányítja a weboldal a felhasználót:

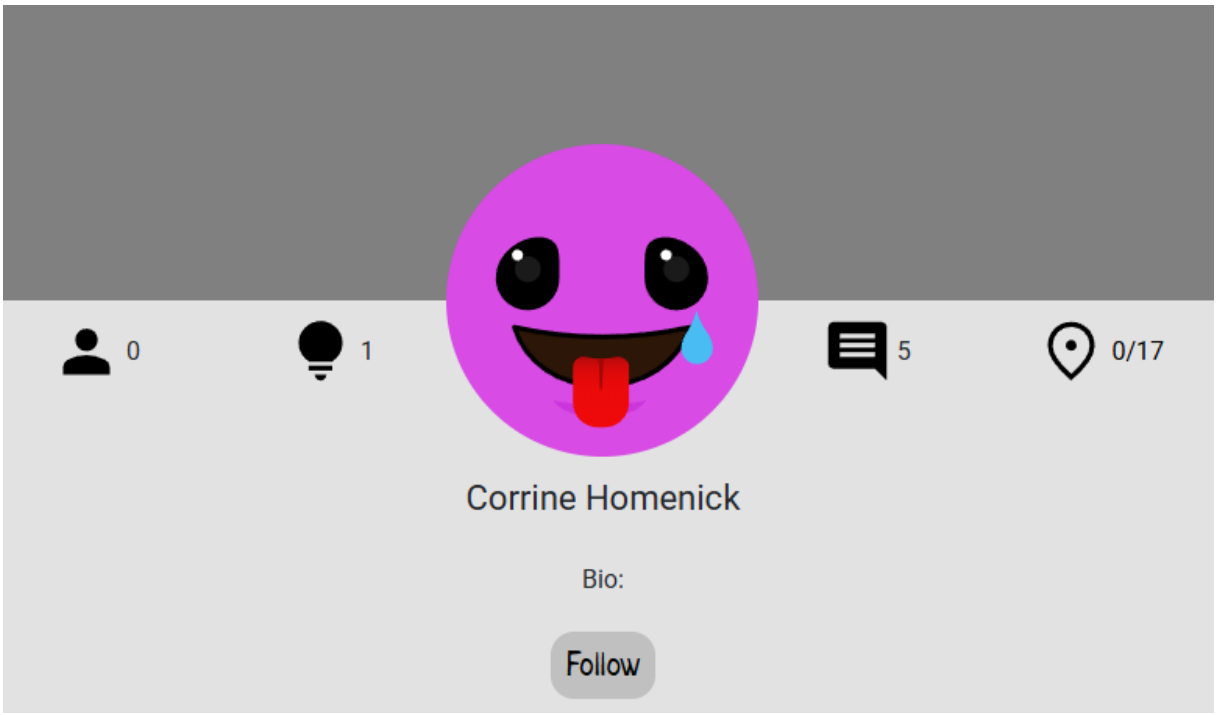
21. ábra Profil oldal navigációs ikon

Ez a weboldal megjeleníti a felhasználó saját posztjait és adatait, mint például a nevüket, a profilképüket, vagy éppen a felhasználó egyedi statisztikáit a weboldalon belül.



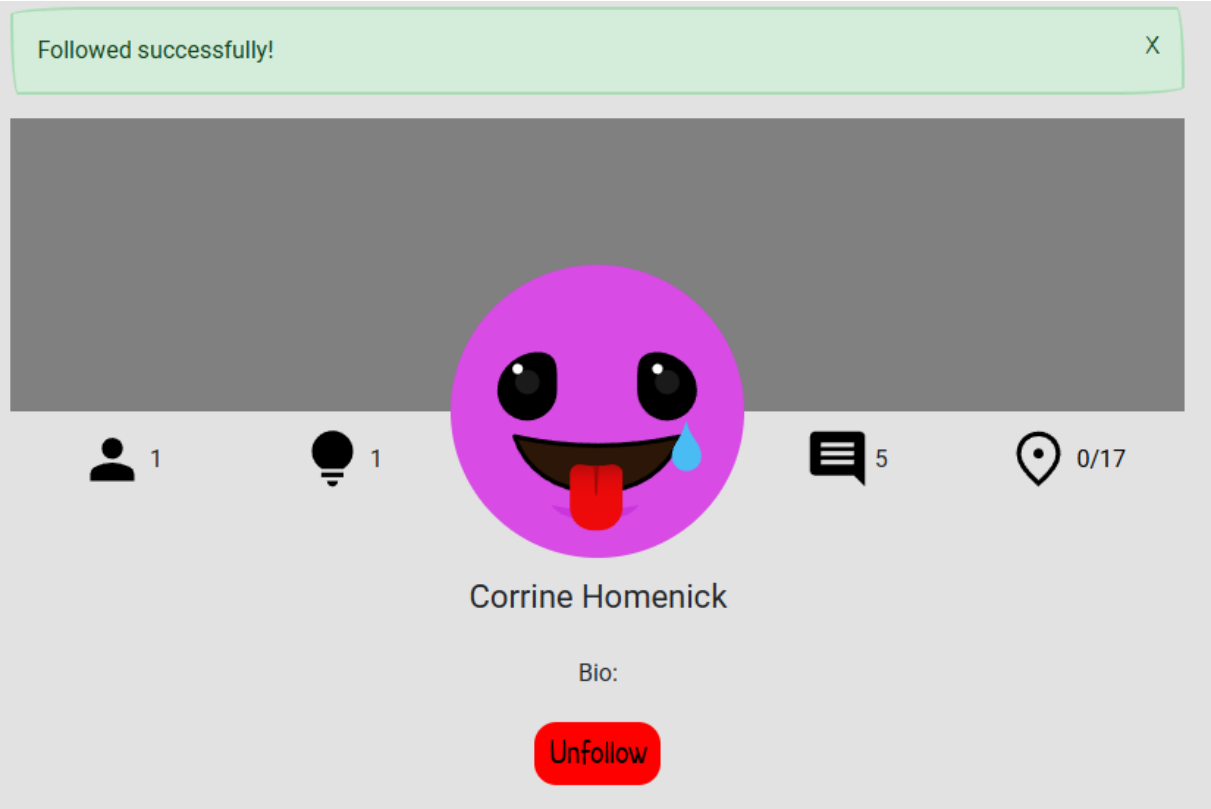
22. ábra Profil oldal

Attól függően, hogy a felhasználó valaki más profilját, vagy a sajátját nézi, 2 különböző gomb jelenik meg. Amennyiben valaki más profilját tekinti meg, egy “Follow” gomb fog megjelenni, amivel az adott felhasználót be lehet követni.



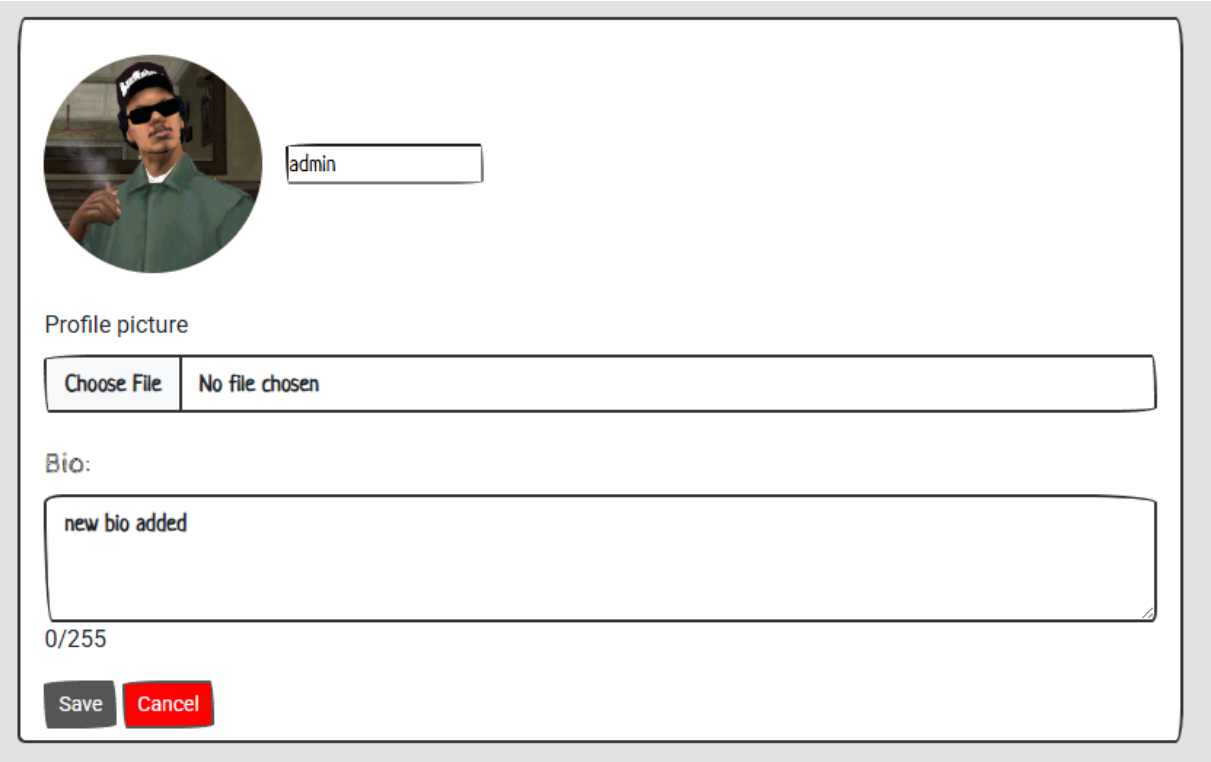
23. ábra Más felhasználó profilja

Ha a felhasználót már követjük, az “Unfollow” gomb segítségével szüntethetjük meg ezt a követést.



24. ábra Felhasználó bekövetése

Amennyiben viszont a saját profilját tekinti meg a felhasználó, az “Edit profile” szövegű gomb fog megjelenni. Erre rákattintva a felhasználó a saját adatait módosíthatja. Ezealatt értve a felhasználónevét, a profilképét, továbbá a “bio” szövegmező segítségével egy rövid szöveget jeleníthet meg önmagáról a weboldalon. A szövegszerkesztő felület alján található még két gomb, a “Save”, illetve “Cancel” szövegű gombok. A felhasználó a módosításait a “Save” szövegű gomb segítségével mentheti el. A felület eltüntetéséhez a “Cancel” szövegű gombot megnyomva átirányít minket a rendszer a profilunk oldalára, a változtatásokat eldobva.



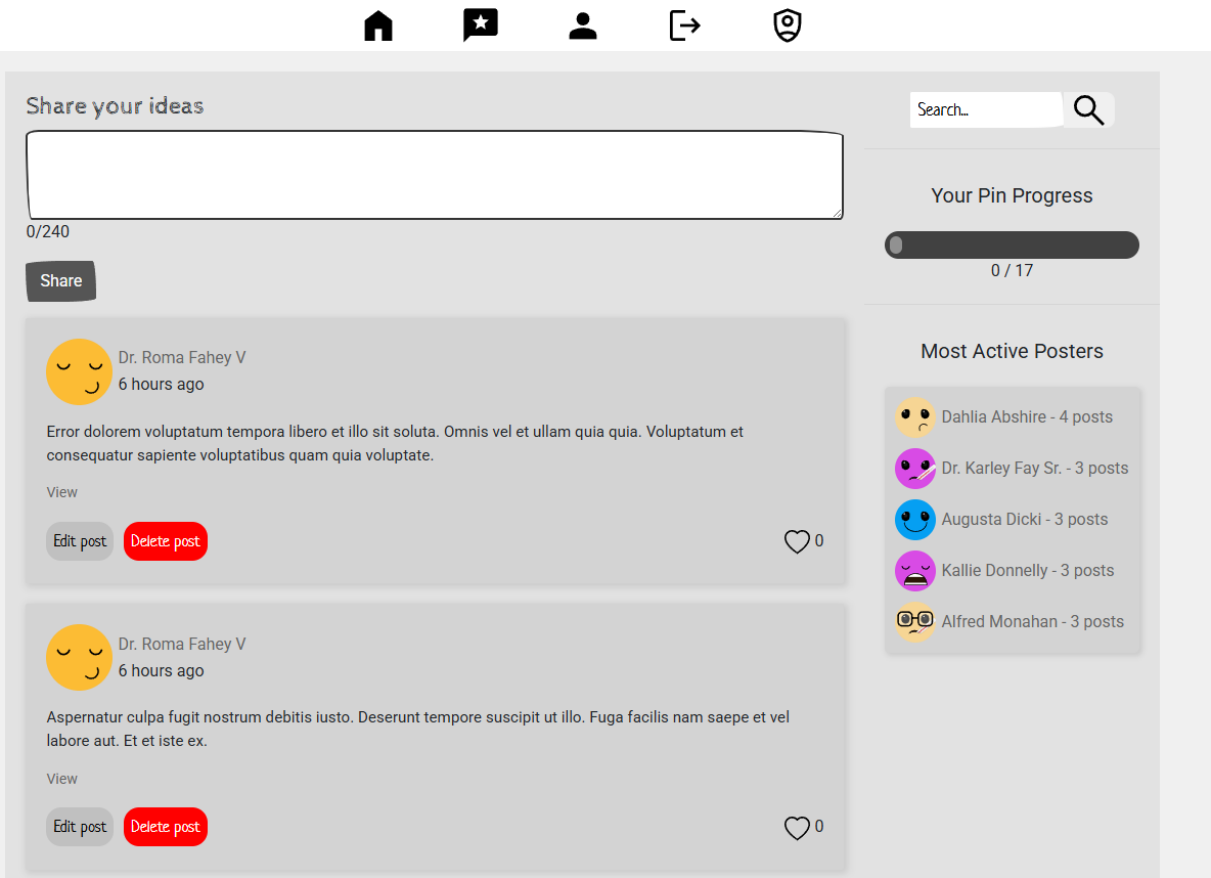
25. ábra Szövegszerkesztő felület profil módosítására

3.1.5. Feed weboldal

A feed oldal megegyező szerkezetű a főoldallal, azzal a különbséggel, hogy itt kizárólag a felhasználó által követett felhasználók posztjai jelennek meg

Ez az oldal a bejelentkezést követően válik elérhetővé a navigációs sávban, ahol a következő ikon megnyomására juthatunk el erre a weboldalra:

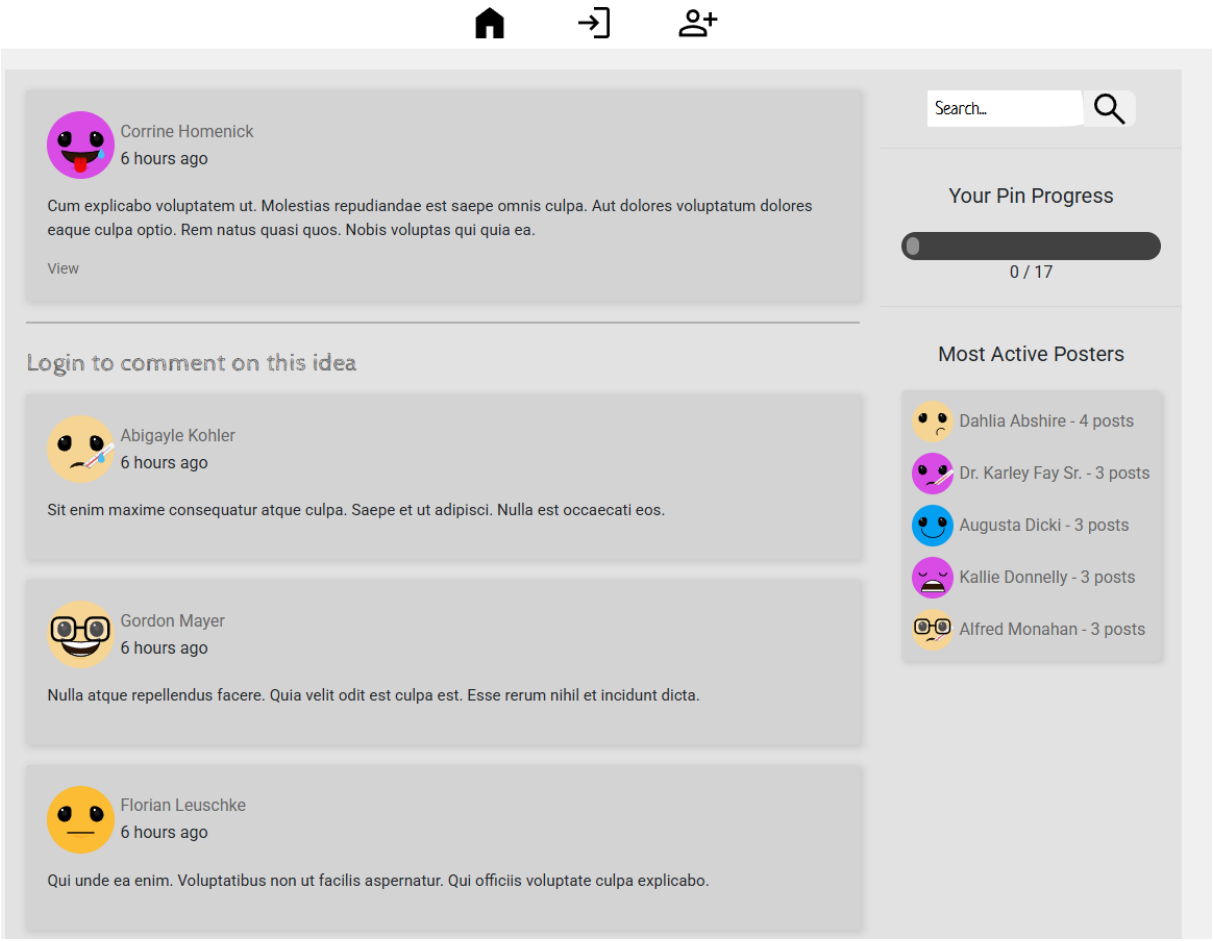
26. ábra Feed page navigációs ikon



27. ábra Feed page

3.1.6. Kommentek weboldal

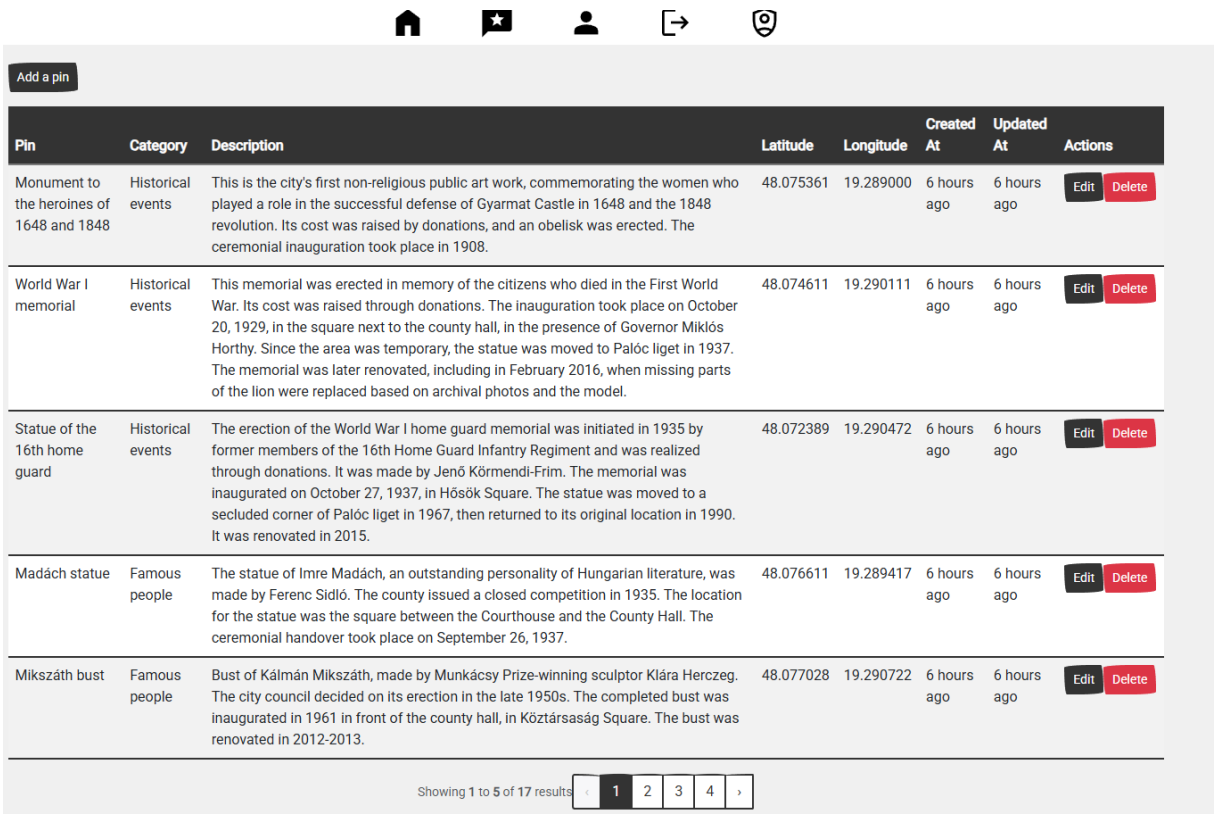
Amennyiben egy felhasználó rákattintott a “View” gombra egy poszt alatt, átirányul egy külön weboldalra, ahol megtekintheti a posztot a kommentjeivel együtt kilistázva. A poszt alatt található egy beviteli mező, mellyel megjegyzést lehet írni a posztra, ha viszont nem vagyunk bejelentkezve, a weboldal azt írja ki, hogy jelentkezzünk be megjegyzés írásához (“Login to comment on this post”). Amennyiben üres hozzászólás próbálunk elküldeni, a következő hibaüzenet jelenik meg: “The comment content field is required.”.



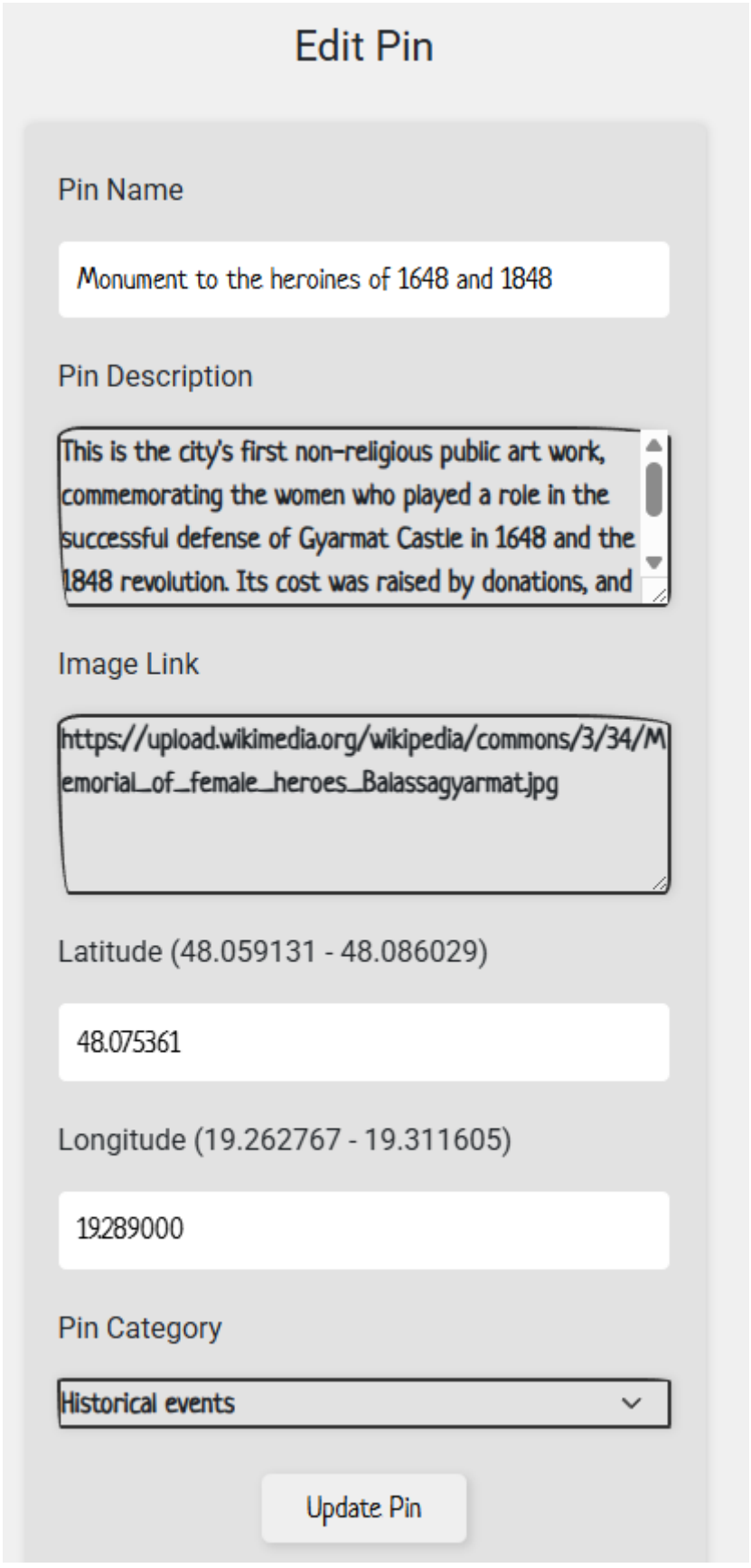
28. ábra Ötlet megtekintése kommentekkel együtt

3.1.7. Admin panel

Amennyiben egy fiók admin jogosultsággal rendelkezik, a navigációs menüben elérhetővé válik az “Admin” gomb. Erre rákattintva átirányít a weboldal az admin “dashboard” -ra. Itt kizárólag az adminisztrátorok számára megjelennek az adatbázis nevezetességei, melyeken változtathatnak, kitörölhetik őket, valamint további nevezetességeket is felvihetnek az adatbázisba.



29. ábra Admin panel



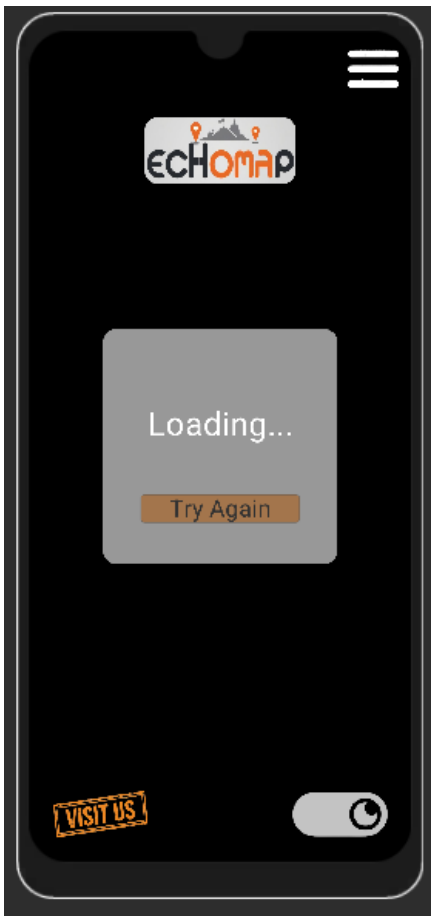
30. ábra Nevezetesség módosítása

3.2. Mobilalkalmazás

Az applikáció Unity-ben készült, és ennek köszönhetően egyaránt használható Apple és Androidos készülékeken is. Az applikáció az Echomap nevet kapta, és a felhasználók regisztrálás és bejelentkezés után használni tudják a térképet. Az applikáció szín dizájnja követi a weboldalét.

3.2.1. Főoldal

Miután a felhasználó elindította az applikációt, van egy gyors kis rövid ‘check’, ami megnézi, hogy csatlakozva van a felhasználó az internethez. Ha nincsen, akkor a probálja újra gomb kattintható lesz és miután rákattint, újra próbálkozik az alkalmazás.



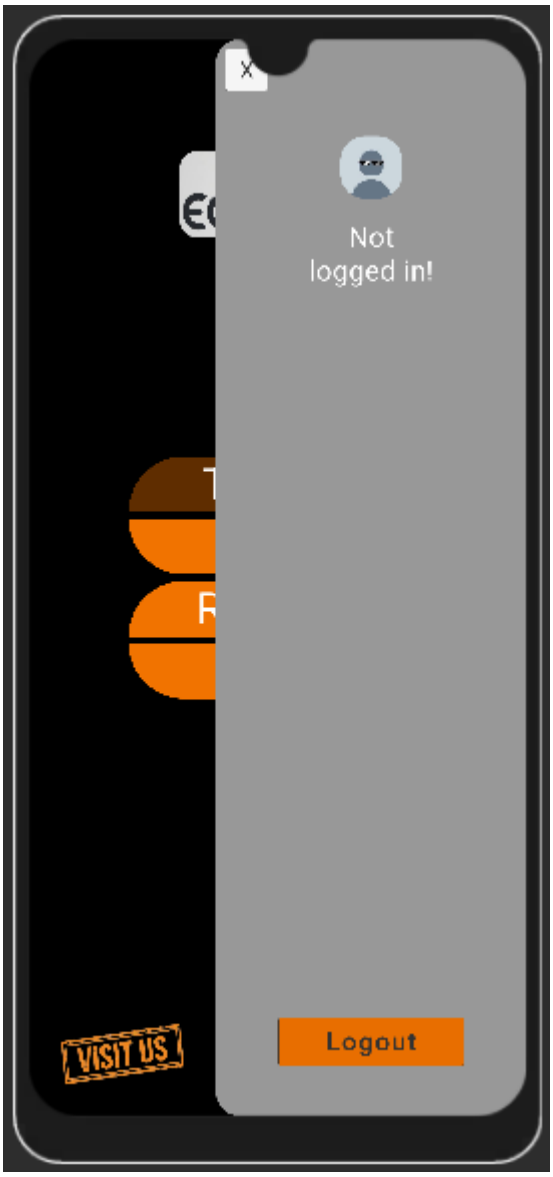
31. ábra Internetelérhetőség ellenőrzése

32. ábra Sikertelen internetelérhetőség ellenőrzés

Ha csatlakozva van az internethez akkor a főmenüben fogja magát találni. A felhasználó betudja állítani magának, hogy sötét vagy világos hátteret szeretne. A főoldalon látható a térkép, bejelentkezés, regisztrálás, kilépés, weboldalra jutás és a profil megtekintés gomb. Amíg nincsen bejelentkezve addig a térképre nem tud eljutni.

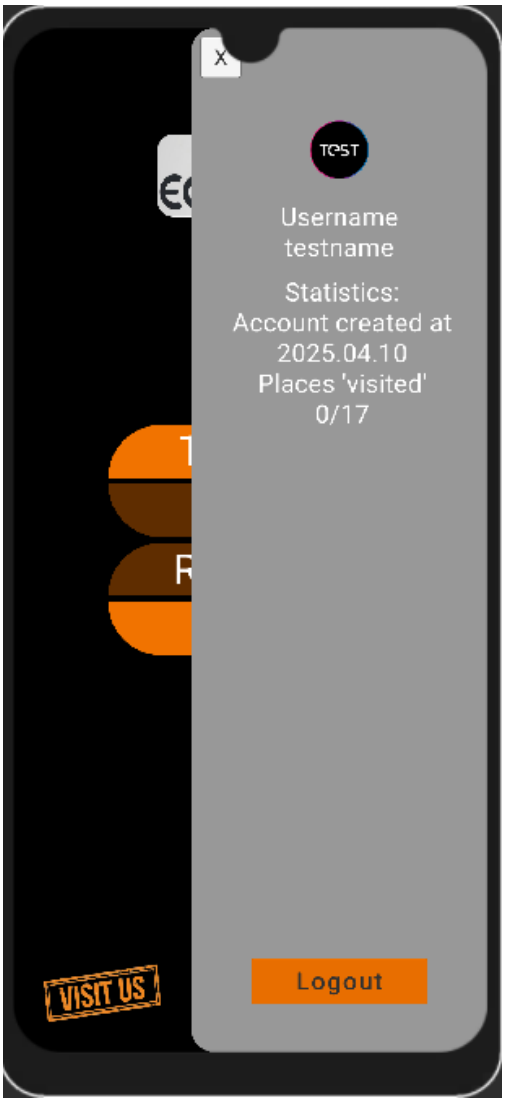
33. ábra Főoldal sötét mód

34. ábra Főoldal világos mód

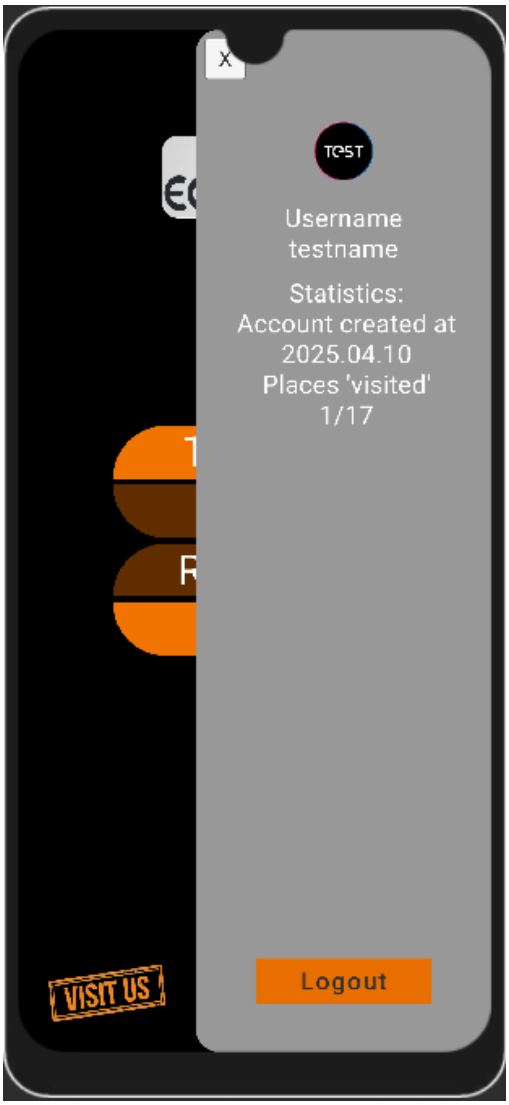


35. ábra Profil oldalmenü kijelentkezve

Miután be van jelentkezve, a profil gombra kattintva előjön egy oldal menü, ahol látható a felhasználó neve és pár adat a felhasználóról, mint például, hogy mikor lett készítve a felhasználó fiókja és hogy mennyi történelmi emléket tekintett meg az összes közül. Miután a felhasználó hozzáadja a meglátogatott történelmi emlékek közé valamelyik tűzőt a térképen keresztül, utána megtekinthető mennyit ‘látogatott meg’ az összes történelmi emlék közül.



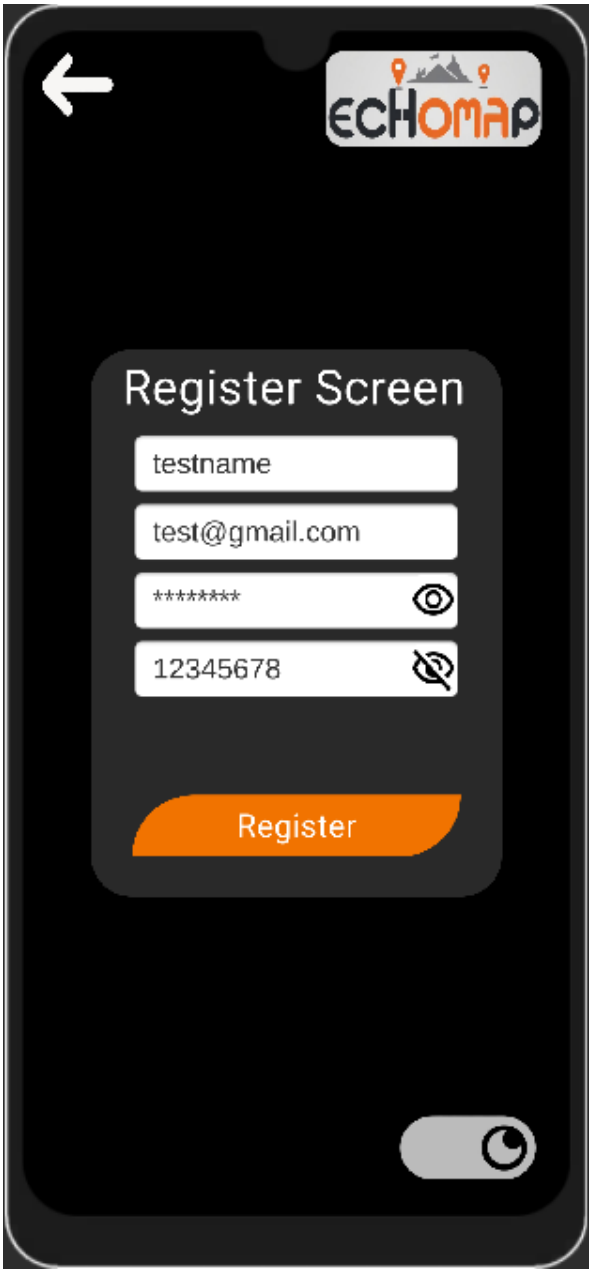
36. ábra Profil oldalmenü bejelentkezve 0 megtekintett tűzővel



37. ábra Profil oldalmenü bejelentkezve 1 megtekintett tűzővel

3.2.2. Regisztrálás

A felhasználó miután a regisztrálás gombra kattint, be kell írnia egy email címet, egy felhasználó nevet, a jelszavát kétszer. A jelszó megtekintésére található egy gomb mindkét részen. Ezután át dobja a belépés részre.



38. ábra Regisztrálás

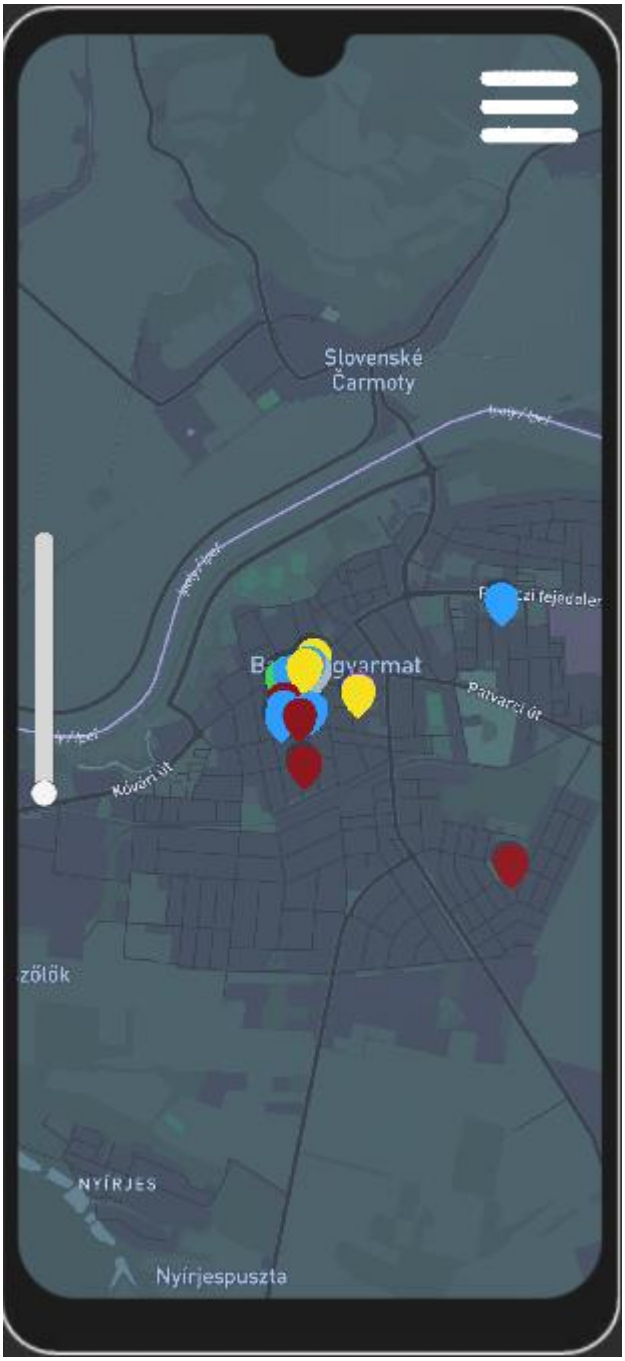
3.2.3. Belépés

A sikeres regisztrálás után, a bejelentkezés részen ismét be kell írnia az email címét és a jelszavát. Itt is található egy gomb a jelszó megtekintésére. Sikeres bejelentkezés után visszadobja a főmenüre.

39. ábra Bejelentkezés

3.2.4. Térkép

A térkép gombra kattintva eljut a felhasználó Balassagyarmat térképére, ahol láthatóak a történelmi emlékek helyein lévő tűzők. Mindegyik tűzőnek van egy különleges színe, ez indikálja, hogy milyen kategóriájú a történelmi emlék tűzője.



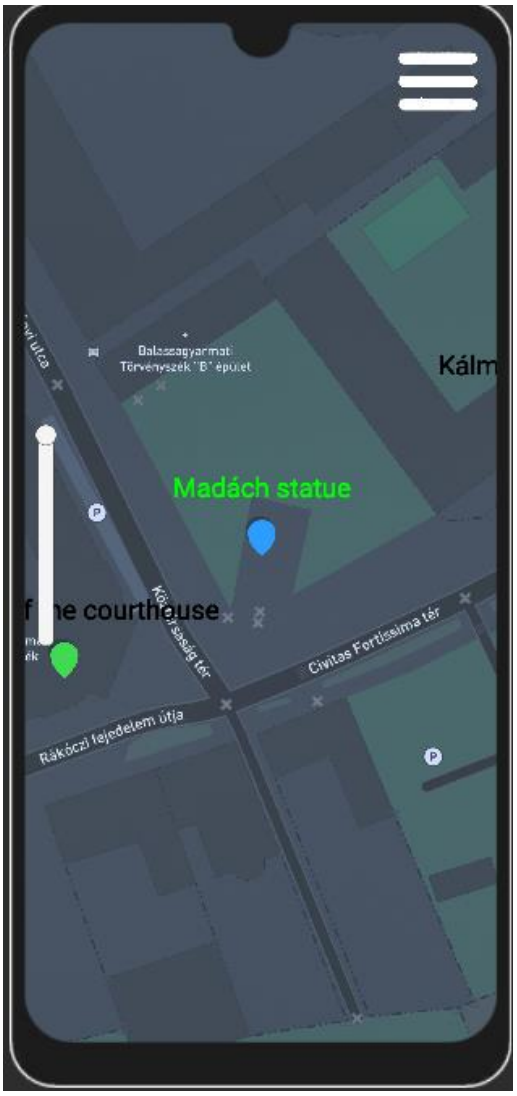
40. ábra Térkép

A bal oldalon található zoom slider segítségével a felhasználó nagyíthat a térképre, és amint teljesen ránagyít a térképre, akkor jelennek meg a tűzők felett a történelmi emlékek nevei.

41. ábra Ránagyított térkép

A tűzőre kattintva, feldob egy albakot, ami az adott történelmi emlékről több adatot is megmutat, mint például a név, leírás, koordináták és a kategóriája. Miután rákattint az OK gombra, hozzáadódik a tűző a meg látogatott történelmi emlékekhez.

42. ábra Tűző információ ablak



43. ábra Meglátogatott tűző

Jobb fent található egy gomb, aminek meg nyomásra jobb oldalról elő jön egy menü, ahol található a kategóriákról információ, amely meg mutatja melyik kategória, melyik színhez tartozik, egy tűző kereső és egy gomb, amivel vissza tud lépni a főmenübe a felhasználó.

44. ábra Térkép oldalmenü

Ebben a keresőben a felhasználó tud választani egy történelmi emléket, és miután rákattint, az oldal menü becsukódik és az választott történelmi emlék lesz a térkép/képernyő közepén.

45. ábra Térkép oldalmenü lenyitott keresővel

4. Fejlesztői dokumentáció

4.1. Webes felület

A választott fejlesztői platformunk egy Laravel nevű PHP keretrendszer. A Laravel egy ingyenes és nyílt forráskódú eszköz, ami a PHP-ra épül. Lehetővé teszi, hogy gyorsabban és jobban készítsünk el bármilyen PHP-val megvalósítható dolgot, ráadásul biztonságosabban. A Laravel nagymértékben felgyorsítja a fejlesztési időt, és könnyen lehet vele bonyolult alkalmazásokat is építeni.

A weboldal fejlesztése során közel kizárólag a keretrendszer dokumentációja elégséges volt, ahol ugyanis részletesen le van írva és meg van fogalmazva a funkciók meghatározó része. Nyomokban azonban szükség volt a Stackoverflow weboldalán megtalálható megoldások alapján létrehozni néhány funkciót.

A Laravel egy MVC (Model-View-Controller) nevű felépítést használ. Ez azt jelenti, hogy a programkódunk három külön részre van osztva: az adatok kezelésére (Model), a megjelenítésre (View), és a kettő közötti irányításra (Controller). A Laravel számos beépített funkcióval rendelkezik, amik megkönnyítik a modern webes alkalmazások fejlesztését, például az útvonalak kezelését, az adatbázis lekérdezéseket, a felhasználói bejelentkezést és regisztrációt, valamint a weboldal kinézetének (sablonok) kezelését.

A Laravel könnyen tanulható, moduláris, vagyis könnyen bővíthető, és egy nagy, segítőkész közösség is tartozik hozzá. A használatának előnyei közé tartozik a gyorsabb fejlesztés és a könnyebb karbantartás. A Laravel rendelkezik egy Artisan nevű parancssori eszközzel is, ami segít az ismétlődő feladatok gyors elvégzésében. A Laravel modellek segítségével pedig könnyebben tudjuk kezelni az adatbázisban tárolt adatokat.

4.1.1. Weboldal elérése

A webes felület elérhetősége a bdk.teamorange.hu oldalon keresztül valósul meg. A tartalom megjelenítéséhez egy webszerver szükséges, melynek kiválasztása a projekt igényeitől függ. Mi az Apache2 webszervert használjuk, mert projektünkre tekintve elégséges. A webalkalmazás saját szerveren fut. A domaint a https://www.rackhost.hu/ szolgáltatótól béreljük. A biztonságos kommunikáció érdekében HTTPS protokollt használunk, melyhez a CloudFlare proxy szolgáltatását vesszük igénybe. A keretrendszeren belül az AppServiceProvider.php boot() metódusában a következő kóddal érhetjük el a https kényszerítést:

```php
if (app()->environment('production')) {
    URL::forceScheme('https');
}
```

46. ábra HTTPS kényszerítés

4.1.2. A webalkalmazás üzemeltetésének menete

4.1.2.1. Szükséges technológiák

A Laravel keretrendszerrel fejlesztett webalkalmazások telepítéséhez és futtatásához PHP-ra, Composer-re és egy adatbázis-szerverre (például MySQL) van szükség. A .env fájlban konfigurálhatók a webalkalmazás alapvető beállításai, beleértve az adatbázis-kapcsolatot és az URL-t. A Composer telepítése után szükségünk van a következő parancs lefuttatására: composer global require laravel/installer.

Ez installálja a Laravel installert, mely lehetővé teszi a composer parancsok futtatását a projektben.

Először a projekt forráskódját le kell tölteni a Github weboldaláról, például a Git segítségével egy távoli tárolóból (repository). Ehhez ez a parancs futtatása szükséges a terminálban:

git clone https://github.com/ArpKevin/echomap_website.git

Ezután telepíteni kell a projekt PHP moduljait a Composer segítségével. Ezt a composer install parancs futtatásával lehet megtenni abban a könyvtárban, ahová a projekt forráskódja letöltésre került (a git clone https://github.com/ArpKevin/echomap_website.git parancs után akár közvetlen el tudunk menni a projekt mappájába a következő paranccsal: cd echomap_website). A Composer a projektben definiált modulokat tölti le.

4.1.2.2. Környezeti változók beállítása

A következő lépés a környezeti változók konfigurálása. A projekt tartalmaz egy .env.example nevű fájlt, amelyből létre kell hozni egy .env fájlt:

copy .env.example .env

Ebben a fájlban a webalkalmazás alapvető beállításait lehet megadni, mint például az adatbázis kapcsolódási adatai, alkalmazás neve, URL és egyéb konfigurációs paraméterek. A .env fájlban megadott beállításokat a keretrendszer beolvassa.

Ezután az adatbázis sémáját létre kell hozni az adatbázis-kezelő rendszerben. A Laravel projektekben ezt általában migrációk futtatásával lehet megtenni a php artisan migrate parancs

segítségével a projekt főkönyvtárából. A migrációk PHP fájlok, amelyek leírják az adatbázis tábláinak szerkezetét és a szükséges módosításokat. Ezek a migrációk a migrations/ mappában találhatóak

A webalkalmazások igényelnek egy alkalmazáskulcsot a biztonságos működéshez (például titkosításhoz). Ezt a kulcsot a php artisan key:generate paranccsal lehet generálni.

A .env fájlban beállíthatjuk a webalkalmazás adatbázisához való csatlakozás részleteit. Elsőként megadhatjuk, hogy milyen adatbázis csatlakozó driver-t használunk, ami a projektben MySQL-re van állítva (DB_CONNECTION=mysql). Ezt követően meg kell adnunk a szerver címét (DB_HOST), ami lokális fejlesztés esetén gyakran 127.0.0.1. Beállíthatjuk a portot is, ami MySQL esetében alapértelmezés szerint 3306 (DB_PORT=3306). A DB_DATABASE változóban adhatjuk meg az adatbázis nevét, amihez csatlakozni szeretnénk. Végül pedig meg kell adnunk az adatbázis bejelentkező adatait, a felhasználónevet (DB_USERNAME) és a jelszót (DB_PASSWORD). Ezekkel a beállításokkal konfiguráljuk a webalkalmazásunk adatbázis-kapcsolatát a .env fájlban.

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

47. ábra Adatbázis konfigurálása

A webalkalmazás elindításához lokális fejlesztői környezetben egy beépített fejlesztői szervert használnak, amelyet a php artisan serve paranccsal lehet elindítani. Ha XAMPP-ot használ a fejlesztői környezethez, győződjön meg róla, hogy az Apache webszerver el van indítva. Az éles környezetben az alkalmazást egy konfigurált webszervernek kell kiszolgálnia, amely lehet például az Apache, vagy a XAMPP által biztosított Apache. A .env fájlban beállított URL kulcsfontosságú a megfelelő működéshez.

```
APP_NAME=Ideas
APP_ENV=local
APP_KEY=base64:spYmztz3nKguRJ2XiqwaQQhLU3gPpl2KIXxwXlA+RWc=
APP_DEBUG=false
APP_TIMEZONE = "Europe/Budapest"
APP_URL=http://localhost
```

48. ábra Környezeti változók konfigurálása

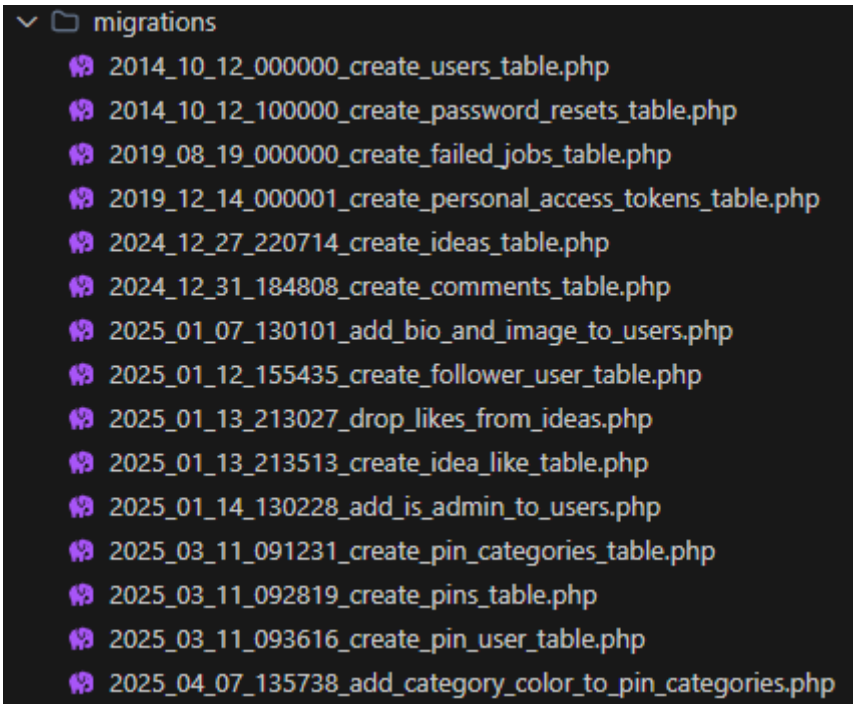
Az email küldésének beállításához konfigurálni kell az email szerver típusát, például SMTP-t (Simple Mail Transfer Protocol). Szükséges megadni a szerver címét, portját, a

bejelentkezési információkat, a titkosítási beállításokat, valamint a küldő email címét és a címzettet, ami alapértelmezetten az alkalmazás nevét tartalmazza.

```
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}"
```

49. ábra Mail server konfigurálása

Az adatbázisban található táblák létrehozásához és sémájának beállításához a migrációkat kell futtatnunk. Ezt a php artisan migrate paranccsal tehetjük meg a projektünk főkönyvtárában. A Laravel keretrendszerben a migrációk PHP kódként vannak megírva, amelyek SQL utasításokká alakulnak és kerülnek végrehajtásra az adatbázisban. A php artisan migrate parancs lefuttatja azokat a migrációs fájlokat, amelyek még nem lettek végrehajtva, létrehozva ezzel az adatbázis tábláit és azok szerkezetét. Sikeres futás esetén a rendszer visszajelzi, hogy minden migráció problémamentesen lefutott. Ha a parancsot újra futtatjuk, a Laravel ellenőrzi, hogy vannak-e új migrációk, és ha nincsenek, akkor jelzi, hogy nincs mit migrálni. Abban az esetben, ha hibák lépnek fel a migráció során, az valószínűleg a MySQL szerver nem megfelelő működésére vagy a konfigurációs beállítások hiányosságaira vezethető vissza. Létezik továbbá a php artisan migrate:refresh parancs is, amely először visszaállítja az összes korábbi migrációt, majd újra futtatja azokat. A migrációs fájlok létrehozásához a php artisan make:migration <migration_neve> parancs használható. Ezek a migrációk a migrations/ mappában találhatóak.



50. ábra Migrációk

A migrációk nyomon követéséhez a Laravel automatikusan létrehoz egy migrations nevű táblát az adatbázisban. Ez a tábla verziókezelőként működik: minden egyes migráció futtatásakor bejegyzést készít ebbe a táblába a migráció fájlnevével és egy batch számmal. A batch érték azt mutatja meg, hogy egy adott migráció melyik futtatási körhöz (batch-hez) tartozik. Például, ha egyszerre több migráció fut le, mindegyik ugyanazt a batch számot kapja. Ez lehetővé teszi, hogy később, például egy php artisan migrate:rollback parancs segítségével, egy teljes batch-nyi migrációt vissza tudjunk vonni.

		id	migration	batch
☐	Edit Copy Delete	1	2014_10_12_000000_create_users_table	1
☐	Edit Copy Delete	2	2014_10_12_100000_create_password_resets_table	1
☐	Edit Copy Delete	3	2019_08_19_000000_create_failed_jobs_table	1
☐	Edit Copy Delete	4	2019_12_14_000001_create_personal_access_tokens_ta...	1
☐	Edit Copy Delete	5	2024_12_27_220714_create_ideas_table	1
☐	Edit Copy Delete	6	2024_12_31_184808_create_comments_table	1
☐	Edit Copy Delete	7	2025_01_07_130101_add_bio_and_image_to_users	1
☐	Edit Copy Delete	8	2025_01_12_155435_create_follower_user_table	1
☐	Edit Copy Delete	9	2025_01_13_213027_drop_likes_from_ideas	1
☐	Edit Copy Delete	10	2025_01_13_213513_create_idea_like_table	1
☐	Edit Copy Delete	11	2025_01_14_130228_add_is_admin_to_users	1
☐	Edit Copy Delete	12	2025_03_11_091231_create_pin_categories_table	1
☐	Edit Copy Delete	13	2025_03_11_092819_create_pins_table	1
☐	Edit Copy Delete	14	2025_03_11_093616_create_pin_user_table	1
☐	Edit Copy Delete	15	2025_04_07_135738_add_category_color_to_pin_catego...	1

51. ábra Migrations tábla

4.1.3. Seeder-ek

Az adatbázis kezdeti adatokkal való feltöltéséhez seeder fájlokat használhatunk. A migrációk létrehozzák az adatbázis struktúráját, míg a seederek beviszik az elsődleges adatokat (például tesztfelhasználókat, alapértelmezett beállításokat).

Utólagos seedeléshez a projekt főkönyvtárában futtatható a php artisan db:seed parancs. Ez lefuttatja azokat a seeder osztályokat, amelyek a database/seeders mappában találhatók. A futtatandó seedereket a DatabaseSeeder.php fájlban lehet definiálni.

```php
public function run()
{
    // \App\Models\User::factory(10)->create();

    // \App\Models\User::factory()->create([
    //     'name' => 'Test User',
    //     'email' => 'test@example.com',
    // ]);
    $userSeedCount = 30;
    $ideaSeedCount = 50;
    $commentSeedCount = 200;
    $pinCategorySeedCount = 3;
    $pinSeedCount = 10;

    $users = User::factory()->count($userSeedCount)->create();

    $ideas = Idea::factory()->count($ideaSeedCount)->create([
        'user_id' => fn() => User::inRandomOrder()->first()->id
    ]);

    Comment::factory()->count($commentSeedCount)->create([
        'user_id' => fn() => User::inRandomOrder()->first()->id,
        'idea_id' => fn() => Idea::inRandomOrder()->first()->id
    ]);

    $pinCategories = PinCategory::factory()->count($pinCategorySeedCount)->create();

    $pins = Pin::factory()->count($pinSeedCount)->create([
        'pin_category_id' => fn() => $pinCategories->random()->id,
    ]);

    foreach ($users as $user) {
        // Attach 1 to 5 random pins to each user
        $user->pins()->attach(
            $pins->random(rand(1, 5))->pluck('id')->toArray()
        );
    }
```

52. ábra Seeder

A migrációk futtatásakor azonnal elindíthatjuk a seedereket is a --seed flag használatával. Tehát a `php artisan migrate --seed` parancs először lefuttatja a függőben lévő migrációkat, majd utána a seedereket. Ez különösen hasznos lehet éles környezetben vagy automatizált telepítési folyamatok során.

A seeder fájlok létrehozásához a `php artisan make:seeder <SeederNeve>` parancs használható.

4.1.4. Factory-k

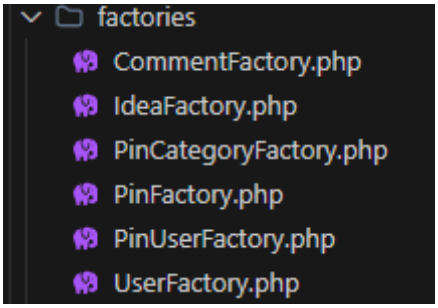
A factory-k arra szolgálnak, hogy teszteléshez szükséges adatbázis rekordokat hozzunk létre. Így nem szükséges manuálisan rekordokat felvinni, ezáltalán nagy mértékben megnő a tesztelés időbeli hatékonysága

Például a User::factory()->create(['email' => $email]); sor egy új felhasználói modell példány létrehozására utal. A factory() metódus egy factory osztályt hív meg (ebben az esetben a UserFactory-t), a create() metódus pedig létrehozza és elmenti az adatbázisba az új felhasználói rekordot a megadott e-mail címmel.

Egy másik példában a User::factory()->create(); sor egy felhasználói modell példányt hoz létre alapértelmezett attribútumokkal.

Tehát a Laravel factoryk kényelmes módot biztosítanak teszt adatok generálásához, lehetővé téve a fejlesztők számára, hogy egyszerűen hozzanak létre modellekhez tartozó adatbázis bejegyzéseket anélkül, hogy manuálisan kellene megadniuk az összes szükséges adatot minden egyes teszthez.

A factory definíciók határozzák meg, hogy egy adott modellhez milyen fake data kerül generálásra. Ezek a definíciók a projekt Database\Factories mappájában találhatóak.



53. ábra Factory-k

A factory osztályokon belül a definition() metódus tartalmazza a modell attribútumaihoz rendelt véletlenszerű értékeket.

Ezek az értékek a $this->faker tulajdonságon keresztül érhetőek el. A $this->faker egy FakerPHP példány, ami egy PHP könyvtár valósághű, de véletlenszerű adatok generálására.

Példaként nézzük meg újra a PinFactory definícióját:

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Pin>
 */
class PinFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            'pin_name' => $this->faker->sentence(rand(2, 3)),
            'pin_description' => $this->faker->text(200),
            'image_link' => $this->faker->imageUrl(100, 100),
            'latitude' => $this->faker->latitude(48.059131, 48.086029),
            'longitude' => $this->faker->longitude(19.262767, 19.311605),
        ];
    }
}
```

54. ábra PinFactory

Ebben a példában:

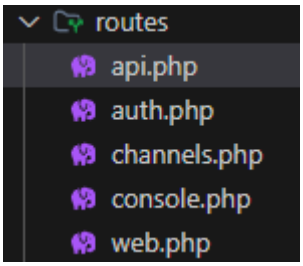
- A pin_name attribútumhoz a $this->faker->sentence(rand(2, 3)) egy véletlenszerű mondatot generál, amely 2 vagy 3 szóból áll.
- A pin_description attribútumhoz a $this->faker->text(200) egy 200 karakter hosszú véletlenszerű szöveget hoz létre.
- Az image_link attribútumhoz a $this->faker->imageUrl(100, 100) egy véletlenszerű kép URL-jét generálja, 100x100 pixel méretben.
- A latitude és longitude attribútumokhoz a $this->faker->latitude() és $this->faker->longitude() metódusok véletlenszerű földrajzi koordinátákat generálnak a megadott határokon belül. A generált koordináták kizárólag Balassagyarmat területén lehetnek

A fejlesztők a $this->faker objektum számos más metódusát is felhasználhatják, mint például nevek ($this->faker->name), e-mail címek ($this->faker->email), dátumok ($this->faker->date), és sok más valósághű formátumú adat generálására.

A seederek azután ezeket a factorykat használják fel az adatbázis kezdeti, teszt jellegű adatokkal való feltöltéséhez. A korábban említett php artisan db:seed paranccsal futtathatjuk a seedereket, amelyek a factoryk segítségével generált adatokat szúrják be az adatbázisba.

4.1.5. Route-ok

A weblap forgalomirányítása (routing) egy alapvető eljárás, mely kijelöli, hogy a felhasználói megkeresések miként jutnak el a megfelelő vezérlőegységekhez vagy eljárásokhoz egy internetes alkalmazáson belül. Így a forgalomirányító rendszer összekapcsolja a világhálós címeket (webes helyeket) a mögöttük rejlő tartalommal vagy működéssel. Ebben a keretrendszerben ez a folyamat különösen alakítható és egyszerűen kezelhető, lehetővé téve a fejlesztők számára, hogy könnyen irányítsák az alkalmazásuk áramlását és szabályozzák a címekhez rendelt műveleteket. Ebben a rendszerben a címkezelési utasítások általában a címek könyvtárában lévő állományokban találhatóak. Ez a könyvtár többféle címkezelő állományt is tartalmazhat, mint például a webes kérésekhez tartozó web.php és a programozási felület (API) kéréseihez tartozó api.php. Ezek a fájlok a routes/ mappában találhatóak.



55. ábra Route-ok

Ezekben az állományokban határozhatóak meg a címek és a hozzájuk tartozó működés. A kezdőoldalhoz tartozó cím egy GET típusú kérést jelöl ki, amikor beírjuk a böngésző címsorába, hogy http://bdk.teamorange.hu, akkor a böngésző a kiszolgálónak küld egy GET típusú kérést, a böngésző pedig megmutatja a dashboard.blade.php nevű oldalt. (A Controllerekről csak a későbbiekben esik szó.)

```
Route::get('/', [DashboardController::class, 'index'])->name('dashboard');
```

56. ábra "/" GET Route

app > Http > Controllers > DashboardController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Idea;
use App\Models\User;

class DashboardController extends Controller
{
    public function index()
    {
        $ideas = Idea::orderBy('created_at', 'desc');

        $ideas = $ideas->search(request()->search);

        return view("dashboard", [
            'ideas'=> $ideas->paginate(5)
        ]);
    }
}

```

57. ábra DashboardController

A Laravelben a routing alapvetően az a folyamat, amely meghatározza, hogy az HTTP kérések (például egy weboldal betöltésére irányuló kérés) hogyan kerülnek a megfelelő kódrészlethez az alkalmazásban. A Laravel lehetővé teszi, hogy útvonalakat (routes) definiáljunk, amelyek meghatározott URL-ekhez kapcsolódnak. Amikor egy felhasználó egy bizonyos URL-t látogat meg az alkalmazásban, a Laravel routing rendszere felelős azért, hogy az adott URL-hez definiált kód lefusson.

A route definíciók általában a routes könyvtárban található fájlokban helyezkednek el. A leggyakrabban használt fájl a web.php, amely a webes felülethez tartozó útvonalakat tartalmazza. Létezhetnek más route fájlok is, például az api.php az API végpontokhoz, vagy akár külön fájlok a különböző funkcionális területekhez.

A web.php fájlban definiálhatjuk az alkalmazás által kezelt különböző HTTP kéréseket. A leggyakoribb HTTP metódus a GET, amelyet a weboldalak és más erőforrások lekérésére használunk. Például a következő kódsor: `Route::get('/', function () { return view('welcome'); });` egy GET kérést definiál a weboldal gyökeréhez (/), és megadja, hogy erre a kérésre a welcome nézetet (view) kell visszaadni. A view() függvény a resources/views mappában keresi a megadott nevű .blade.php kiterjesztésű fájlt és azt jeleníti meg a felhasználónak.

Az útvonalakhoz közvetlenül nézeteket rendelhetünk egy closure (névtelen függvény) segítségével, ahogy a példában is látható. Ez egyszerű esetekben, például statikus oldalak megjelenítésére elegendő. Azonban bonyolultabb logika esetén, amikor adatbázisból kell lekérni információkat vagy valamilyen feldolgozást kell végezni, az útvonalakat gyakran kontrollerekhez kötjük. A kontrollerek PHP osztályok, amelyek metódusai tartalmazzák az adott útvonalhoz tartozó alkalmazáslogikát.

A web.php fájlban különböző útvonalakat definiálhatunk az alkalmazás különböző oldalaihoz. Például definiálhatunk útvonalat egy profiloldalhoz (/profile), egy hírfolyamhoz (/feed), vagy egy felhasználói oldalhoz (/users).

A Laravelben lehetőségünk van útvonalak elnevezésére is a name() metódus segítségével. A névvel ellátott útvonalak megkönnyítik az útvonalakra való hivatkozást a kódban, különösen a Blade sablonokban a route() segédfüggvény használatával. Ezáltal elkerülhetjük az útvonalak "beégetését" a kódba, és ha megváltozik egy útvonal URL-je, nem kell az egész alkalmazásban átírni a hivatkozásokat.

```
Route::post('users/{user}/follow', [FollowerController::class, 'follow'])->middleware('auth')->name
('users.follow');
```

58. ábra Route model binding I.

```
<form action="{{ route('users.follow', $user->id) }}" method="post">
    @csrf
    <button type="submit" class="btn btn-primary btn-sm"> Follow </button>
</form>
```

59. ábra Route model binding II.

A Laravel lehetővé teszi az útvonalak csoportosítását a group() metódus segítségével, amely hasznos lehet például, ha közös jellemzőkkel rendelkező útvonalakat szeretnénk kezelni, mint például a közös URL prefix, név vagy más beállítások.

Például az alábbi kód esetén a prefix és as attribútumok segítségével az ideas útvonalakat csoportosítjuk, hogy könnyebben kezelhetőek legyenek. A group() metódus segítségével könnyedén alkalmazhatunk közös jellemzőket a hozzájuk tartozó útvonalakra.

```php
Route::prefix('ideas/')
    ->as('ideas.')
    ->group(function () {
        Route::get('/{idea}', [IdeaController::class, 'show'])->name('show');

        Route::middleware('auth')->group(function () {
            Route::get('/{idea}/edit', [IdeaController::class, 'edit'])->name('edit');
            Route::post('', [IdeaController::class, 'store'])->name('store');
            Route::put('/{idea}', [IdeaController::class, 'update'])->name('update');
            Route::delete('/{idea}', [IdeaController::class, 'destroy'])->name('destroy');

            Route::post('/{idea}/comments', [CommentController::class, 'store'])->name('comments.
            store');
        });
    });
```

60. ábra Csoportosított route-ok

Ez lehetőséget ad arra, hogy a hasonló funkciókat tartalmazó útvonalakat logikusan rendszerezzük, és így a kódunk karbantartása egyszerűbbé válik.

A Laravel Route::resource() metódus egy rendkívül hasznos és kényelmes módja annak, hogy gyorsan és egyszerűen létrehozzunk az adott erőforráshoz (resource) tartozó alapvető CRUD (Create, Read, Update, Delete) műveleteket. Amikor ezt a metódust használjuk, Laravel automatikusan létrehozza a megfelelő útvonalakat és azokat az függvényeket, amelyek egy adott erőforrás (resource) kezeléséhez szükségesek.

Például az alábbi kódban a Route::resource() metódus használatával létrehozhatjuk az ideas erőforráshoz tartozó alapvető műveletekhez szükséges útvonalakat:

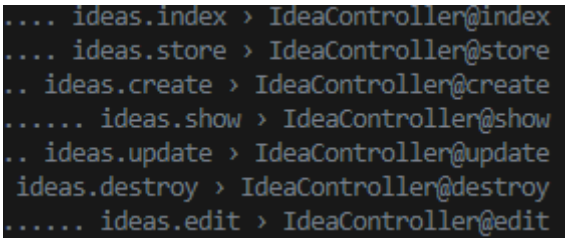
```php
Route::resource('ideas', IdeaController::class);
```

61. ábra Resource routing

Ez az egyetlen sor automatikusan generálja a következő útvonalakat és az azokhoz tartozó controller függvényeket:

```
GET|HEAD        ideas ..............
POST            ideas ..............
GET|HEAD        ideas/create .......
GET|HEAD        ideas/{idea} .......
PUT|PATCH       ideas/{idea} .......
DELETE          ideas/{idea} .......
GET|HEAD        ideas/{idea}/edit .
```

62. ábra Resource route-ok I.



63. ábra Resource route-ok II.

Ez tehát jelentősen csökkenti az útvonalak és a controller akciók írásához szükséges kód mennyiségét. A Route::resource() metódus automatizálja azokat a műveleteket, amelyeket gyakran alkalmazunk, így ahelyett, hogy minden egyes útvonalat külön-külön definiálnánk, elegendő csak egyet írni, és Laravel elvégzi a többit.

Összességében a Laravel routing rendszere egy rugalmas és hatékony eszköz a webalkalmazások forgalmának kezelésére. A jól definiált útvonalak, a kontrollerek használata és a middleware (melyekről később részletesebben lesz szó) integráció hozzájárulnak egy tiszta és karbantartható kódalap kialakításához.

4.1.6. Route Model Binding

A route model binding egy speciális Laravel koncepció, amely segít lerövidíteni a kódot és javítani annak minőségét. Korábban, amikor például törölni szerettünk volna egy ötletet, le kellett kérnünk az adatbázisból az adott ötletet azonosító alapján, majd meghívni a delete függvényt rajta. A Laravel egy kényelmesebb módszert kínál erre a route model binding segítségével.

A route model binding működése nagyon egyszerű. Mindössze annyit kell tennünk, hogy megmondjuk a Laravelnek, hogy egy adott útvonalparaméter (ID) melyik modellhez tartozik. Ezt úgy tehetjük meg, hogy a route definícióban vagy a kontroller metódusának paraméterlistájában típusdeklaráljuk a modellt. A Laravel háttérfolyamatok segítségével automatikusan kikövetkezteti, hogy ez az azonosító a megadott modell elsődleges kulcsa. Így a Laravel ezt a kódrészletet a háttérben elvégzi helyettünk. Ez lehetővé teszi számunkra, hogy eltávolítsuk vagy lerövidítsük ezt a kódot.

```php
Route::delete("ideas/{idea}", [IdeaController::class, 'destroy'])->name('ideas.destroy');
```

64. ábra Route model binding III.

```php
public function destroy(Idea $idea)
{
    $idea->delete();
}
```

65. ábra Route model binding IV.

Például a példa szerint, amikor törölni szeretnénk egy ötletet, a kontroller destroy metódusában típus deklarálhatjuk az Idea modellt a paraméterként átvett $idea változóval. A Laravel automatikusan lekérdezi az ID-nak megfelelő Idea modellpéldányt.

4.1.7. Middleware

A middleware a Laravelben olyan köztes rétegek, amelyek lehetővé teszik, hogy logikát hajtsunk végre a webalkalmazásunkba érkező HTTP kérések feldolgozása előtt vagy után. A middleware célja a kérések manipulálása és feldolgozása a keretrendszerben.

A Laravel már számos beépített middleware-rel rendelkezik, például azzal, amely átirányítja a felhasználót, ha nincs hitelesítve.

A middleware-ek regisztrálása és konfigurálása a kernel.php fájlban történik. Ez a fájl tartalmazza a következő tulajdonságokat:

- $middleware: Itt definiálhatjuk a globális middleware-eket, amelyek minden egyes útvonalra alkalmazásra kerülnek az alkalmazásban. Ilyen például a kérések megakadályozása karbantartás közben vagy a posztméret validálása.

```php
/**
 * The application's global HTTP middleware stack.
 *
 * These middleware are run during every request to your application.
 *
 * @var array<int, class-string|string>
 */
protected $middleware = [
    // \App\Http\Middleware\TrustHosts::class,
    \App\Http\Middleware\TrustProxies::class,
    \Illuminate\Http\Middleware\HandleCors::class,
    \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
    \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
    \App\Http\Middleware\TrimStrings::class,
    \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
];
```

66. ábra Middleware I.

- $middlewareGroups: Ez lehetővé teszi több middleware csoportosítását. A Laravel alapértelmezés szerint tartalmaz web és api middleware csoportokat. Ha egy middleware csoportot alkalmazunk egy útvonalra, akkor a csoportban lévő összes middleware is lefut.

```php
/**
 * The application's route middleware groups.
 *
 * @var array<string, array<int, class-string|string>>
 */
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        // \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        // \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
        'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];
```

67. ábra Middleware II.

- $routedMiddleware: Ezek a middleware-ek közvetlenül az egyes útvonalakhoz vagy útvonalcsoportokhoz vannak rendelve. Ahelyett, hogy globálisan vagy csoportosan alkalmaznánk őket, itt konkrét route-ok szintjén adhatjuk meg, hogy mely middleware-ek fussanak le. Ez lehetővé teszi a finomhangolást, például egy adott útvonal csak akkor legyen elérhető, ha egyedi jogosultsági middleware teljesül.

```php
/**
 * The application's route middleware.
 *
 * These middleware may be assigned to groups or used individually.
 *
 * @var array<string, class-string|string>
 */
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
];
```

68. ábra Middleware III.

```php
Route::resource('ideas', IdeaController::class)->except('index','create','show')->middleware('auth');
Route::resource('ideas', IdeaController::class)->only('show');

Route::resource('ideas.comments', CommentController::class)->only('store')->middleware('auth');

Route::resource('users', UserController::class)->only('show', 'edit', 'update')->middleware('auth');
Route::resource('users', UserController::class)->only('show');
```

69. ábra Middleware egy route-on

A middleware-eket egyéni útvonalakra vagy útvonalcsoportokra alkalmazhatjuk. Az alkalmazás módja a middleware() metódus használatával történik az útvonal definíciójánál.

A middleware lényegében lehetővé teszi a kérések "elfogását" mielőtt az alkalmazás tényleges kódja lefutna. A handle metódusban definiált logika dönti el, hogy a kérés továbbhaladhat-e a következő middleware-hez vagy a tényleges útvonalhoz. A $next($request) hívása a middleware-ben elengedhetetlen a kérés láncban való továbblépéséhez. A middleware-eket az útvonalaknál határozzuk meg, hogy mely útvonalakat szeretnénk ellenőrzés alá vonni.

```php
class RedirectIfAuthenticated
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)  $next
     * @param  string|null  ...$guards
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     */
    public function handle(Request $request, Closure $next, ...$guards)
    {
        $guards = empty($guards) ? [null] : $guards;

        foreach ($guards as $guard) {
            if (Auth::guard($guard)->check()) {
                return redirect(RouteServiceProvider::HOME);
            }
        }

        return $next($request);
    }
}
```

70. ábra Middleware osztálya

4.1.8. Kontrollerek

A kontrollerek a Laravel alkalmazásokban a Model-View-Controller (MVC) architektúrán alapuló minta legfelső szintjét képviselik, és kulcsszerepet töltenek be az alkalmazás logikájának szervezésében. A kontrollerek feladata, hogy közvetítsenek a modellek (adatbázis-kezelés, róluk később) és a view fájlok (felhasználói felület) között, valamint, hogy diktálják az alkalmazás folyamatát. Amikor egy HTTP kérés érkezik az alkalmazásba, a Laravel routing rendszere határozza meg, hogy melyik kontrollernek és melyik metódusának kell a kérést kezelnie.

A kontrollerek PHP osztályok, amelyeket a app/Http/Controllers könyvtárban helyezünk el. Új kontrollereket az Artisan parancssori eszköz segítségével hozhatunk létre a

php artisan make:controller <KontrollerNeve> paranccsal. Ezek az osztályok kiterjesztik a Laravel által biztosított Controller alaposztályt, így öröklik annak alapvető funkcionalitását.

A kontrollerekben definiált metódusok felelősek a beérkező HTTP kérések különböző típusainak (GET, POST, PUT, DELETE) kezeléséért. A routing rendszerben útvonalakat rendelünk hozzájuk, amelyek meghatározzák, hogy egy adott URL kérés esetén melyik kontroller melyik metódusa fusson le. A kontrollerek metódusai fogadják a beérkező kéréseket (gyakran a Request objektum formájában), interakcióba léphetnek a modellekkel az adatok lekéréséhez vagy módosításához, előkészíthetik az adatokat a nézetek számára, és végül HTTP válaszokat generálhatnak.

A kontrollerek által visszaadott válaszok sokfélék lehetnek. Gyakran nézeteket (Blade sablonokat) adnak vissza a view() helper függvénnyel, amelyek megjelenítik a felhasználói felületet. Emellett képesek JSON formátumú adatokat visszaadni API kérések esetén, vagy átirányításokat végrehajtani más útvonalakra a redirect() helper vagy a Redirect facade segítségével.

A kontrollerek fontos szerepet játszanak a kód karbantarthatóságának és strukturáltságának javításában. Ahelyett, hogy az összes alkalmazáslogikát az útvonalak definícióiban vagy a nézetekben helyeznénk el, a kontrollerek lehetővé teszik a logika központi helyen történő kezelését. Ezáltal a kód átláthatóbbá válik, könnyebben tesztelhető és bővíthető.

A kontrollerek a middleware-ekkel együttműködve biztosíthatják az alkalmazás biztonságát és működését. A middleware-ek a kérések feldolgozása előtt vagy után futnak le, és lehetővé teszik például a hitelesítés (auth()->check()) vagy a jogosultságkezelés (can, Gates, Policies, ezekről később) megvalósítását a kontroller metódusainak meghívása előtt. A projektünkben például egy Gate ellenőrzi, hogy a felhasználó rendelkezik-e adminisztrátori jogosultságokkal, mielőtt hozzáférne az admin oldalhoz, továbbá 2 további Gate biztosítja azt, hogy a felhasználók csak a saját posztjaikat tudják törölni és módosítani.

```
0 references | 0 overrides
public function boot(): void
{
    // Role
    Gate::define(ability: 'admin', callback: function(User $user) : bool{
        return $user->is_admin;
    });
    // Permission
    Gate::define(ability: 'idea.delete', callback: function(User $user, Idea $idea) : bool{
        return $user->is_admin || $user->id === $idea->user_id;
    });
    Gate::define(ability: 'idea.edit', callback: function(User $user, Idea $idea) : bool{
        return $user->is_admin || $user->id === $idea->user_id;
    });
}
```

71. ábra Gate-ek

A Laravelben létezik egy új koncepció, az úgynevezett invokable kontrollerek. Ezek olyan speciális kontrollerek, amelyek csak egyetlen műveletet hajtanak végre. Létrehozásukhoz a php artisan make:controller <KontrollerNeve> parancsot használjuk a --invokable opcióval. Az invokable kontrollerekben egyetlen __invoke metódus található, ami egyszerűsíti a dolgokat, ha egyetlen feladatot szeretnénk egy adott útvonalhoz kötni.

```
1   <?php
2
3   namespace App\Http\Controllers;
4
5   class ProvisionServer extends Controller
6   {
7       /**
8        * Provision a new web server.
9        */
10      public function __invoke()
11      {
12          // ...
13      }
14  }
```

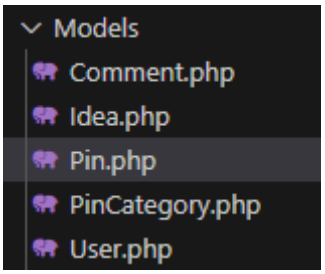
72. ábra Invokable kontroller

4.1.9. Modellek

A modellek olyan PHP osztályok a Laravelben, amelyek az adatbázis táblák objektumorientált reprezentációi. Segítségükkel a kód egyszerűsíthetővé és jobban karbantarthatóvá válik. A modellek objektumokban tárolják az adatokat, ami megkönnyíti azok

kezelését és manipulálását. A modellek elrejtik az adatbázis-specifikus kódot, ezáltal a kód hordozhatóbbá válik. Emellett a modellek segítségével könnyebben lehet a bemeneti adatokat validálni.

Egy új modellt a php artisan make:model <ModelNeve> paranccsal hozhatunk létre. A parancsnak meg kell adni a modell nevét, és ez létrehoz egy új fájlt az app/Models mappában.



73. ábra Modellek

Ez a fájl tartalmazza a modell osztályát, ahol definiálhatók a modell mezői, kapcsolatai és validációs szabályai.

A modell osztályok a Laravelben az Illuminate\Database\Eloquent\Model osztályt terjesztik ki. Ez az alap modell osztály számos hasznos funkciót tartalmaz, például táblával kapcsolatos információkat (pl. $table), amivel meg tudjuk határozni, hogy melyik tábla struktúráját és adatait képviselje az osztály, elsődleges kulcsot ($primaryKey), amivel meghatározzuk a tábla azonosítóját, hogy azt továbbá sokkal gördülékenyebben lehessen a táblával dolgozni, például egy Pin nevű modellnél a Pin::find(azonosító) függvénnyel megkaphatjuk értéknek azt a rekordot, amelyik ezzel az azonosítóval rendelkezik, továbbá sok egyéb metódus áll rendelkezésünkre, mint például az all() metódus, amely lekéri az összes rekordot a táblából.

4.1.9.1.ORM (Object-Relational Mapping)

A Laravel modellekhez kapcsolódó beépített függvények jelentősen növelik a fejlesztési hatékonyságot és biztonságot. Ezek a függvények az Eloquent ORM (Object-Relational Mapping) részei, amely lehetővé teszi, hogy objektumorientált módon, tiszta és olvasható PHP kóddal manipuláljuk az adatbázis rekordokat. Ahelyett, hogy kézzel írnánk SQL lekérdezéseket, egyszerű metódushívásokkal érhetünk el olyan funkciókat, mint például rekordok lekérdezése, szűrése, beszúrása, frissítése vagy törlése.

A Laravel automatikusan lefordítja ezeket a metódusokat hatékony és optimalizált SQL lekérdezésekké a háttérben, így a fejlesztőnek nem kell közvetlenül SQL nyelvet használnia. Ez nemcsak gyorsítja a fejlesztést, hanem csökkenti a hibalehetőségek számát is.

A biztonság szempontjából is előnyös megoldásról van szó: a Laravel ORM védelmet nyújt az SQL injekciós támadások ellen azáltal, hogy automatikusan gondoskodik a bemeneti

adatok helyes escape-eléséről és tisztításáról. A Laravel saját eszközeivel biztonságos módon kezeli az adatbázis-interakciókat, például paraméterezett lekérdezéseken keresztül.

Itt van például egy Csoki nevű tábla modelljének a Kontrollere, ahogy láthatjuk, néhány sor kódból egy teljes REST API-t képesek vagyunk leprogramozni a keretrendszer jóvoltából. Ez a kódrészlet natív PHP programozási nyelvben sokkal több időt és energiát venne igénybe.

```php
public function index()
{
    return Csoki::all();
}

/**
 * Store a newly created resource in storage.
 */
public function store(Request $request)
{
    $request->validate(['nev' => 'required', 'ara' => 'required', 'raktaron' => 'required']);

    return Csoki::create($request->except('id'));
}

/**
 * Display the specified resource.
 */
public function show(string $id)
{
    return Csoki::findOrFail($id);
}

/**
 * Update the specified resource in storage.
 */
public function update(Request $request, string $id)
{
    $request->validate(['nev' => 'required', 'ara' => 'required', 'raktaron' => 'required']);

    return Csoki::findOrFail($id)->update($request->except('id'));
}

/**
 * Remove the specified resource from storage.
 */
public function destroy(string $id)
{
    return Csoki::findOrFail($id)->delete();
}
```

74. ábra REST API Laravelben

4.1.9.2.$fillable és $guarded

A modell mezőit a $fillable és $guarded tulajdonságok segítségével definiálhatjuk. A $fillable tulajdonságban adhatjuk meg azokat a mezőket, amelyeket tömegesen hozzárendelhetővé szeretnénk tenni (mass assignment). Ez azt jelenti, hogy ezeket a mezőket a

create() vagy update() metódusok segítségével egyszerre lehet feltölteni. A $guarded tulajdonságban pedig azokat a mezőket adhatjuk meg, amelyeket nem szeretnénk tömegesen hozzárendelni. Ide általában generált értékek (pl. jelszó) kerülnek.

```php
/**
 * The attributes that are mass assignable.
 *
 * @var array<int, string>
 */
protected $fillable = [
    'name',
    'email',
    'password',
    "bio",
    "image",
];
```

75. ábra $fillable

```php
public function store(){
    $validated = request()->validate([
        'name'=>'required|min:3|max:40',
        'email'=> 'required|email|unique:users,email',
        'password'=> 'required|confirmed|min:8'
    ]);
    if($validated){
        $user = User::create([
            'name'=> $validated['name'],
            'email'=> $validated['email'],
            'password'=> Hash::make($validated['password']),
        ]);

        return redirect()->route('login')->with('success','Account created successfully');
    }
```

76. ábra Mass assignment

4.1.10. Nézetek (View)

A nézetek (view) kényelmes módot nyújtanak az összes HTML kód külön fájlokban történő elhelyezésére a Laravel alkalmazásokban. Nem praktikus teljes HTML dokumentumokat közvetlenül a route-okból és kontrollerekből visszaadni.

A nézetfájlok általában a resources/views könyvtárban találhatók.

A Blade sablonok .blade.php kiterjesztéssel rendelkeznek. A resources/views mappában tároljuk az összes HTML-ünket, és ahogy a név is sugallja, ez az, ami a felhasználó számára látható.

A nézeteket a globális view() helper függvénnyel lehet visszaadni a route-okból és a kontrollerekből. A view() helper első argumentuma a nézetfájl neve a resources/views

könyvtárban. A view() függvény második argumentuma egy asszociatív tömb, amellyel adatokat adhatunk át a nézetnek.

Alternatív megoldásként a with() metódus is használható egyedi adatok hozzáadására a nézethez, mielőtt az visszaadásra kerül. A with() metódus a nézet objektumának egy példányát adja vissza, így a metódusok láncolhatók. Például: return view('greeting')->with('name', 'Victoria')->with('occupation', 'Astronaut');.

A compact() egy PHP-s segédfüggvény, ami asszociatív tömböt készít a megadott változó(k)ból úgy, hogy a változó neve lesz a kulcs, és az értéke a változó értéke. A compact()-ot gyakran használjuk a view() függvénnyel, hogy a controllerből egyszerűen adjunk át adatokat a view-nak.

```
return view("profile", compact("user"));
```

Ez a sor kód ugyanazt csinálja, mint ez:

```
return view("profile", ["user" => $user]);
```

A Laravel elég okos ahhoz, hogy automatikusan megtalálja a nézetfájlt a megadott név alapján, anélkül, hogy a .blade.php kiterjesztést meg kellene adni.

A beágyazott nézetkönyvtárak eléréséhez pontszintaxist használhatunk. Például egy resources/views/admin/profile.blade.php fájlt a 'admin.profile' névvel hivatkozhatunk.

Időnként szükség lehet arra, hogy adatokat osszunk meg az alkalmazás által renderelt összes nézettel. Ezt a View facade share() metódusával tehetjük meg. A share() metódus hívásait általában egy szolgáltató (ServiceProvider) boot() metódusában kell elhelyezni. Hozzáadhatjuk őket az App\Providers\AppServiceProvider osztályhoz, vagy létrehozhatunk egy külön szolgáltatót a tárolásukhoz. Például: View::share('key', 'value');.

A Nézet Komponensek (View Composers) lehetővé teszik adatok kötését a nézetekhez, mielőtt azok renderelésre kerülnének. A nézetkomponensek a szolgáltatói konténeren keresztül kerülnek feloldásra, így a függőségek bevihetők a komponens konstruktorába. A View facade composer() metódusával regisztrálhatjuk a nézetkomponenseket. A komponenseket osztály alapú komponensekként vagy closure-ök segítségével definiálhatjuk. Egy komponenshez tartozó osztálynak rendelkeznie kell egy compose() metódussal, amely megkapja a View példányát és a with() metódus segítségével köti az adatokat a nézethez. Egy nézetkomponens több nézethez is csatolható egy nézettömb átadásával a composer() metódus első argumentumaként. A '*' karakter helyettesítőként is használható, amely lehetővé teszi egy komponens csatolását az összes nézethez.

```php
namespace App\View\Composers;

use Illuminate\View\View;
use App\Models\User;

class TopUserComposer
{
    /**
     * Bind data to the view.
     */
    public function compose(View $view): void
    {
        $view->with('topUsers', User::withCount('ideas')->orderBy('ideas_count', 'desc')->take(5)->get());
    }
}
```

77. ábra View Composer

```php
public function boot()
{
    Paginator::useBootstrapFive();

    if (app()->environment('production')) {
        URL::forceScheme('https');
    }

    View::composer(["*"], TopUserComposer::class);
}
```

78. ábra View Composer regisztrálása

4.1.11. Direktívák

A Laravel Blade sablonmotor számos kényelmes direktívát kínál, amelyek egyszerűsítik a gyakori PHP vezérlési struktúrák és egyéb műveletek használatát a nézetekben. Ezek a direktívák tiszta és tömör módon teszik lehetővé a PHP kód használatát, miközben ismerősek maradnak PHP megfelelőikkel. A Blade direktívák @ szimbólummal kezdődnek.

Alapvető Direktívák:

- Adatmegjelenítés: A nézeteknek átadott adatok megjelenítésére a {{ $változó }} szintaxis használható. Ezek a {{ }} echo utasítások alapértelmezés szerint a PHP htmlspecialchars függvényén keresztül futnak az XSS támadások megelőzése érdekében. Ha nem szeretnénk, hogy az adat ki legyen kódolva, a {!! $változó !!} szintaxist használhatjuk. Vigyázni kell azonban a felhasználó által megadott tartalommal, ilyenkor általában a kódolt szintaxis javasolt az XSS megelőzése végett. PHP függvények eredményét is megjeleníthetjük a Blade echo utasításokban.

Vezérlési Struktúrák:

- If Utasítások: @if, @elseif, @else és @endif direktívák használhatók if utasítások létrehozására, amelyek ugyanúgy működnek, mint PHP megfelelőik. Létezik továbbá

egy @unless direktíva is. Az @isset és @empty direktívák a PHP isset() és empty() függvények kényelmes helyettesítői.

- Autentikációs Direktívák: Az @auth és @guest direktívákkal gyorsan ellenőrizhetjük, hogy az aktuális felhasználó be van-e jelentkezve vagy vendég. Megadható az ellenőrizendő autentikációs "guard" is, például @auth('admin').
- Ciklusok: A Blade egyszerű direktívákat biztosít a PHP ciklusokhoz: @for, @foreach, @forelse, @while. A @continue és @break direktívák használhatók az aktuális iteráció átugrására vagy a ciklus befejezésére. A folytatási vagy megszakítási feltétel közvetlenül a direktíva deklarációjában is megadható, például @continue($user->type == 1).

Nézetek Beillesztése:

- @include Direktíva: Lehetővé teszi egy másik Blade nézet beillesztését egy nézetbe. Az eredeti nézetben elérhető összes változó elérhető lesz a beillesztett nézetben is. További adatokat is átadhatunk a beillesztett nézetnek egy tömb formájában a @include direktíva második argumentumaként. Ha a beillesztendő nézet nem létezik, hiba keletkezik. A nem létező nézetek beillesztésére használható az @includeIf direktíva.
- Kommentek: A {{-- Komment --}} szintaxissal Blade kommenteket hozhatunk létre, amelyek nem jelennek meg a renderelt HTML-ben.
- @extends direktíva: Ez a direktíva egy gyermek nézetben használatos annak megadására, hogy melyik szülő layoutot (elrendezést) kell örökölnie. A @extends hivatkozza a layout nézetének fájlját (például 'layouts.app'). Ezáltal a gyermek nézet átveszi a szülő layout struktúráját és a benne definiált szekciókat.
- @yield direktíva: Ezt a direktívát a szülő layout fájljában használják arra, hogy helyet foglaljanak a gyermek nézetek által definiált, dinamikus tartalomnak. A @yield egy nevet vár paraméterként (például @yield('content')), amely megegyezik a gyermek nézetben definiált szekció nevével. A @yield direktíva második paraméterként egy alapértelmezett értéket is elfogadhat, amely akkor jelenik meg, ha a gyermek nézet nem definiál tartalmat az adott szekcióhoz.
- @section direktíva: Ezt a direktívát mind a szülő layoutban, mind a gyermek nézetekben használják. A szülő layoutban a @section direktívával lehet alapértelmezett tartalmat definiálni egy adott szekcióhoz. Ezt a szekciót a @yield direktívával lehet megjeleníteni. A szekció definíciója @endsection vagy @show direktívával zárulhat. A @show azonnal megjeleníti is a szekció tartalmát.

A Blade direktívák hatékony eszközei a nézetekben történő fejlesztésnek, lehetővé téve a tiszta és olvasható kód írását a gyakori feladatokhoz.

4.1.12. Validáció

A kontrollerekben való adatvalidáció a validate() metódus segítségével egy rendkívül egyszerű folyamat a Laravelben a beérkező adatok megfelelőségének biztosítására. A request() helper függvényen keresztül érhetjük el a validate() metódust. Ennek a metódusnak egy tömböt adhatunk át, amely meghatározza, hogy mely bemeneti mezőket szeretnénk validálni és milyen követelményeknek kell megfelelniük.

Például, ha van egy űrlapunk egy idea nevű szövegterülettel, validálhatjuk, hogy ez a mező kötelező legyen, legalább 3 karakter hosszú legyen, és ne legyen hosszabb 240 karakternél:

```
$validated = request()->validate([
    'idea' => 'required|min:3|max:240',
]);
```

A validációs szabályokat egy függőleges vonallal (|) választhatjuk el egymástól. Számos előre elkészített validációs szabály áll rendelkezésünkre, mint például a required (a mezőnek kötelezően tartalmaznia kell értéket), a min (minimum karakterszám vagy elem), és a max (maximum karakterszám vagy elem). A Laravel validációs dokumentációjában az összes elérhető szabály megtalálható (https://laravel.com/docs/11.x/validation). Nem szükséges ezeket megjegyezni, bármikor megtekinthetők a dokumentációban.

Ha a validáció sikertelen, a Laravel automatikusan visszairányítja a felhasználót az előző oldalra, és a hibákat a munkamenetben tárolja. A Blade sablonokban egyszerűen megjeleníthetjük ezeket a hibaüzeneteket az @error direktíva és a $message változó segítségével:

```
@error('idea')
    <span class="fs-6 text-danger">{{ $message }}</span>
@enderror
```

Ez a kód a idea nevű mezőhöz tartozó hibaüzenetet jeleníti meg, ha van ilyen mezőhöz kapcsolódó hiba.

4.1.13. Pagination

A pagination a Laravelben egy módszer az ötletek megjelenítésének kezelésére azáltal, hogy azokat külön oldalakra osztja. A vezérlőben a modell lekérdezésekor a paginate()

metódust használjuk a get() helyett, megadva az oldalonként megjelenítendő elemek számát. A Blade sablonban a lapozási linkek megjelenítéséhez a kollekción (a paginate() eredményén) a links() metódust hívjuk meg ($ideas->links()). A lapozás alapértelmezett stílusa Tailwind CSS, de Bootstrap használatához a AppServiceProvider boot() metódusában be kell állítani (\Illuminate\Pagination\Paginator::useBootstrapFive(); Bootstrap 5 esetén). Ezután a lapozási gombok a beállított stílussal jelennek meg, lehetővé téve a felhasználók számára az oldalak közötti navigálást. Amennyiben nem található ötlet a leszűrt tartományban, a weboldal ezt üzenet formájában jelzi az ötleteket tartalmazó konténer helyén. Ezt egy @forelse direktívával valósítjuk meg.

```php
public function index()
{
    $ideas = Idea::orderBy('created_at', 'desc');

    $ideas = $ideas->search(request()->search);

    return view("dashboard", [
        'ideas'=> $ideas->paginate(5)
    ]);
}
```

79. ábra Pagination I.

```php
<div class="mt-2">
    {{ $ideas->withQueryString()->links() }}
</div>
```

80. ábra Pagination II.

```php
public function boot()
{
    Paginator::useBootstrapFive();
```

81. ábra Bootstrap 5 regisztrálása

```php
@forelse ($ideas as $idea)
<div class="mt-3">
    @include('ideas.shared.idea-card')
</div>
@empty
<p class="text-center mt-4">No results found.</p>
@endforelse
<div class="mt-2">
    {{ $ideas->withQueryString()->links() }}
</div>
```

82. ábra Pagination III.

4.1.14. Keresősáv (Searchbar)

A keresősáv funkció implementálásához a Laravel alkalmazásban egy dedikált HTML űrlapot hozunk létre, amelyet gyakran egy külön Blade komponensbe szervezünk a kód tisztasága érdekében. Ez az űrlap a GET metódust használja az adatok küldéséhez. Az űrlap action attribútuma meghatározza azt az útvonalat, ahová a kérés elküldésre kerül, ami a példában a dashboard elnevezésű útvonal.

Az űrlap tartalmaz egy szöveges beviteli mezőt, amelynek name attribútuma search. A felhasználó által beírt keresési kifejezés ezen a néven kerül elküldésre a szerverre a URL paraméterek között. A beviteli mező value attribútuma dinamikusan beállítható a request('search', ' ') helper függvénnyel, így ha korábban történt keresés, a beírt érték megmarad a mezőben. Az űrlapon továbbá található egy "Search" feliratú gomb a keresés elindításához.

A szerver oldalon, a megfelelő kontrollerben ellenőrizzük, hogy a kérés tartalmaz-e search nevű paramétert a request()->has('search') metódussal. Ha a keresési paraméter létezik, akkor módosítjuk az adatbázis lekérdezést úgy, hogy csak azokat az elemeket (például ötleteket) kérjük le, amelyek a keresési kifejezést tartalmazzák. Ezt a where() metódus segítségével érjük el a modellünkön (például Idea::where()), ahol a keresett oszlopot (content), a keresési típust ('like'), és a keresési kifejezést adjuk meg. A keresési kifejezés elé és mögé % (százalék) karaktereket illesztünk a like operátorral való használathoz, ezzel lehetővé téve a részleges egyezéseket. A tényleges keresési kifejezést a request('search') helper függvénnyel szerezzük be.

A szűrést követően az eredményeket a paginate() metódussal lapozzuk. Ez biztosítja, hogy nagy mennyiségű találat esetén az adatok több oldalon jelenjenek meg. A lapozási linkek a Blade sablonban a kollekción meghívott links() metódussal jeleníthetők meg.

A keresési logikát az Idea modellben definiáljuk scopeSearch néven. Ez egy Laravel query scope, amely lehetővé teszi, hogy a search paramétert egyszerűen láncoljuk a lekérdezéshez (Idea::search($search) formában). A scope belsejében a where feltétel a content mezőt vizsgálja, és a LIKE operátor segítségével részleges egyezést keres a megadott kifejezéssel. Az % karakterek lehetővé teszik, hogy a keresett szó bármelyik részén szerepeljen a találatban. A lekérdezés ezután paginate() hívással darabolódik oldalanként, így a találatok kényelmesen megjeleníthetők a felhasználó számára.

```php
public function index()
{
    $ideas = Idea::orderBy('created_at', 'desc');

    $ideas = $ideas->search(request()->search);

    return view("dashboard", [
        'ideas'=> $ideas->paginate(5)
    ]);
}
```

83. ábra Keresősáv I.

```blade
<div class="card">
    <div class="card-header pb-0 border-0">
        <h5 class="">Search</h5>
    </div>
    <div class="card-body">
        <form action=" {{ route('dashboard') }} " method="get">
            <input value="{{ request('search', '') }}" placeholder="..." class="form-control w-100" type="text" name="search">
            <button class="btn btn-dark mt-2"> Search</button>
        </form>
    </div>
</div>
```

84. ábra Keresősáv II.

```php
public function scopeSearch(Builder $query, $search = ""){
    if($search){
        $query->where('content','like','%'.$search.'%');
    }
}
```

85. ábra Scope

4.1.15. Kapcsolatok Laravelben

A Laravelben a kapcsolatok (relationships) teszik lehetővé, hogy az adatbázisban tárolt különböző modellek közötti összefüggéseket egyszerűen kezeljük. A forrás egy "one-to-many" (egy-a-többhöz) kapcsolat létrehozását mutatja be egy ötlet (Idea) és a hozzátartozó megjegyzések (Comment) között.

Kapcsolat definiálása a modellben:

A kapcsolatokat a Laravel modelljeiben definiáljuk metódusok segítségével. A metódus neve általában a kapcsolódó tábla neve szemesen (egyes számban), ha egyetlen kapcsolódó modellre hivatkozunk, vagy többes számban, ha több kapcsolódó modellre.

Egy "one-to-many" kapcsolat definiálásához az ősosztály (Illuminate\Database\Eloquent\Model) által biztosított hasMany() metódust használjuk. Például az Idea modellben a megjegyzésekkel való kapcsolatot a következőképpen definiálhatjuk:

```php
public function comments(){
    return $this->hasMany(Comment::class);
}
```

86. ábra 1:N kapcsolat definiálása

A hasMany() metódus első argumentuma a kapcsolódó modell osztályának neve (Comment::class).

Külső és helyi kulcsok:

A hasMany() metódus opcionálisan elfogadhat két további argumentumot: a külső kulcsot (foreign key) és a helyi kulcsot (local key). A külső kulcs az a mező a kapcsolódó táblában (a comments táblában), amely a szülő modell egyedi azonosítójára (a ideas táblában) mutat. A helyi kulcs a szülő modell egyedi azonosítója (általában az id mező).

Ha követjük a Laravel névadási konvencióit (a külső kulcs neve a szülő modell nevéből képződik aláhúzással és _id végződéssel, például idea_id, és a szülő modell elsődleges kulcsa id), akkor nem szükséges ezeket az argumentumokat megadnunk. A Laravel automatikusan felismeri a kapcsolatot. A forrás példájában a comments migrációban az idea_id foreign keyként lett definiálva, ezért a kapcsolat automatikusan működni fog.

Kapcsolódó adatok elérése:

A kapcsolat definiálása után a kapcsolódó adatokat egyszerűen elérhetjük a modell egy tulajdonságaként. Például egy adott $idea objektumhoz tartozó összes megjegyzést a $idea->comments kóddal érhetjük el. Ez egy Eloquent kollekciót ad vissza, amely tartalmazza az összes kapcsolódó Comment modellt. Fontos megjegyezni, hogy a kapcsolat nevét (a metódus nevét a modellben) használjuk tulajdonságként, zárójelek nélkül.

```php
class CommentController extends Controller
{
    public function store(CreateCommentRequest $request ,Idea $idea){
        $validated = $request->validated();

        $idea->comments()->create([
            "idea_id"=> $idea->id,
            "user_id"=>auth()->id(),
            "content"=>$validated["comment_content"]
        ]);



        return to_route('ideas.show', $idea->id)->with("success","Comment successfully added");
    }
}
```

87. ábra Kapcsolat használata kontrollerben

Kapcsolatok használata Blade sablonokban:

A Blade sablonokban a kapcsolódó adatokat ciklusokkal jeleníthetjük meg. Például egy ötlet megjegyzéseinek listázásához a comments kapcsolaton keresztül iterálhatunk:

```
<hr>
@forelse ($idea->comments->sortByDesc('created_at') as $comment)
    <div class="d-flex align-items-start">
        <img style="width:35px" class="me-2 avatar-sm rounded-circle" src="{{ $comment->user->getImageURL() }}"
            alt="{{ $comment->user->name }} Avatar">
        <div class="w-100">
            <div class="d-flex justify-content-between">
                <h6 class="">{{ $comment->user->name }}
                </h6>
                <small class="fs-6 fw-light text-muted"> {{ $comment->created_at->diffForHumans() }} </small>
            </div>
            <p class="fs-6 mt-3 fw-light">
                {{ $comment->content }}
            </p>
        </div>
    </div>
@empty
    <p class="text-center mt-4">This post doesn't have any comments.</p>
@endforelse
```

88. ábra Kapcsolat használata view fileban

Rövidebb módszer a megjegyzés létrehozásához (nem közvetlenül a kapcsolat, de kapcsolódik):

A Laravel kapcsolatok használata jelentősen leegyszerűsíti az adatbázisban lévő modellek közötti interakciót, csökkentve a manuális lekérdezések írásának szükségességét. A forrásban bemutatott "one-to-many" kapcsolat csak egy a Laravel által támogatott többféle kapcsolattípus közül.

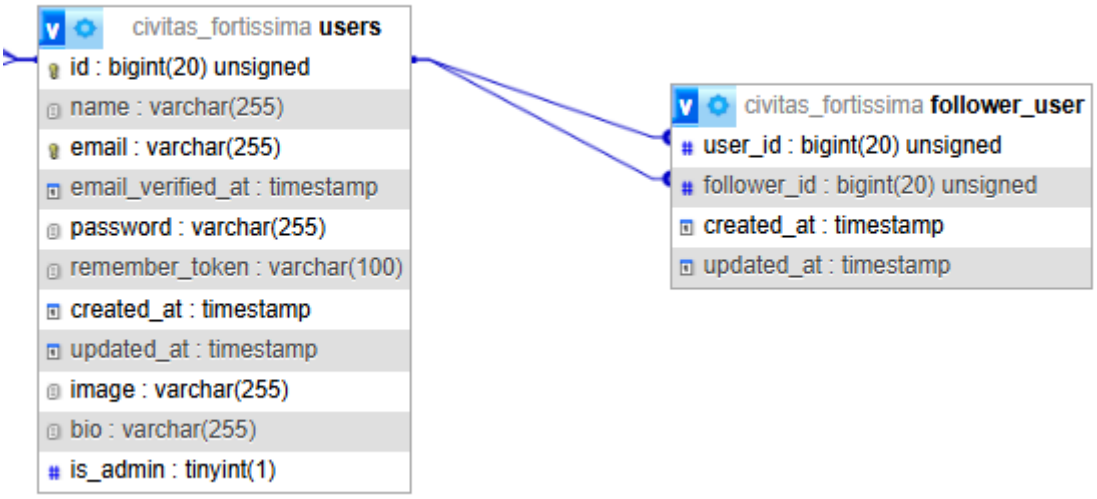
4.1.15.1. Pivot táblák

A pivot táblák a Laravelben a sok-a-sokhoz (many-to-many) kapcsolatok kezelésére szolgálnak. Amikor két modell sok közös elemmel rendelkezhet, egy köztes táblára van szükség az ezen kapcsolatok tárolásához. Ezt a köztes táblát nevezzük pivot táblának.

Pivot tábla elnevezési konvenciói:

A Laravel közösségében elterjedt névadási konvenció szerint a pivot tábla neve a két kapcsolódó tábla egyes számú elnevezéséből képződik, aláhúzással elválasztva és alfabetikus sorrendben. Például, ha van egy posts táblád és egy tags táblád, a pivot tábla neve általában post_tag lesz.

Azonban, ha ez a konvenció nem elég leíró, vagy ha a két kapcsolódó modell ugyanaz (például felhasználó követ egy másik felhasználót), akkor eltérő elnevezés is használható. Például a mi projektünkben a user és a follower közötti kapcsolatnál a follower_user név lett választva, mivel ez jobban tükrözi a kapcsolat jellegét, mint az, hogy user_user.



89. ábra Pivot tábla

A pivot táblához először egy migrációt kell létrehozni. A migrációban definiálni kell a tábla sémáját, ami általában a két kapcsolódó modell foreign keyeit tartalmazza. Például egy follower_user táblában lenne egy user_id és egy follower_id. A foreign ID a Laravelben az idid névadási konvenciót is követheti. Általában a pivot táblákhoz nem szükséges külön modellt létrehozni.

Kapcsolat definiálása a modellekben:

Az N:M kapcsolatot a Laravel modelljeiben a belongsToMany() metódussal definiáljuk. Például a User modellben a követőkkel való kapcsolatot így lehet definiálni:

```
public function followers(){
    return $this->belongsToMany(User::class, 'follower_user', 'user_id','follower_id')->withTimestamps();
}
```

90. ábra N:M kapcsolat definiálása pivot táblával I.

A belongsToMany() metódus legalább a kapcsolódó modell osztályát (User::class) fogadja el. A második argumentum a pivot tábla neve (follower_user). Ha a Laravel névadási konvencióit követjük, és mindkét kapcsolódó entitásnak van külön modellje, akkor a pivot tábla nevét nem feltétlenül kell megadni, mert a Laravel automatikusan felismeri. Azonban a forrás példájában, mivel nincs külön Follower modell, a pivot tábla nevét expliciten meg kell adni.

A harmadik argumentum a jelenlegi modell idegen kulcsa a pivot táblában (user_id), a negyedik pedig a kapcsolódó modell idegen kulcsa a pivot táblában (follower_id). Ha a Laravel névadási konvencióit használjuk (user_id, idea_id), akkor ezeket az argumentumokat sem feltétlenül kell megadnunk.

Egy másik példa a forrásban az ötletek (Idea) és a felhasználók (User) közötti "like" kapcsolat, ahol a pivot tábla neve idea_like. A kapcsolat az Idea modellben így néz ki:

```php
public function likes(){
    return $this->belongsToMany(User::class, 'idea_like', 'user_id', 'idea_id')->withTimestamps();
}
```

91. ábra N:M kapcsolat definiálása pivot táblával II.

Itt is meg kell adni a pivot tábla nevét, mivel nem követi a szigorú névadási konvenciót.

4.1.16. Flash Messages

A flash üzenetek olyan sikeres műveleteket jelző üzenetek, amelyeket a felhasználóknak szeretnénk megjeleníteni. Ezek egyszeri munkamenetek (one-time sessions), ami azt jelenti, hogy a megjelenítésük után automatikusan törlődnek. Ezáltal nagyon hasznosak.

A Laravelben a vezérlők (például egy store metódusban) a with() függvény segítségével hozhatók létre. Ennek a függvénynek az első argumentuma a munkamenet kulcsa (például 'success'), a második pedig az üzenet maga:

```php
public function store(CreateIdeaRequest $request){
    $validated = $request->validated();

    Idea::create([
        'content' => $validated['idea_content'] ?? null,
        "user_id"=>auth()->id()
    ]);

    return to_route('dashboard')->with('success','Idea was created successfully.');
}
```

92. ábra Megerősítő szöveg I.

A flash üzenetek megjelenítéséhez a Blade sablonokban a session() helper függvényt használhatjuk. Először érdemes ellenőrizni a has() metódussal, hogy létezik-e a megadott kulcsú munkamenet (például session()->has('success')). Ha létezik, akkor a session('success') módszerrel magát az üzenetet is lekérhetjük és megjeleníthetjük. A példában egy 'shared.success message' Blade komponenst használnak fel, amely egy Bootstrap értesítést jelenít meg, de csak akkor, ha a 'success' munkamenet létezik. Miután az üzenet megjelent, és a felhasználó egy új oldalt tölt be vagy navigál el, az automatikusan eltűnik

resources > views > shared > success-message.blade.php > ...

```blade
@if (session()->has('success'))
    <div class="alert alert-success alert-dismissible fade show" role="alert">
        {{ session('success') }}
        <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
    </div>
@endif
```

93. ábra Megerősítő szöveg II.

4.1.17. Regisztráció

A regisztráció egy új felhasználói fiók létrehozásának folyamata. Projektünkben a regisztrációt a következő lépésekkel valósítottuk meg:

Először is definiáltunk egy útvonalat a routes/auth.php fájlban a regisztrációs űrlap megjelenítéséhez. Ezt egy GET kéréssel kezeltük, a /register útvonalon. Emellett definiáltunk egy másik útvonalat ugyanazon az URL-en, de POST kéréssel, a regisztrációs űrlap adatainak fogadására és feldolgozására. Ezeket az útvonalakat a Route::get() és Route::post() statikus metódusokkal hoztuk létre. Lehetőséget adtunk az útvonal elnevezésére is, például register, ami megkönnyíti a hivatkozást a nézetekben.

```
Route::prefix('auth')->group(function () {
    Route::post('/register', [AuthController::class, 'store']);
    Route::post('/login', [AuthController::class, 'authenticate'])->name('login');
    Route::post('/logout', [AuthController::class, 'logout'])->name('logout');
});

Route::get('/login', [AuthController::class, 'login']);
Route::get('/register', [AuthController::class, 'register'])->name('register');
```

94. ábra Auth route-ok

A regisztrációs logikát egy kontrollerben helyeztük el. Létrehoztunk egy AuthControllernevű kontrollert az Artisan parancssori eszközzel: php artisan make:controller AuthController. Ebben a kontrollerben implementáltunk egy register metódust a regisztrációs űrlap megjelenítésére, és egy store metódust a beérkező regisztrációs adatok validálására és az új felhasználó létrehozására.

```php
public function register(){
    return view('auth.register');
}
public function store(){
    $validated = request()->validate([
        'name'=>'required|min:3|max:40',
        'email'=> 'required|email|unique:users,email',
        'password'=> 'required|confirmed|min:8'
    ]);
    if($validated){
        $user = User::create([
            'name'=> $validated['name'],
            'email'=> $validated['email'],
            'password'=> Hash::make($validated['password']),
        ]);

        return redirect()->route('login')->with('success','Account created successfully');
    }


    return response()->json(["error" => "Invalid or missing credentials"]);

    Mail::to($user->email)
    ->send(new WelcomeEmail($user));

    return redirect()->route('login')->with('success','Account created successfully');
}
```

95. ábra Regisztrációs logika a kontrollerben

Létrehoztunk egy Blade sablonfájlt (például register.blade.php) a regisztrációs űrlaphoz a resources/views/auth mappában. A Blade sablonban egy HTML űrlapot hoztunk létre a felhasználói adatok (név, email, jelszó stb.) bekéréséhez. Az űrlapon elhelyeztünk egy @csrf direktívát a CSRF (Cross-Site Request Forgery) védelem érdekében. Az űrlap egy POST kérést küld a regisztrációt feldolgozó útvonalra. A nézetben kiterjesztettük az alapértelmezett elrendezést (@extends('layouts.app')) és definiáltuk a tartalom szekciót (@section('content')).

```blade
<form class="form mt-5" action="{{ route('register') }}" method="post">
    @csrf
    <h3 class="text-center text-dark">Register</h3>
    <div class="form-group">
        <label for="name" class="text-dark">Name:</label><br>
        <input type="text" name="name" id="name" class="form-control">
        @error('name')
            <span class="d-block fs-6 text-danger mt-2">{{ $message }}</span>
        @enderror
    </div>
    <div class="form-group mt-3">
        <label for="email" class="text-dark">Email:</label><br>
        <input type="email" name="email" id="email" class="form-control">
        @error('email')
            <span class="d-block fs-6 text-danger mt-2">{{ $message }}</span>
        @enderror
    </div>
    <div class="form-group mt-3">
        <label for="password" class="text-dark">Password:</label><br>
        <input type="password" name="password" id="password" class="form-control">
        @error('password')
            <span class="d-block fs-6 text-danger mt-2">{{ $message }}</span>
        @enderror
    </div>
    <div class="form-group mt-3">
        <label for="password_confirmation" class="text-dark">Confirm Password:</label><br>
        <input type="password" name="password_confirmation" id="password_confirmation" class="form-control">
        @error('password')
            <span class="d-block fs-6 text-danger mt-2">{{ $message }}</span>
        @enderror
    </div>
    <div class="form-group">
        <label for="remember-me" class="text-dark"></label><br>
        <input type="submit" name="submit" class="btn btn-dark btn-md" value="submit">
    </div>
    <div class="text-right mt-2">
        <a href="/login" class="text-dark">Login here</a>
    </div>
</form>
```

96. ábra Regisztrációs form

A kontroller store metódusában implementáltuk a beérkező adatok validációját a request()->validate() metódussal. Megadtuk a validációs szabályokat a szükséges mezőkre (például required, string, email, unique:users, min:8, confirmed a jelszóhoz és annak megerősítéséhez). A validációs hibák megjelenítéséhez a Blade nézetben az @error direktívát használtuk.

Sikeres validáció esetén a kontrollerben létrehoztuk az új felhasználói rekordot az adatbázisban a megadott adatokkal. A jelszót titkosítottuk a Hash::make() metódus segítségével. Ehhez a User modellt használtuk, amely az adatbázis users táblájával való interakciót teszi lehetővé. A fillable tulajdonságban adtuk meg, mely mezők tömeges hozzárendeléssel módosíthatóak.

A sikeres regisztrációt követően a felhasználót átirányítottuk egy másik oldalra (például a bejelentkezési oldalra vagy a főoldalra) a redirect() funkcióval. Flash üzeneteket is megjeleníthettünk a with() metódussal, tájékoztatva a felhasználót a sikeres regisztrációról.

```php
return response()->json(["error" => "Invalid or missing credentials"]);
```

97. ábra JSON alapú hibaüzenet

A regisztráció után beállíthattuk, hogy egy üdvözlő emailt küldjünk az új felhasználónak a Laravel beépített Mail funkciójával. Ehhez létrehoztunk egy Mailable osztályt (például WelcomeEmail) az Artisan paranccsal: php artisan make:mail WelcomeEmail. Ebben definiáltuk az email tárgyát, a feladót, a címzettet és a tartalmát (ami lehet egy Blade nézet). Az emailt a regisztrációt követően a Mail::to($user->email)->send(new WelcomeEmail($user)) kóddal küldtük el. A Mailtrap szolgáltatást használhattuk az éles email küldés tesztelésére.

```php
Mail::to($user->email)->send(new WelcomeEmail($user));
```

98. ábra Email küldése

```php
public function envelope()
{
    return new Envelope(
        subject: 'Thanks for joining ' . config('app.name', ''),
    );
}

/**
 * Get the message content definition.
 *
 * @return \Illuminate\Mail\Mailables\Content
 */
public function content()
{
    return new Content(
        view: 'emails.welcome-email',
        with: [
            'user' => $this->user
        ]
    );
}
```

99. ábra Mailable osztály

4.1.18. Belépés

A belépés (bejelentkezés) egy már regisztrált felhasználó számára teszi lehetővé az alkalmazás funkcióinak elérését. A projektünkben a belépési folyamat a következőképpen működik:

Hasonlóan a regisztrációhoz, definiáltunk egy GET útvonalat a bejelentkezési űrlap megjelenítésére (például /login), és egy POST útvonalat ugyanazon az URL-en a bejelentkezési

adatok fogadására és hitelesítésére. A POST kérést az AuthController metódusához irányítottuk. Az útvonalat elneveztük például login.

A belépési logikát is az AuthController nevű kontrollerben implementáltuk. A bejelentkezési űrlap megjelenítésére egy login metódust, a bejelentkezési kísérlet kezelésére pedig egy authenticate metódust hoztunk létre.

```php
public function login(){
    return view('auth.login');
}

public function authenticate(){
    $validated = request()->validate([
        'email'=> 'required|email',
        'password'=> 'required|min:8'
    ]);

    if(auth()->attempt($validated)){
        request()->session()->regenerate();
        return redirect()->route('dashboard')->with('success','Logged in as ' . auth()->user()->name);
    }

    return redirect()->route('login')->withErrors([
        'email'=> 'No matching email and password combination found.'
    ]);
}
```

100. ábra Belépési logika a kontrollerben

Létrehoztunk egy Blade sablonfájlt a bejelentkezési űrlaphoz a resources/views/auth mappában. Az űrlapon az email cím és jelszó megadására szolgáló mezőket helyeztünk el, valamint a @csrf direktívát a CSRF védelemhez.

```
<form class="form mt-5" action="{{route('login')}}" method="post">
    @csrf
    <h3 class="text-center text-dark">Login</h3>
    <div class="form-group">
        <label for="email" class="text-dark">Email:</label><br>
        <input type="email" name="email" id="email" class="form-control">
        @error('email')
        <span class="d-block fs-6 text-danger mt-2">{{ $message }}</span>
    @enderror
    </div>
    <div class="form-group mt-3">
        <label for="password" class="text-dark">Password:</label><br>
        <input type="password" name="password" id="password" class="form-control">
        @error('password')
        <span class="d-block fs-6 text-danger mt-2">{{ $message }}</span>
    @enderror
    </div>
    <div class="form-group">
        <label for="remember-me" class="text-dark"></label><br>
        <input type="submit" name="submit" class="btn btn-dark btn-md" value="submit">
    </div>
    <div class="text-right mt-2">
        <a href="/register" class="text-dark">Register here</a>
    </div>
</form>
```

101. ábra Bejelentkezési form

A kontroller authenticate metódusában a Auth::attempt() metódust (vagy az auth() helper függvény attempt() metódusát) használtuk a felhasználó hitelesítésére. Ez a metódus egy tömböt vár, amely tartalmazza a hitelesítő adatokat (például email és password), és megpróbálja bejelentkeztetni a felhasználót. A megadott jelszót automatikusan összehasonlítja az adatbázisban tárolt titkosított jelszóval.

Sikeres bejelentkezés esetén az Auth::attempt() metódus true értéket ad vissza, és a Laravel automatikusan létrehoz egy munkamenetet a felhasználó számára. A bejelentkezett felhasználó adatai az Auth::user() metódussal érhetők el. Sikertelen bejelentkezés esetén az Auth::attempt() false értéket ad vissza, és a felhasználót visszairányíthatjuk a bejelentkezési oldalra hibaüzenetekkel a withErrors() metódussal.

Sikeres bejelentkezés után a felhasználót átirányítottuk a főoldalra a redirect()->route('dashboard') segítségével. Ehhez csatoltunk egy with() függvényt is, amellyel egy flash üzenetet adtunk át a következő kéréshez. Ez az üzenet tájékoztatja a felhasználót a sikeres bejelentkezésről.

```
if(auth()->attempt($validated)){
    request()->session()->regenerate();
    return redirect()->route('dashboard')->with('success','Logged in as ' . auth()->user()->name);
}
```

102. ábra Belépési logika kontrollerben

4.1.19. Kijelentkezés

A kijelentkezési folyamat lehetővé teszi a bejelentkezett felhasználók számára a kijelentkezést az alkalmazásból. Ezt a következőképpen valósítottuk meg:

Definiáltunk egy útvonalat a kijelentkezéshez. Mivel ez egy műveletet hajt végre az alkalmazás állapotában, egy POST kérést használunk.

```
Route::post('/logout', [AuthController::class, 'logout'])->name('logout');
```

103. ábra Logout route

Az AuthControllerben létrehozott logout metódusban a bejelentkezett felhasználó munkamenetét megszüntetjük az Auth::logout() metódussal. A felhasználó munkamenetének érvénytelenítjük a request()->session()->invalidate() metódussal. Az alkalmazás által használt CSRF tokent újrageneráljuk a request()->session()->regenerateToken() metódussal. A felhasználót átirányítjuk a főoldalra egy sikeres kijelentkezés üzenettel. A navigációs sávban elhelyeztünk egy gombot, amely a /logout POST útvonalra mutat.

```
public function logout(){
    auth()->logout();
    request()->session()->invalidate();
    request()->session()->regenerate();

    return redirect()->route('dashboard')->with('success','Successfully logged out');
}
```

104. ábra Kijelentkezési logika kontrollerben

4.1.20. Autorizáció

Új ötlet létrehozásakor tároljuk a bejelentkezett felhasználó azonosítóját. Az IdeaController store metódusában hozzáadjuk a user_id-t a validátorhoz. A bejelentkezett felhasználó azonosítóját az auth()->id() helper függvénnyel szerezhetjük meg. A user_id-t hozzáadjuk az Idea modell fillable tömbjéhez a tömeges hozzárendelés engedélyezéséhez.

```
public function store(CreateIdeaRequest $request){
    $validated = $request->validated();

    Idea::create([
        'content' => $validated['idea_content'] ?? null,
        "user_id"=>auth()->id()
    ]);

    return to_route('dashboard')->with('success','Idea was created successfully.');
}
```

105. ábra Ötletlétrehozási logika kontrollerben

Annak biztosítására, hogy csak bejelentkezett felhasználók küldhessenek be ötleteket, az ötletbeküldő Blade komponensben egy @auth direktívát használunk. Ha a felhasználó nincs bejelentkezve, egy üzenetet jelenítünk meg.

A bejelentkezett felhasználó nevének megjelenítéséhez az Idea modellben definiálunk egy user kapcsolatot a belongsTo típussal, mivel egy ötlet egy felhasználóhoz tartozik. A User modellben definiálunk egy ideas kapcsolatot a hasMany típussal, mivel egy felhasználónak több ötlete lehet. Ezt a kapcsolatot felhasználhatjuk az ötlethez tartozó felhasználó adatainak eléréséhez a Blade sablonban ($idea->user->name).

Annak megakadályozására, hogy a ki nem jelentkezett felhasználók szerkeszthessék az ötleteket, a routes/web.php fájlban az ötletek szerkesztésére szolgáló útvonalhoz hozzáadjuk az auth middleware-t. Ez biztosítja, hogy csak bejelentkezett felhasználók érhessék el ezeket az útvonalakat, ellenkező esetben a rendszer átirányítja őket a bejelentkezési oldalra. Ezt megtesszük az update (frissítés) és destroy (törlés) útvonalakhoz, valamint a hozzászólások közzétételéhez is.

```
Route::resource('ideas', IdeaController::class)->except('index','create','show')->middleware('auth');
Route::resource('ideas', IdeaController::class)->only('show');

Route::resource('ideas.comments', CommentController::class)->only('store')->middleware('auth');
```

106. ábra Auth middleware

Végül, annak biztosítására, hogy csak az ötlet tulajdonosa törölhesse vagy szerkeszthesse azt, az IdeaController destroy és update metódusaiban ellenőrizzük, hogy a bejelentkezett felhasználó azonosítója megegyezik-e az ötlet user_id-jével. Ha nem egyezik, a abort(404) függvényt használjuk egy "nem található" hibaüzenet küldéséhez. Ezt az ellenőrzést az edit metódushoz is hozzáadjuk, hogy a nem tulajdonosok még a szerkesztőoldalt se láthassák. Az "edit" és "delete" gombok attól függően jelennek meg, hogy a bejelentkezett felhasználó az ötlet tulajdonosa-e.

4.1.21. Profile page

A profile page lehetővé teszi a felhasználók számára, hogy megtekintsék egymás, illetve a saját profiljaikat. Amennyiben a profil a sajátuké, lehetőségük van módosításokat végezni a felhasználó-adataikon. Például, beállíthatnak az alapértelmezettől eltérő, egyedi profilképet, vagy adhatnak leírást a profilukhoz (bio).

Létrehoztunk egy külön mappát "users" néven a nézetek (views) között, amelyben a felhasználói profilok megjelenítésére (show.blade.php) és szerkesztésére (edit.blade.php) szolgáló fájlok találhatók.

A profiloldal elrendezése hasonló az "idea" oldaléhoz. Ezt a HTML-kódot refaktoráltuk egy külön Blade fájlba "user-card.blade.php" néven, amelyet a "shared" könyvtárban helyeztünk el. A user-card sablont a users.show nézetben használjuk. A felhasználói profiloldal útvonala /users/{id}.

Implementáltunk egy logikát, amely lehetővé teszi a "follow" (követés) gomb megjelenítését más felhasználók profilján, de ennek megtekintéséhez be kell jelentkezni. Ezt az auth() segédfüggvénnyel és egy feltételes utasítással valósítottuk meg, amely ellenőrzi a bejelentkezett felhasználó azonosítóját.

```
@auth
    @if (auth()->id() !== $user->id)
        <div class="mt-3">
            @if (auth()->user()->follows($user))
                <form action="{{ route('users.unfollow', $user->id) }}" method="post">
                    @csrf
                    <button type="submit" class="btn btn-danger btn-sm"> Unfollow </button>
                </form>
            @else
                <form action="{{ route('users.follow', $user->id) }}" method="post">
                    @csrf
                    <button type="submit" class="btn btn-primary btn-sm"> Follow </button>
                </form>
            @endif
        </div>
    @endif
@endauth
```

107. ábra Follow vagy Unfollow gomb megjelenítése követéstől függően

Hozzáadtuk a profilkép feltöltésének és frissítésének lehetőségét a profil szerkesztő oldalhoz egy "image" nevű fájlfeltöltő mező segítségével.

Amennyiben a felhasználó nem töltött fel profilképet, alapértelmezett profilképpel fog visszatérni ez a függvény

A navigációs menüben a "profile" útvonalat rendeltük a profil linkhez. A profil szerkesztő oldal címét beállíthatjuk a Blade sablonban egy szekció segítségével.

A felhasználó megjelenítő oldal címét dinamikusan beállítjuk, a felhasználónevet megjelenítve.

4.1.22. Profilkép feltöltése

A szerkesztő oldalon létrehoztunk egy "image" nevű fájlfeltöltő mezőt (<input type="file" name="image" class="form-control">) a profilkép feltöltéséhez. A formot úgy állítottuk be, hogy támogassa a fájlfeltöltést az enctype="multipart/form-data" attribútum hozzáadásával.

Amikor a felhasználó elküldi a formot, a feltöltött fájl a szerverre kerül. A backend oldalon validáljuk a beküldött képet az image validációs szabállyal, ami ellenőrzi, hogy a fájl

egy érvényes képformátum (például jpeg, jpg, png). Ezután ellenőrizzük, hogy érkezett-e kép a kérésben a request()->has('image') metódussal. A feltöltött képet a store() metódussal mentjük el a szerver fájlrendszerére. A request()->file('image')->store('profile', 'public') parancs a képet a storage/app/public/profile könyvtárba menti a 'public' disk használatával.

A validálás során kapott ideiglenes fájl elérési útját lecseréljük a ténylegesen elmentett fájl elérési útjára a validált adatok tömbjében, mielőtt frissítenénk a felhasználói modellt.

A duplikált profilképek kezelésére implementáltunk egy logikát, ami a régi kép törlését végzi el az új feltöltése előtt a Storage::disk('public')->delete($user->image) metódussal. Végül a felhasználói modellben létrehoztunk egy getImageURL() metódust.

```php
public function update(UpdateUserRequest $request, User $user){
    $validated = $request->validated();

    if($request->has("image")){
        $imagePath = $request->file("image")->store("profile", "public");
        $validated['image'] = $imagePath;

        Storage::disk('public')->delete($user->image ?? '');
    }

    $user->update($validated);

    return redirect()->route("profile");
}
```

108. ábra Profilkép módosítási logikája kontrollerben

```php
public function getImageURL(){
    if($this->image){
        return url('storage/'. $this->image);
    }
    else{
        return "https://api.dicebear.com/6.x/fun-emoji/svg?seed=" . $this->name;
    }
}
```

109. ábra getImageURL() függvény

4.1.23. Follow, Unfollow

A felhasználóknak módja van arra, hogy egymást kövessék, ami a feed page-nél nyer jelentőséget, ugyanis itt csak a felhasználó által követett felhasználók ötletei jelennek meg. Ennek megvalósításához pedig szükség volt egy kapcsolótáblára felhasználó és felhasználó között.

Először is, létrehoztunk egy pivot táblát follower_user néven az adatbázisban a követők és a követettek közötti N:M kapcsolathoz. Ennek a táblának a létrehozásához egy migrációt generáltunk a php artisan make:migration create_follower_user_table --create paranccsal. A migrációs fájlban definiáltuk a tábla szerkezetét, amely két idegen kulcsot tartalmaz: user_id (a követett felhasználó azonosítója) és follower_id (a követő felhasználó azonosítója), mindkettő a users táblára mutat és cascadeOnDelete beállítással rendelkezik. A migrációt a php artisan migrate paranccsal futtattuk le.

```php
public function up()
{
    Schema::create('follower_user', function (Blueprint $table) {
        $table->foreignId('user_id')->constrained()->cascadeOnDelete();
        $table->foreignId('follower_id')->constrained('users', 'id')->cascadeOnDelete();
        $table->timestamps();
    });
}
```

110. ábra Pivot tábla migrációja

A User modellben definiáltuk az N:M kapcsolatot a belongsToMany() metódussal. Két metódust hoztunk létre: followings() (azok a felhasználók, akiket az aktuális felhasználó követ) és followers() (azok a felhasználók, akik az aktuális felhasználót követik). Mindkét metódusban megadtuk a kapcsolódó modellként a User::class-t, a pivot tábla nevét (follower_user), a külső pivot kulcsot (follower_id), és a kapcsolódó pivot kulcsot (user_id) a followings() metódusnál, illetve fordítva a followers() metódusnál. Emellett használtuk a withTimestamps() metódust is, hogy a pivot táblában rögzítésre kerüljön a kapcsolat létrehozásának és utolsó módosításának időpontja.

```php
public function followings(){
    return $this->belongsToMany(User::class, 'follower_user', 'follower_id','user_id')->withTimestamps();
}

public function followers(){
    return $this->belongsToMany(User::class, 'follower_user', 'user_id','follower_id')->withTimestamps();
}
```

111. ábra Követési kapcsolatok meghatározása a User modellben

Létrehoztunk útvonalakat a követéshez és a követés megszüntetéséhez a routes/web.php fájlban. Mindkettő egy POST metódusú útvonal lett definiálva: /users/{user}/follow a követéshez és /users/{user}/unfollow a követés megszüntetéséhez, ahol {user} a követni vagy a követésből eltávolítani kívánt felhasználó azonosítója. Mindkét útvonal a FollowerController-ben lévő follow() és unfollow() metódusokhoz van rendelve, és elnevezésre is kerültek (users.follow, users.unfollow).

Létrehoztunk egy FollowerController nevű vezérlőt a követés és követés megszüntetése logikájának kezelésére. A follow() metódus fogadja a követni kívánt User modellt (route model binding segítségével). A bejelentkezett felhasználót az auth() helperrel szerezzük meg. Az új követési kapcsolat létrehozásához a bejelentkezett felhasználó followings() kapcsolatán meghívjuk az attach() metódust, átadva a követni kívánt felhasználó modelljét. Sikeres követés esetén a felhasználót visszairányítjuk a követett felhasználó profiloldalára egy sikeres üzenettel.

```php
class FollowerController extends Controller
{
    public function follow(User $user)
    {
        $follower = auth()->user();

        $follower->followings()->syncWithoutDetaching($user); //->attach() allows duplicates

        return redirect()->route('users.show', $user->id)->with('success', 'Followed successfully!');
    }
    public function unfollow(User $user)
    {
        $follower = auth()->user();

        $follower->followings()->detach($user);

        return redirect()->route('users.show', $user->id)->with('success', 'Unfollowed successfully!');
    }
}
```

112. ábra Követési logika kontrollerben

A felhasználói profiloldalon (user-card.blade.php) implementáltuk a "Follow" gombot egy formon belül, amely a /users/{user}/follow útvonalra küld POST kérést a megfelelő felhasználói azonosítóval. A form tartalmazza a @csrf direktívát is.

Azt, hogy egy felhasználó követ-e egy másikat, egy follows() helper metódussal ellenőrizzük a User modellben. Ez a metódus a felhasználó followings() kapcsolatán keresztül lekérdezi, hogy létezik-e olyan bejegyzés, ahol a user_id megegyezik a vizsgált felhasználó azonosítójával az exists() metódussal.

```php
public function follows(User $user){
    return $this->followings()->where('user_id', $user->id)->exists();
}
```

113. ábra follows() metódus a User modellben

A "Follow" gombot dinamikusan váltogatjuk "Unfollow" gombra a Blade sablonban a follows() metódus eredménye alapján. Ha a bejelentkezett felhasználó követi a megtekintett profil tulajdonosát, akkor az "Unfollow" gomb jelenik meg, ellenkező esetben a "Follow" gomb.

A "Unfollow" gomb egy hasonló formban található, amely a /users/{user}/unfollow útvonalra küld POST kérést. Az unfollow() metódus a FollowerController-ben hasonlóan működik a

follow() metódushoz, de a kapcsolat létrehozása helyett a meglévő kapcsolatot távolítja el a detach() metódussal. Sikeres követés megszüntetése esetén a felhasználót visszairányítjuk a felhasználó profiloldalára egy sikeres üzenettel.
Ezáltal a felhasználók követhetik egymást a weboldalon, és bármikor megszüntethetik a követést.

4.1.24. Köszöntő Email

A projektben az email funkció a felhasználók regisztrációjakor egy üdvözlő email küldésére lett implementálva.

Először létrehozásra került egy "mailable class" objektum a php artisan make:mail WelcomeEmail parancs futtatásával a terminálban. Ez a parancs létrehoz egy új mappát mail néven az app könyvtáron belül, és benne egy WelcomeEmail.php fájlt.

Ez az email osztály kiterjeszti a Mailable osztályt, és alapértelmezetten három metódust tartalmaz:

envelope(): Ez a metódus tartalmazza az email fejlécét, mint például a tárgyat, a feladót (ha egyedi beállítás szükséges) stb.. A tárgy beállítására a $this->subject() metódus használható, és itt az alkalmazás nevét is beilleszthetjük a config('app.name') segítségével.

content(): Ebben a metódusban definiáljuk az email tartalmát, ami lehet HTML vagy Markdown. Itt megadhatunk egy Blade fájlt, amely tartalmazza az email kinézetét. A mi projektünkben egy mail nevű mappa található a views könyvtárban, és azon belül egy welcome-email.blade.php fájl.

A with() metódus segítségével adatokat adunk át a Blade sablonnak. Ehhez az email osztályban definiálnunk kell egy privát tulajdonságot (például $user), amelyet a konstruktorban kapunk meg és beállítunk. Így a Blade sablonban hozzáférhetünk az átadott adatokhoz (például $user->name).

attachments(): Ha csatolmányokat szeretnénk küldeni az emailhez, ebben a metódusban definiálhatjuk a fájlokat. Az Attachment::fromStorageDisk() metódus használható a public diszkről való csatoláshoz.

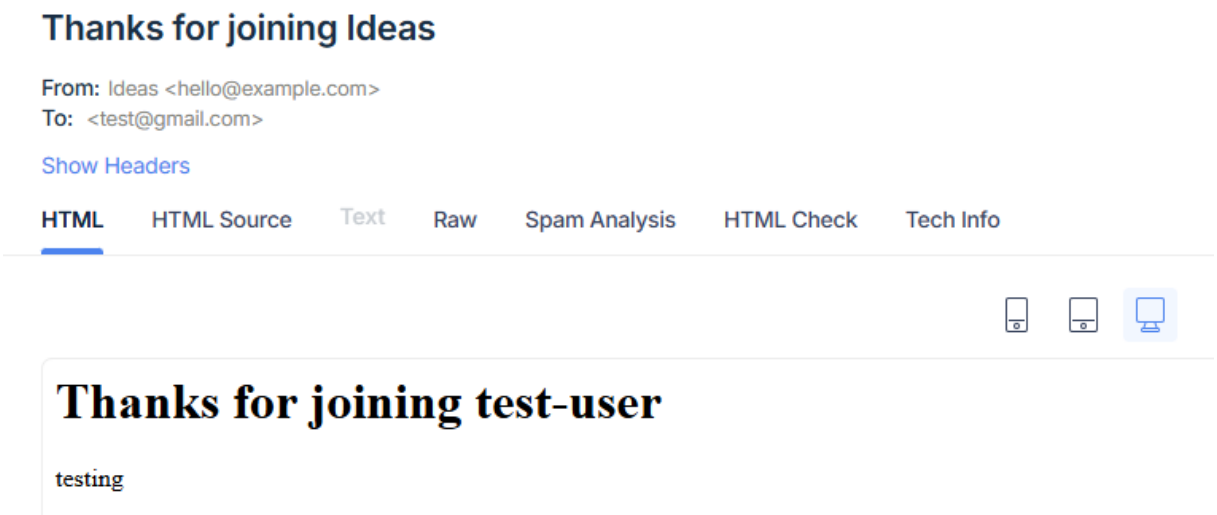
Az email elküldéséhez a Mail facade-ot használjuk az Illuminate\Support\Facades\Mail névtérből. A regisztrációt kezelő AuthController store() metódusában, a felhasználó létrehozása után hívjuk meg a Mail::to() metódust, megadva a címzett email címét ($user->email), majd a send() metódust, amelynek átadjuk az újonnan létrehozott email osztály egy példányát (new WelcomeEmail($user)). A WelcomeEmail konstruktora várja a felhasználó objektumot.

Az email küldéséhez konfigurálni kell a mail szerver beállításait a .env fájlban a MAIL_* kezdetű változók segítségével. Beállíthatjuk az SMTP szerver részleteit, a globális feladó címet és nevet.

Tesztelési célokra a Mailtrap nevű ingyenes weboldalt használtuk. A Mailtrap konfigurációs beállításait (host, port, username, password, encryption, from address) a Mailtrap weboldalán találhatjuk meg, és ezeket kell beállítani a .env fájlban.

Lehetőség van az emailek előnézetének megtekintésére is fejlesztési célból. A vezérlőben a return new WelcomeEmail($user); sorral közvetlenül visszaadhatjuk az email osztály egy példányát, és így a böngészőben láthatjuk a generált HTML-t.

Az emailhez csatolmányokat is hozzáadhatunk az attachments() metódusban a Attachment::fromStorageDisk('public', 'profile/fájlnév.png') módszerrel.



114. ábra Köszöntő email a Mailtrap-en belül

4.1.25. Like gomb

A projektben a like gomb funkció egy N:M-hez kapcsolatot hoz létre az ötletek (ideas) és a felhasználók (users) között.

Létrehoztunk egy idea_like pivot táblát egy migrációval. Ez a tábla két idegen kulcsot tartalmaz (user_id és idea_id), amelyek a users és ideas táblákra mutatnak, biztosítva az adatok integritását a constrained() és cascadeOnDelete() beállításokkal. A tábla tartalmazza továbbá a created_at és updated_at oszlopokat is.

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('idea_like', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()->cascadeOnDelete();
            $table->foreignId('idea_id')->constrained()->cascadeOnDelete();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('idea_like');
    }
};
```

115. ábra idea_like tábla migráció

Az Idea és a User modellekben definiáltunk egy likes() metódust, amely egy belongsToMany() kapcsolatot hoz létre egymással. A pivot tábla neve (idea_like) explicit módon meg lett adva, mivel nem követtük a Laravel névelési konvencióját (ami idea_user lenne). A withTimestamps() metódus biztosítja, hogy a pivot tábla created_at és updated_at oszlopai megfelelően működjenek.

Két új útvonalat definiáltunk a like és unlike műveletekhez:

POST /ideas/{idea}/like: Egy adott ötlet kedveléséhez.

POST /ideas/{idea}/unlike: Egy adott ötlet kedvelésének visszavonásához.

Mindkét útvonal az IdeaLikeController megfelelő metódusaira mutat. A {idea} egy route model binding segítségével automatikusan az Idea modell egy példányává alakul. A auth middleware biztosítja, hogy csak bejelentkezett felhasználók használhassák ezeket az útvonalakat.

Létrehoztunk egy IdeaLikeController-t két metódussal:

like(Idea $idea): Ez a metódus lekéri a bejelentkezett felhasználót (Auth::user()), majd a felhasználó likes() kapcsolatán keresztül meghívja az attach() metódust az adott $idea modellhez. Ez egy új bejegyzést hoz létre az idea_like pivot táblában. Végül a felhasználót visszairányítja a dashboardra egy sikerüzenettel.

```
public function like(Idea $idea){
    $liker = auth()->user();

    $liker->likes()->syncWithoutDetaching($idea);

    return redirect()->route('dashboard', $idea->id)->with('success', 'Liked
    successfully!');
}
```

116. ábra Kedvelési logika kontrollerben

unlike(Idea $idea): Ez a metódus hasonlóan lekéri a bejelentkezett felhasználót, majd a likes() kapcsolatán meghívja a detach() metódust az adott $idea modellhez. Ez eltávolítja a megfelelő bejegyzést az idea_like pivot táblából. A felhasználót ezután visszairányítja.

```
public function unlike(Idea $idea){
    $liker = auth()->user();

    $liker->likes()->detach($idea);

    return redirect()->route('dashboard', $idea->id)->with('success', 'Liked
    successfully!');
}
```

117. ábra Kedvelés megszüntetési logika kontrollerben

A like gomb egy külön like-button.blade.php nevű parciális nézetbe lett kihelyezve az ideas/shared mappában. Ez a nézet tartalmaz egy HTML formot, amely a /ideas/{$idea}/like vagy /ideas/{$idea}/unlike útvonalakra küld POST kérést. A gomb stílusa egy <button> elemmel van megvalósítva. A like gomb megjelenítése attól függ, hogy a bejelentkezett felhasználó már kedvelte-e az adott ötletet. A User modellben létrehoztunk egy likesIdea($idea) metódust, amely ellenőrzi, hogy a felhasználó rendelkezik-e a megadott ötlettel a likes() kapcsolatában. A Blade sablonban egy @if direktívával ellenőrizzük ezt, és ennek megfelelően jelenítjük meg a "like" vagy "unlike" gombot, valamint a hozzá tartozó szív ikont (üres vagy teli). Az ötlet kedveléseinek számát az $idea->likes()->count() kifejezéssel jelenítjük meg.

Ha egy nem bejelentkezett felhasználó látja az oldalt, a like gomb helyett csak a kedvelések száma jelenik meg egy linkkel a bejelentkező oldalra.

4.1.26. Feed page

A projektben a feed page egy olyan oldal, amely a bejelentkezett felhasználó által követett személyek ötleteit jeleníti meg. Ennek célja, hogy értelmet adjon a követés funkciónak, mivel korábban a főoldalon minden ötlet látható volt.

Egy új, feed nevű útvonal lett létrehozva a routes/web.php fájlban. Ez az útvonal megkapta az auth middleware-t, ami biztosítja, hogy csak bejelentkezett felhasználók érhessék el. Az útvonalhoz egy feed név is lett rendelve.

Létrehoztásra került egy FeedController nevű, invokable controller. Az invokable controller egy speciális típusú controller, amely csak egyetlen műveletet hajt végre, az __invoke() metóduson keresztül. Ez először lekéri a bejelentkezett felhasználót (Auth::user()). Ezután lekéri az azon felhasználók azonosítóit, akiket a bejelentkezett felhasználó követ. Ezt a following() kapcsolaton keresztül teszi meg a User modellen, majd a pluck('user_id') metódussal kinyeri a követett felhasználók azonosítóit. A following() kapcsolatnév a User modellben van definiálva. Végül lekérdezi az ideas táblát, és csak azokat az ötleteket hozza vissza, amelyeknek a user_id oszlopa szerepel a követett felhasználók azonosítóinak tömbjében. Ezt a whereIn() metódussal valósítja meg. A lekérdezés az latest() metódussal a legújabbak szerint van rendezve. Az eredményül kapott ötleteket átadja a nézetnek.

```php
class FeedController extends Controller
{
    /**
     * Handle the incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Contracts\View\View|\Illuminate\Contracts\View\Factory
     */
    public function __invoke(Request $request)
    {
        $user = auth()->user();
        $followingIDs = $user->followings()->pluck('user_id');

        $ideas = Idea::whereIn('user_id', $followingIDs)->latest();

        $ideas = $ideas->search($request->search);

        return view("dashboard", [
            'ideas'=> $ideas->paginate(5)
        ]);
    }
}
```

118. ábra invokable függvény a FeedControllerben

A navigációs sávban egy Feed menüpont található, amely a /feed útvonalra mutat. Ha a bejelentkezett felhasználó nem követ senkit, akkor a feed oldal nem fog ötleteket megjeleníteni. Ilyen esetben megjelenik egy szöveg "No results found" üzenettel.

Továbbá, a dátumok formázása a nézetekben a Carbon könyvtár diffForHumans() metódusával lett megváltoztatva, így a dátumok rövidebb és emberibb formában jelennek meg (pl. "22 minutes ago", "one month ago"). Ez a formázás a hozzászólások dátumán is megtörtént.

4.1.27. Admin page

A projektben az admin oldal egy speciális felület, amelyhez csak a megfelelő engedélyekkel rendelkező felhasználók (adminisztrátorok) férhetnek hozzá.

Létrehozástunk egy új útvonalat /admin címmel a routes/web.php fájlban. Ehhez az útvonalhoz egy admin előtagot is társult az útvonal névként. A nagyobb projektekben lehetőség lett volna külön fájlt létrehozni az admin útvonalak számára, de ebben a kisebb projektben az útvonal a web.php fájlban található.

Külön Admin mappa lett létrehozva a app/Http/Controllers könyvtárban az adminisztrációs funkciókhoz tartozó kontrollerek tárolására. Ebben a mappában található az Admin\DashboardController, amelynek egy index metódusa van. Ez a metódus egy egyszerű nézetet ad vissza.

Az AuthServiceProvider boot metódusában definiálásra került egy admin nevű Gate-et. Ez a Gate egy closure-t fogad el, amely ellenőrzi a bejelentkezett felhasználó is_admin értékét.

A can middleware használható az útvonalakon egy adott Gate ellenőrzésére. Például a /admin útvonalhoz hozzáadva a can:admin middleware-t, a rendszer ellenőrzi, hogy a felhasználó áthalad-e az admin Gate-en. Ha nem, egy 403 válasz kerül visszaküldésre.

A bejelentkezett adminisztrátorok számára a navigációs sávban elérhetővé válik egy ikon. Ez a link csak akkor jelenik meg, ha a bejelentkezett felhasználó is_admin oszlopa 1-re van állítva, és a /admin útvonalra mutat.

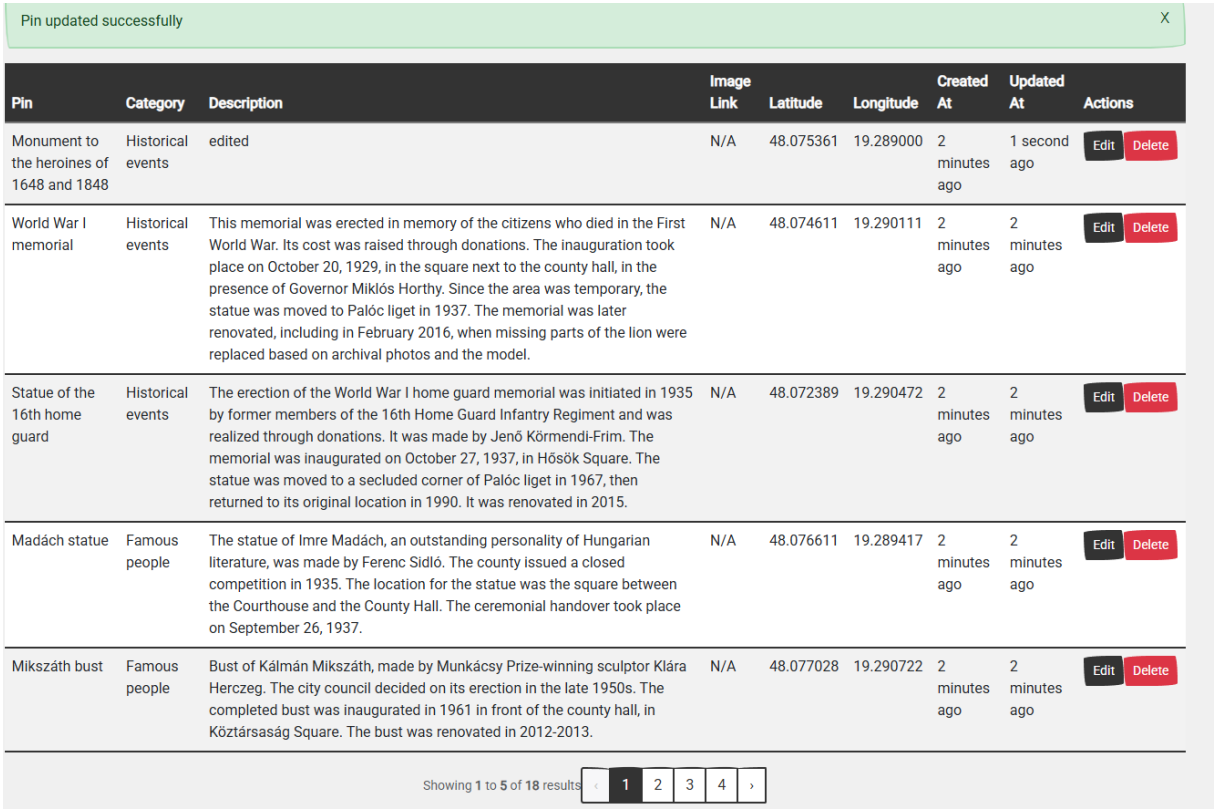
Ezen felül az admin dashboardon az adminoknak lehetőségük van bármelyik nevezetességet szerkeszteni és törölni, illetve az adatbázishoz új nevezetességet hozzáadni. A nevezetességek CRUD operációinak megvalósításának menete megegyezik az ötletekével, azon kívül, hogy külön megtekinteni egy nevezetességet nem lehet, hanem az összes nevezetesség egy táblázatban helyezkedik el, oldalanként mindig legfeljebb 5, és az oldal alján található egy lapozó.

```php
Route::prefix('admin')->group(function () {
    Route::get('/', [AdminDashboardController::class, 'index'])->middleware(['auth'])->can('admin')->name('admin.dashboard');
    Route::get('/add-pin', [AdminPinController::class, 'create'])->middleware(['auth'])->can('admin')->name('admin.pins.create');
    Route::post('/add-pin', [AdminPinController::class, 'store'])->middleware(['auth'])->can('admin')->name('admin.pins.store');
    Route::get('/pins/{pin}/edit', [AdminPinController::class, 'edit'])->middleware(['auth'])->can('admin')->name('admin.pins.edit');
    Route::put('/pins/{pin}', [AdminPinController::class, 'update'])->middleware(['auth'])->can('admin')->name('admin.pins.update');
    Route::delete('/pins/{pin}', [AdminPinController::class, 'destroy'])->middleware(['auth'])->can('admin')->name('admin.pins.destroy');
});
```

119. ábra Admin Gate-tel védett route-ok

```php
public function create()
{
    return view('admin.shared.add-pin', [
        'pinCategories' => PinCategory::all()
    ]);
}

public function store(CreatePinRequest $request)
{
    Pin::create($request->validated());

    return redirect()->route('admin.dashboard')->with('success', 'Pin created successfully');
}

public function edit(Pin $pin)
{
    return view('admin.shared.edit-pin', [
        'pin' => $pin,
        'pinCategories' => PinCategory::all()
    ]);
}

public function update(Pin $pin, UpdatePinRequest $request)
{
    $request->validated();

    $pin->update($request->validated());

    return redirect()->route('admin.dashboard')->with('success', 'Pin updated successfully');
}

public function destroy(Pin $pin)
{
    $pin->delete();
    return redirect()->route('admin.dashboard')->with('success', 'Pin deleted successfully');
}
```

120. ábra CRUD logika admin kontrollerben



121. ábra Nevezetesség módosítása

4.1.28. WithCount

A projektben a withCount() metódus a Laravel Eloquent ORM-ben egy hasznos funkció a kapcsolatokhoz tartozó rekordok számának hatékony lekérdezésére.

A withCount() metódus lehetővé teszi, hogy egy kapcsolatban lévő modellek számát eager loadingolja, amikor egy lekérdezést futtatunk egy adott modellre. Ez azt jelenti, hogy a Laravel egyetlen lekérdezéssel lekéri az eredeti modelleket és a kapcsolódó modellek számát is, elkerülve az úgynevezett "N+1 lekérdezés" problémát, amely akkor fordulhat elő, ha a kapcsolódó modellek számát külön-külön kérdezzük le minden egyes fő modellhez.

Mi a dashboard oldalon az ötletekhez tartozó kedvelések (likes) számának megjelenítésére használjuk. Tehát, amikor az ötleteket lekérdezik a dashboardhoz, a következőképpen használjuk a withCount() metódust:

A withCount('likes') hatására minden egyes Idea modellhez a rendszer hatékonyan lekérdezi a hozzá tartozó kedvelések számát.

A withCount() automatikusan hozzáad egy új attribútumot az eredeti modellekhez, amely tartalmazza a kapcsolódó modellek számát. Ennek az attribútumnak a névkonvenciója a kapcsolat neve, aláhúzással (_) és a count szóval képzett. A fenti példában tehát minden Idea

objektum rendelkezni fog egy likes_count attribútummal, amely megadja, hogy hány kedvelése van az adott ötletnek.

Ha egy adott oldalon (például egyetlen ötlet részletező oldalán) is szeretnénk megjeleníteni a kedvelések számát, akkor ott is külön le kell hívnunk a withCount('likes') metódust a lekérdezés során, mert a korábbi oldalon történt lekérdezéskor betöltött számláló nem lesz automatikusan elérhető egy másik lekérdezés eredményében.

4.1.29. Form Requestek

A form requestek a Laravelben lehetővé teszik a validációs szabályok elkülönítését a kontrollerektől egy saját, külön fájlba. Ezáltal a kontrollerek kevésbé lesznek terjedelmesek, különösen nagyobb projektekben, ahol sok validációs szabály létezhet.

Form requestet a PHP artisan make:request Artisan parancs segítségével hozhatunk létre. A parancs után meg kell adni a form request nevét. A Laravel közösségben elterjedt elnevezési konvenció a következő: a végrehajtott művelet neve (pl. Update), majd a modell neve (User), végül a Request szó (UpdateRequest). Például UpdateUserRequest.

A form requesteket alapértelmezés szerint az app/Http/Requests mappába hozza létre a Laravel. A névben mappákat is megadhatunk a perjel (/) használatával, így a form request egy almappába kerül létrehozásra (pl. User/UpdateUserRequest).
A form request felépítése: A létrehozott form request egy egyszerű osztály, amely a Illuminate\Foundation\Http\FormRequest osztályt terjeszti ki.

Alapértelmezés szerint két metódust tartalmaz: authorize() és rules(). A authorize() metódus azt ellenőrzi, hogy a felhasználó jogosult-e a kérés végrehajtására. Alapértelmezés szerint false értéket ad vissza, de ezt át kell állítani true-ra, vagy ide kell beírni a jogosultság-ellenőrző logikát. Ha ez a metódus false-t ad vissza, a kérés nem fogadható el.

A rules() metódusban kell megadni az összes validációs szabályt egy tömb formájában. A tömb kulcsai a validálandó mezők nevei, az értékei pedig a validációs szabályok láncolata (stringként vagy tömbként). Ugyanazok a validációs szabályok használhatók itt, mint a kontrollerekben a request()->validate() metódusban.

```php
class UpdateUserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return $this->user()->can('update', $this->user);
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, mixed>
     */
    public function rules()
    {
        return [
            "name"=> "required|min:3|max:40",
            "bio"=> "nullable|min:1|max:255",
            "image"=> "image",
        ];
    }
}
```

122. ábra UpdateUserRequest

Ahhoz, hogy egy form requestet használjunk egy kontroller metódusban, be kell típusoznunk a form request osztályt a metódus argumentumaként. A Laravel szolgáltatáskonténere automatikusan injektálja a form request objektumot a metódusba, amikor a kérés a routerről érkezik. Ügyelni kell arra, hogy a form request osztály importálva legyen a kontroller fájlba.

A form request objektumot általában $request néven érjük el a kontroller metódusban. A Laravel automatikusan lefuttatja a validációt, mielőtt a kontroller metódusa egyáltalán meghívásra kerülne. Ha a validáció sikertelen, a rendszer automatikusan visszairányítja a felhasználót az előző oldalra a validációs hibákkal. A validált adatok eléréséhez a form request objektumon a validated() metódust használhatjuk. Ez a metódus egy tömböt ad vissza a sikeresen validált mezőkkel és azok értékeivel. Ez a módszer hasonlóan működik a korábban használt request() helper validated() metódusához.

A form requestben a this->user() metódussal érhetjük el a bejelentkezett felhasználót. Útvonalmodell kötés esetén (route model binding), a modell példánya is elérhető a form requestben a $this kulcsszóval, a modell nevével (pl. $this->user).

Jogosultság-ellenőrzés a form requestben:

A authorize() metódusban elvégezhetjük a jogosultság-ellenőrzést is. Itt ellenőrizhetjük például, hogy a bejelentkezett felhasználó jogosult-e a kérésben szereplő modell frissítésére. A $this->user() segítségével hozzáférhetünk a bejelentkezett felhasználóhoz, és a route modell kötésnek köszönhetően a kapcsolódó modellhez is (pl. $this->user a frissítendő felhasználó esetén). Ha a authorize() metódus false-t ad vissza, egy 403-as (Forbidden) HTTP válasz kerül visszaküldésre.

Létrehoztunk form requesteket a felhasználói profil frissítéséhez (UpdateUserRequest), ötletek létrehozásához (CreateIdeaRequest), ötletek frissítéséhez (UpdateIdeaRequest) és kommentek létrehozásához (CreateCommentRequest). Minden esetben a validációs szabályokat áthelyeztük a kontrollerből a form request rules() metódusába, és a kontrollerekben a form request objektum típusmegadásával használtuk fel őket.

4.1.30. Személyre szabható oldalcímek

A custom page title-ok lehetővé teszik, hogy az alkalmazás minden egyes oldala különböző címmel rendelkezzen. A különböző oldalakra vonatkozó egyedi címek fontosak a SEO (keresőoptimalizálás) szempontjából, valamint a felhasználók számára is, amikor könyvjelzőznek egy oldalt, hogy lássák, miről szól az adott oldal.

Az app.blade.php fájlban a <title> tag-en belül egy @yield('title') direktívát kell elhelyezni. A @yield direktíva lehetővé teszi, hogy kódot, HTML-t vagy egy stringet injektáljunk a szülő Blade sablonba. Ebben az esetben a 'title' egy kulcsnév, amit a gyermek sablonokban fogunk használni. A cím után egy elválasztó karakter és a weboldal neve következik.

```
<title> @yield('title') | {{ config('app.name') }} </title>
```

123. ábra Személyre szabott oldalcím

@section direktíva a nézetekben: Azokon a nézeteken (Blade sablonokon), ahol egyedi címet szeretnénk beállítani, a @section('title', 'Egyedi Cím') direktívát kell használni. Ez a direktíva megadja, hogy a 'title' nevű @yield-be milyen tartalmat (a második argumentumban megadott stringet) illesszen be a layout fájlban.

4.1.31. Scope-ok

A scope-ok a Laravel Eloquent modellekben egy módszer a modell lekérdezéseinek refaktorálására, újra felhasználhatóvá és kifejezőbbé tételére.

Lehetővé teszik a gyakran használt lekérdezési logikák egy helyre gyűjtését, így elkerülhető a kód duplikációja a kontrollerekben és más helyeken. A projektünkben egy keresési funkció három sora duplikálva volt a feed és a dashboard kontrollerekben. Egy scope használatával a komplex where feltételek helyett egy egyszerűbb metódushívás jelenik meg a kontrollerben, így könnyebben érthetővé válik a lekérdezés célja.

A scope-okat a Laravel modelleken belül definiáljuk. A mi példánkban az Idea modellen hoztunk létre egy scope-ot a keresési funkcióhoz.

A scope-ok valójában egyszerű metódusok (függvények) a modellen belül, amelyek valamilyen lekérdezést alkalmaznak a meglévő lekérdezésre. A scope metódusok neve egy speciális formátumot követ: mindig a scope előtaggal kell kezdődnie, majd ezt követi a scope neve. Például a keresést végző scopeunk neve scopeSearch lesz.

Minden scope metódusnak egy kötelező argumentuma van, $query néven, amely az Eloquent query builder egy példánya. A Laravel automatikusan injektálja ezt az argumentumot a scope meghívásakor. Igény esetén a $query típusa megadható \Illuminate\Database\Eloquent\Builderként. A scope metódusnak nincs visszatérési típusa (void).

A scope metóduson belül a $query objektumon keresztül érhetjük el a modell összes lekérdezési metódusát, mint például where, orderBy stb..

A DashboardControllerben és a Feedcontrollerben található redundáns keresési lekérdezést áthelyeztük a scopeSearch metódusba. Fontos megjegyezni, hogy a scope-on belül közvetlenül nem lehet hozzáférni a $request objektumhoz. Ezért a szükséges paramétereket (például a keresési kulcsszót) argumentumként kell átadni a scope metódusnak. A példában a $search paraméter kerül hozzáadásra, és egy üres string az alapértelmezett értéke.

A scope-okat a modellek lekérdezéseinél hívhatjuk meg a scope neve (az scope előtag nélkül) alapján, mintha egy statikus metódus lenne.

A forrásban a feed és dashboard kontrollerekben a korábbi hosszú where feltétel helyett egyszerűen az $ideas->search($request->search) hívás jelenik meg. Ha a scope elfogad további argumentumokat, azokat a scope neve után zárójelben kell átadni. A példában a keresési kulcsszó kerül átadásra a search scope-nak. Ha nem adunk át értéket, az alapértelmezett üres string fog érvényesülni, azaz szűrés nélkül jelenítik meg a blade fájlok a posztokat.

4.1.32. View Composer

Időnként szükség lehet adatok megosztására az alkalmazás által renderelt összes nézettel. A mi projektünkben ékes példa erre a legtöbb oldal jobb kártyáján megtalálható top users funkció a legtöbb posztot megosztó felhasználók rangsorolására. Erre a célra a View facade share() metódusa használható, melynek hívásait általában egy szolgáltató (ServiceProvider) boot() metódusában kell elhelyezni.

Azonban mi nem a share() metódust használtuk, hanem a Nézet Komponensek (View Composers) nyújtotta lehetőséget. Ez abból fakad, hogy amikor Githubról lehúzzuk a projektet, a php artisan parancsok egyszerűen nem működnek, mivel amíg hiba van egy Laravel projektben, a parancsok sem futnak le. A hiba onnan adódott, hogy próbálta a likes_count mezőt a boot() metódus betölteni globális változó deklarálásához, úgy, hogy a migrációk még nem történtek meg, ezáltal patthelyzetbe kerültünk.

A nézetkomponensek lehetővé teszik adatok kötését a nézetekhez, mielőtt azok renderelésre kerülnének. A nézetkomponensek a szolgáltatói konténeren keresztül kerülnek feloldásra, így lehetőség van függőségek bevihetőségére a komponens konstruktorába.

A View::composer(["*"], TopUserComposer::class sor az AppServiceProvider boot() metódusában regisztrál egy nézetkomponenst. A composer() metódus első argumentuma (["*"]) azt jelenti, hogy ez a komponens az összes nézetre vonatkozik. A második argumentum (TopUserComposer::class) megadja a komponensként használandó osztályt.

Egy komponenshez tartozó osztálynak rendelkeznie kell egy compose() metódussal, amely megkapja a View példányát és a with() metódus segítségével köti az adatokat a nézethez.

Ebben a compose() metódusban a $view->with('topUsers', ...) hívás hozzáad egy topUsers nevű változót az összes nézet adataihoz. Ennek a változónak az értéke egy Eloquent lekérdezés eredménye, amely a felhasználókat (User modell) az ötleteik számával (withCount('ideas')) együtt kéri le, az ötletek száma szerint csökkenő sorrendben rendezi (orderBy('ideas_count', 'desc'), és az első 5 eredményt adja vissza (take(5)->get()).

Tehát a View Composer megközelítésével a TopUserComposer osztály compose() metódusa minden nézet renderelése előtt lefut, és a with() metódussal a topUsers változót elérhetővé teszi a nézetekben a megadott lekérdezés eredményével. Ezáltal a nézeteiben közvetlenül használható a {{ $topUsers }} Blade direktívával a legaktívabb felhasználók listájának megjelenítéséhez, anélkül hogy ezt a lekérdezést minden egyes kontrollerben külön-külön le kellene futtatni vagy manuálisan át kellene adni a nézeteknek.

4.1.33. API - Laravel Sanctum

Az API (alkalmazásprogramozási felület) olyan szoftveres megoldás, amely lehetővé teszi a különböző rendszerek vagy alkalmazások közötti adatcserét és kommunikációt. A mi projektünkben ez az adatcsere a weboldal és a mobilalkalmazás között történik.

Az API funkciói közé tartozik a regisztráció és a bejelentkezés tokenizált kezelése, a térképen megjelenítendő látványosságok adatainak biztosítása a mobilapp számára, valamint lehetőség egy felhasználó és egy látványosság adatbázison belüli összekötéséhez a mobilappon belül, amikor is a felhasználó megtekint egy nevezetességet.

A mobilalkalmazás specifikus API végpontokat használ a backend kommunikációhoz. A felhasználói hitelesítéshez kapcsolódó API hívások alap URL-je a https://bdk.teamorange.hu/api. A pinek lekérdezéséhez használt API végpont pedig a https://bdk.teamorange.hu/api/pins.

A mobilalkalmazás POST kéréseket küld a backend API-nak a felhasználó regisztrálásához és bejelentkezéséhez, valamint a felhasználó összekötéséhez egy nevezetességgel. Emellett GET kérést hajt végre a pinek adatainak lekérdezéséhez.

Az API kommunikáció során a felhasználók hitelesítésére tokeneket használnak. Ezek a tokenek a sikeres bejelentkezés után kerülnek vissza a mobilalkalmazásnak.

A tokenezést és az API hitelesítést a Laravel Sanctum kezeli. A Sanctum által kezelt API hozzáférési tokeneket a Laravel automatikusan generált personal_access_tokens táblában tárolja az adatbázisban.

Ez a tábla egy polimorf kapcsolaton keresztül kapcsolódik a token tulajdonosához (például a User modellhez), ami lehetővé teszi, hogy bármilyen olyan modellhez tartozhasson token, amely használja a HasApiTokens trait-et. A HasApiTokens trait beillesztése a modellbe (pl. a User modellbe) biztosítja a szükséges funkcionalitást a tokenek kezeléséhez a Sanctum segítségével.

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
```

124. ábra HasApiTokens

```php
if ($user && Hash::check($credentials['password'], $user->password)) {
    $user->tokens()->delete();
    $token = $user->createToken('auth-token')->plainTextToken;

    return response()->json([
        'token' => $token,
        "user" => $user
    ], 200);
}

return response()->json(['message' => 'Invalid credentials'], 401);
```

125. ábra Külsős regisztráció tokennel

4.1.34. Tesztelés

4.1.34.1. Beépített debugger

A Laravel keretrendszer beépített debuggerrel rendelkezik, amely nagy segítséget nyújt a backend tesztelése során, mivel azonnal jelzi, ha valamilyen probléma merül fel. Lehetőség van a debug felület manuális megjelenítésére is, amely kétféle módon történhet: egy kevésbé részletes és egy részletesebb menüvel. A dd() parancs a minimális debug felületet jeleníti meg, és egy változót átadva megvizsgálhatjuk annak tartalmát. Például a $request változók validálás előtti ellenőrzésére is használták ezt a módszert. A ddd() parancs a részletesebb debug menüt nyitja meg, és itt is szükséges a vizsgálni kívánt változó átadása.

4.1.34.2. Naplóüzenetek (Log Messages)

A Laravel alkalmazásokban keletkező naplóüzenetek központi helyen kerülnek tárolásra. Ha valamilyen hiba vagy esemény történik az alkalmazás futása során, a Laravel ezeket az információkat naplófájlokba írja. Ezek a naplófájlok a storage/logs könyvtárban találhatóak, és a fő naplófájl neve általában laravel.log.

A laravel.log fájl megtekintésével a fejlesztők nyomon követhetik az alkalmazás működését, felderíthetik a hibákat és debuggolhatják a problémákat. A naplófájlba kerülhetnek például:

Hibák és kivételek: Amikor az alkalmazásban nem várt esemény következik be, a hibaüzenetek és a kivétel részletei itt rögzülnek.

Figyelmeztetések: Olyan események, amelyek nem feltétlenül okoznak hibát, de érdemes lehet odafigyelni rájuk.

Információs üzenetek: Az alkalmazás normál működésével kapcsolatos információk, amelyek segíthetnek a működés megértésében.

Adatbázis lekérdezések: A Laravel Debugbar (egy külső csomag, nem része az alap Laravelnek) használatával akár az adatbázis lekérdezések is naplózhatóak, segítve a lassú vagy duplikált lekérdezések azonosítását.

4.1.34.3. Laravel Debugbar

A projektben a Laravel Debugbar egy rendkívül hasznos eszköz a fejlesztés során, amely segít az alkalmazás hibakeresésében és teljesítményének javításában. Lehetővé teszi számunkra, hogy betekintsünk az alkalmazás működésébe, beleértve a futtatott adatbázis-lekérdezéseket, a lassú lekérdezéseket és a duplikált lekérdezéseket is.

A Debugbar telepítéséhez a Composer csomagkezelőt használjuk. A terminálban, az alkalmazás gyökérkönyvtárában a következő parancsot kell futtatni: composer require barryvdh/laravel-debugbar --dev. A --dev flag biztosítja, hogy a csomag csak a fejlesztői környezetben kerüljön telepítésre, és ne a produkciós környezetben, ami biztonsági kockázatot jelenthetne.

A sikeres telepítés után a Debugbar automatikusan működik a helyi fejlesztői környezetben. Egy sáv jelenik meg az oldal alján, amely különböző menüpontokat tartalmaz. Ezt a sávot be lehet csukni vagy minimalizálni, és a Laravel logóra kattintva lehet újra megnyitni.

126. ábra Laravel Debugbar

A Debugbar különböző hasznos információkat jelenít meg, beleértve:

Timeline (Idővonal): Megmutatja, mennyi időt vett igénybe például az alkalmazás indítása.

Exceptions (Kivételek): Megjeleníti az esetlegesen fellépő kivételeket.

Views (Nézetek): Listázza a betöltött nézeteket.

Route (Útvonal): Megmutatja az aktuális útvonalat.

Queries (Lekérdezések): Ez a számunkra jelenleg legfontosabb rész, amely az adatbázis ikonnal van jelölve. Itt láthatjuk az összes futtatott adatbázis-lekérdezést. Jelenlegi példánkban látható lesz, hogy sok adatbázis-lekérdezés fut, és ezek között duplikációk is vannak (például 10 duplikált lekérdezés a felhasználó #2 betöltésére). Ez segít azonosítani azokat a területeket, ahol optimalizálásra van szükség.

Models (Modellek): Megmutatja a betöltött modellek számát (például 10 felhasználó modell, 5 hozzászólás és 5 ötlet).

Session (Munkamenet): Lehetővé teszi a munkamenet adatainak megtekintését.

Request (Kérelem): Megmutatja a beérkező kéréssel kapcsolatos információkat.

Memory (Memóriahasználat): Nyomon követhetjük az alkalmazás memóriahasználatát.

Response (Válaszidő): Megmutatja a válasz generálásának idejét.

Fontos megjegyezni, hogy a Debugbar láthatósága az APP_DEBUG konfigurációs beállítástól függ a .env fájlban. Ha APP_DEBUG értéke false, akkor a Debugbar nem fog megjelenni. Fejlesztés során érdemes ezt true-ra állítani a hibakereséshez. A telepítés során használt --dev flag egy további védelmet nyújt, hogy véletlenül se kerüljön fel a produkciós szerverre.

A lekérdezések részletes elemzésével azonosíthatjuk a duplikált és lassú lekérdezéseket, és lépéseket tehetünk a számuk csökkentésére.

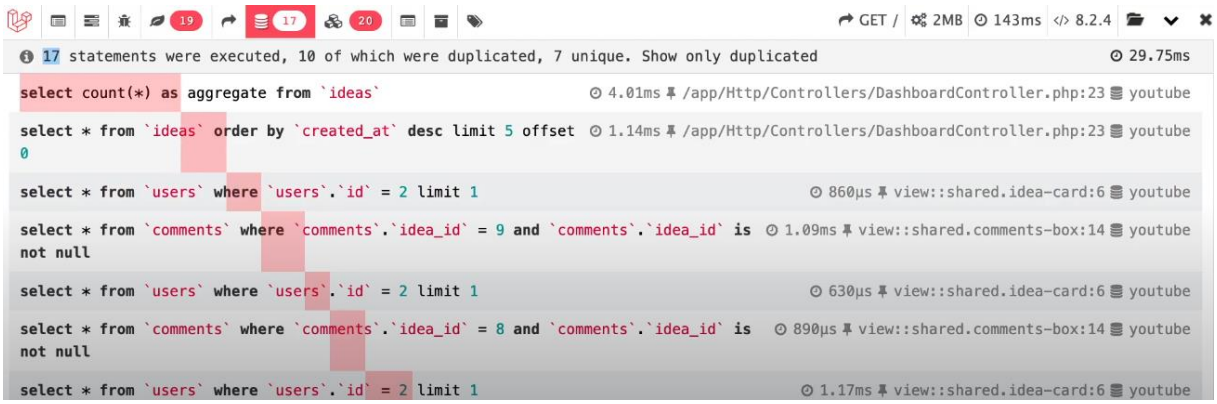
4.1.34.3.1. Eager loading

A projektben az eager loading egy olyan technika, amelyet a duplikált adatbázis-lekérdezések csökkentésére használunk. Amikor a projekt betölti a modelleket és azok kapcsolatait, eager loading használatával egyetlen kérésben tölthetjük be az összes szükséges kapcsolatot, ahelyett, hogy minden egyes kapcsolat elérésekor külön adatbázis-lekérdezést indítanánk. Ezt a jelenséget, amikor minden egyes fő modellhez külön lekérdezés fut a kapcsolódó adatokért, lazy loadingnak nevezzük, és nem hatékony.

A vezérlőben, ahol a modelleket betöltjük (például a dashboard controller-ben), a modell lekérdezésekor a with() metódust használtuk. Ennek a metódusnak átadtuk a betölteni kívánt kapcsolat nevét (ami a modellben van definiálva). Például, ha egy Idea modellnek van egy user és egy comments kapcsolata, akkor ezeket eager loadinggal betölthetjük így: Idea::with('user', 'comments')->get(). Ez azt jelenti, hogy az ötletekkel együtt a hozzájuk tartozó felhasználók és hozzászólások is egyetlen lekérdezéssel kerülnek betöltésre.

A with() metódus elfogad egy stringet, egy string tömböt vagy egy asszociatív tömböt is a betöltendő kapcsolatok megadásához.

A Debugbar segítségével korábban 17 adatbázis-lekérdezést láttunk, amelyek közül 10 duplikált volt, különösen a felhasználó #2 betöltésekor.



127. ábra 17 query (lazy loading)

Az Idea modellnél a user kapcsolat eager loadingjának bevezetése után a lekérdezések száma 13-ra csökkent. Ennek oka, hogy korábban minden egyes ötlethez külön lekérdezés futott a felhasználó adatainak lekéréséhez (ha öt ötlet volt egy oldalon, akkor öt lekérdezés a felhasználókra), míg eager loadinggal ez egyetlen lekérdezés lett.

```
$ideas = Idea::with('user')->orderBy('created_at','DESC');
```

128. ábra With függvény eager loading végett

A comments kapcsolat eager loadingjának hozzáadásával a lekérdezések száma tovább csökkent, 9-re. Ekkor az összes ötlethez tartozó hozzászólás egyetlen lekérdezéssel került betöltésre az IN operátor használatával az ötletek azonosítóival.

```
select * from `comments` where `comments`.`idea_id` = 9 and `comments`.`idea_id` not null
select * from `comments` where `comments`.`idea_id` = 8 and `comments`.`idea_id` not null
select * from `comments` where `comments`.`idea_id` = 7 and `comments`.`idea_id` not null
```

129. ábra Comment query-k eager loading előtt

Így a duplikált lekérdezések már csak egyszer, egy queryként futnak le, maradtunk 10 queryvel:

```
select * from `comments` where `comments`.`idea_id` in (9, 10, 11, 12, 13)
```

130. ábra Comment query-k eager loading után

Az eager loadingot a modellben is definiálhatjuk a $with védett tulajdonság használatával. Az $with tömbben megadott kapcsolatok automatikusan betöltődnek a modell lekérdezésekor. Például az Idea modellben a protected $with = ['user']; beállításával a user kapcsolat mindig eager loadinggal lesz betöltve. Azonban erre óvatosan kell figyelni, mert ha

egy oldalon nem használjuk ezt a kapcsolatot, akkor is le fog futni a lekérdezés. Erre a problémára kínál megoldást a without() metódus, amellyel egy adott lekérdezésnél kikapcsolhatjuk az $with tulajdonságban definiált eager loadingot.

Továbbá, az eager loading során megadhatjuk, hogy a kapcsolódó modellekből mely oszlopokat szeretnénk betölteni a kettőspont (:) használatával. Például az with('user:id,name,image') megadásával csak az id, name és image oszlopok kerülnek betöltésre a users táblából. Ez memóriát takaríthat meg, különösen nagy táblák esetén.

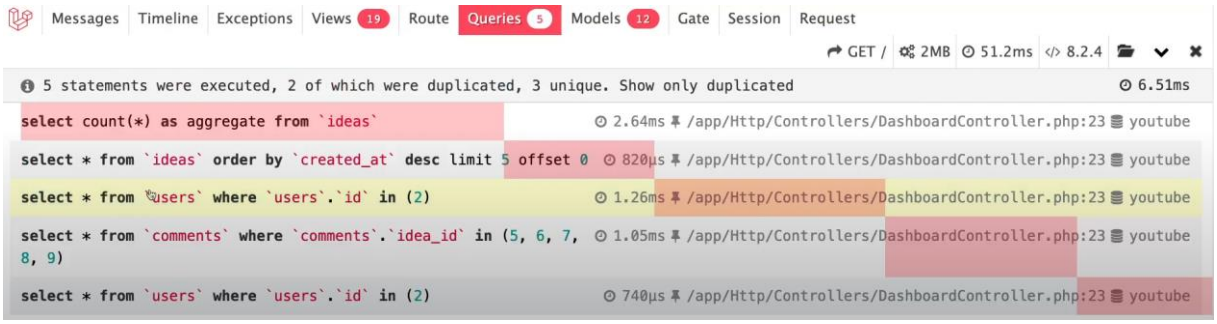
```php
class Idea extends Model
{
    use HasFactory;

    protected $with = ['user:id,name,image', 'comments.user:id,name,image'];

    protected $withCount = ['likes'];
```

131. ábra Extra oszlopok betöltésének definiálása modellben

Ebből kifolyólag a likeokat mutató view fileban is átírjuk a logikát, ami eddig külön queryként kezelte mindegyik ötletnek a kedveléseinek számának lekérdezését, most viszont, hogy az Idea modellben meghatároztuk, hogy a likeok számával töltse be, a queryk száma lecsökkent a maradék 10-ről 5-re.

```
            </span> {{ $idea->likes()->count() }} </button>
    </form>
@else
    <form action="{{ route('ideas.like', $idea->id) }}" method="POST">
        @csrf
        <button type="submit" class="fw-light nav-link fs-6"> <span class="far fa-heart me-1">
            </span> {{ $idea->likes()->count() }} </button>
    </form>
@endif
@auth
@guest
    <a href="{{ route('login') }}" class="fw-light nav-link fs-6"> <span class="far fa-heart me-1">
        </span> {{ $idea->likes()->count() }} </a>
```

132. ábra Felhasználó statisztikák lekérdezése eager loadinggal



133. ábra 5 query (eager loading)

Mindennek eredményeként a példában a lekérdezések száma így 17-ről 5-re csökkent.

4.1.34.4. PHPUnit

A szerveroldali működés ellenőrzéséhez automatikus tesztelést alkalmazunk, melyhez a Laravel beépített PHPUnit támogatását használjuk. Ehhez két funkcionális tesztet készítettünk, amelyek az adminisztrációs felület elérhetőségét vizsgálják különböző jogosultságú felhasználók esetében.

A példában az adminisztrációs felülethez való hozzáférést vizsgálunk két eltérő jogosultságú felhasználóval. Létrehoztunk a php artisan make:test AdminDashboardTest parancs segítségével a tests/Feature mappába egy AdminDashboardTest.php elnevezésű tesztfájlt. A hozzáférést a felhasználó is_admin mezője határozza meg, mely binárisan jelöli, hogy a felhasználó adminisztrátori szerepkört tölt-e be.

Az első tesztesetben létrehozunk egy olyan felhasználót, aki adminisztrátori jogosultsággal rendelkezik. A rendszerbe bejelentkezve az adott felhasználó megpróbálja elérni az adminisztrációs vezérlőpultot. Az elvárt eredmény egy 200-as HTTP státuszkód, valamint az, hogy a megfelelő nézet (admin.dashboard) jelenik meg.

```php
/** @test */
public function admin_can_access_admin_dashboard()
{
    $adminUser = User::factory()->create(['is_admin' => 1]);

    $response = $this->actingAs($adminUser)->get(route('admin.dashboard'));

    $response->assertStatus(200);
    $response->assertViewIs('admin.dashboard');
}
```

134. ábra PHPUnit tesztfüggvény I.

A második teszteset egy nem adminisztrátori jogosultságú felhasználót szimulál. Az ilyen felhasználó próbálkozik az adminisztrációs felület elérésével, de mivel nem rendelkezik megfelelő jogosultsággal, a rendszer megtagadja a hozzáférést, és 403-as státuszkódot ad vissza.

```php
/** @test */
public function regular_user_cannot_access_admin_dashboard()
{
    $regularUser = User::factory()->create(['is_admin' => 0]);

    $response = $this->actingAs($regularUser)->get(route('admin.dashboard'));

    $response->assertStatus(403);
}
```

135. ábra PHPUnit tesztfüggvény II.

A tesztek futtatásakor a php artisan test parancsot felhasználva lefuttatjuk az összes definiált tesztet. Amennyiben minden teszt sikeresen lefut, a rendszer egy megerősítő üzenettel jelzi, hogy az alkalmazás a vártnak megfelelően működik.

```
PS C:\Users\User\Downloads\echomap_website> php artisan test
Warning: TTY mode is not supported on Windows platform.

  PASS  Tests\Feature\AdminDashboardTest
  ✓ admin can access admin dashboard
  ✓ regular user cannot access admin dashboard

  Tests:  2 passed
  Time:   1.63s
```

136. ábra Tesztek lefuttatása

4.1.34.4.1. RefreshDatabase

A RefreshDatabase trait használata lehetővé teszi, hogy minden egyes teszt futtatása előtt egy friss, tiszta adatbázis álljon rendelkezésre. A trait lefuttatja az összes migrációt a teszt elején, majd a teszt végén automatikusan visszaállítja az adatbázist az eredeti, üres állapotra. Ennek köszönhetően a tesztek teljesen izoláltan futnak, és biztosak lehetünk benne, hogy egy adott teszt eredménye nem függ más tesztek által hátrahagyott adatoktól. Ez különösen fontos, ha több teszt párhuzamosan fut, vagy ha egymás után futtatunk sok tesztet különféle körülmények között.

A modellek példányosítására két fő lehetőség van: a make() és a create() metódusok. A make() segítségével csak egy modell példányt hozunk létre memóriában, de ez az objektum nem kerül be az adatbázisba. Ezzel szemben a create() egy új rekordot is ment az adatbázisba, így az a Laravel által teljes értékű adatként kezelhető. A make() használata gyorsabb lehet, ha nincs szükségünk tényleges adatbázis-műveletekre, például ha csak egy modell logikáját akarjuk tesztelni vagy dummy adatot használunk valamilyen függőség kielégítésére.

Fontos azonban, hogy bizonyos Laravel funkciók – például az autentikáció (actingAs()), middleware-ek, route védelem, vagy policy-k – csak olyan modellekkel működnek

megfelelően, amelyek valóban szerepelnek az adatbázisban. Ebben az esetben tehát tesztelés során nem elég a make() használata, mivel az így létrehozott objektum nem fog megfelelni például az autentikációs guardoknak, amelyek lekérdezik az adatbázisból a bejelentkezett felhasználót. Ezért minden olyan tesztben, ahol hitelesített felhasználóval dolgozunk, vagy valamilyen adatbázis-alapú ellenőrzés történik, a create() metódust kell használnunk.

4.2. Mobilalkalmazás

A mobilalkalmazás tervezését és fejlesztését Refka Bence végezte, fejlesztéséhez a Unity játékmotor került felhasználásra. A Unity egy széles körben használt, erőteljes eszköz, amely lehetővé teszi a multiplatform fejlesztést, így az alkalmazás Android és iOS rendszereken is futtatható. A fejlesztés C# programozási nyelven valósult meg, mivel ez a Unity alapértelmezett nyelve, és lehetőséget biztosít a hatékony és strukturált kódolásra.

További előnyt jelentett, hogy a Unity a C# programozási nyelvet alkalmazza, amely már több éve a tanmenet részét képezi, így nem volt szükség új nyelv elsajátítására, ami jelentősen megkönnyítette a fejlesztés menetét.

Az alkalmazás futtatásához a mind a .NET keretrendszer, mind a Unity fejlesztői eszköz telepítése szükséges. A Unity projektben két fő mappa található: Assets és Packages. Az Assets mappa tartalmazza az összes projektfájlt, amelyek további almappákba vannak rendezve, például Scenes, Prefabs stb. Ezekben a mappákban helyezkednek el a jelenetek, kódfájlok, képek, előre beállított objektumok és egyéb szükséges erőforrások.

4.2.1. Unity UI Rounded Corners Package

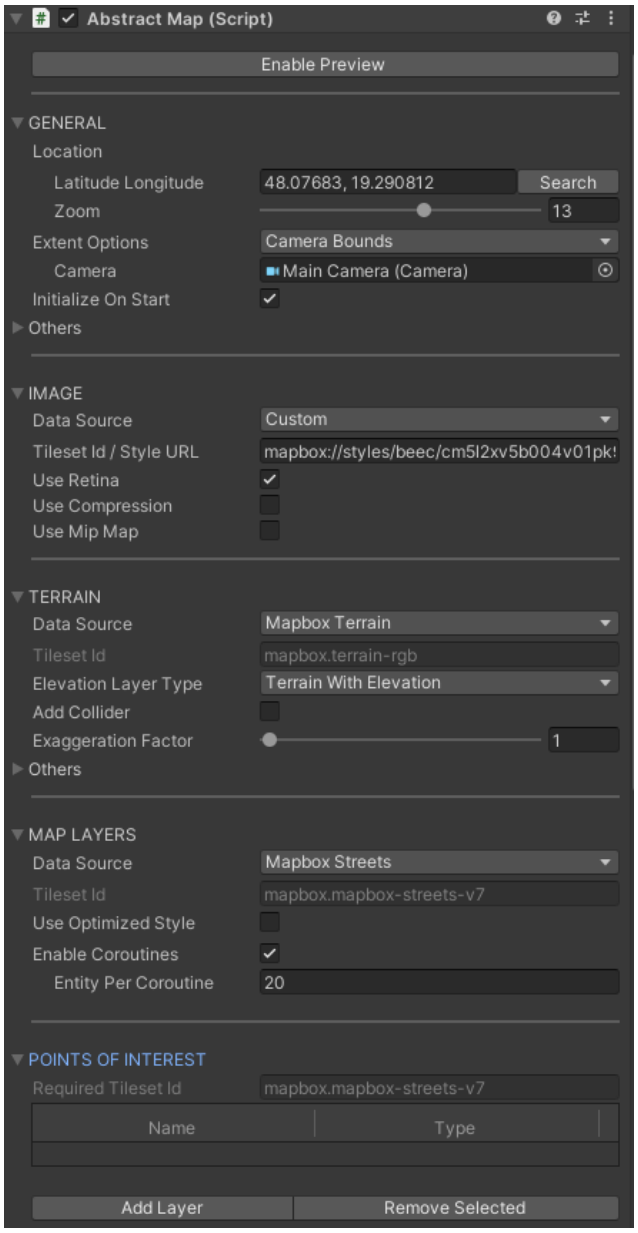
A felhasználói felület kialakításához a Unity UI Rounded Corners csomag került felhasználásra, mivel a Unity alapértelmezett UI rendszere nem biztosít natív lehetőséget a képek alakjának és sarkainak egyszerű módosítására.

Ebben a csomag található két Script, ami lehetővé teszi, hogy a képeket lekerekített sarkakkal és különböző formázási lehetőségekkel jelenítsük meg, így modernebb megjelenést érhetünk el az alkalmazásban.

4.2.2. Mapbox SDK (for Unity)

A térképes megjelenítéshez a Mapbox SDK (for Unity) csomagot használja a projektl, amely lehetőséget biztosít interaktív és testreszabható térképek megjelenítésére. A Unity alapértelmezett rendszere nem tartalmaz natív térképes megoldásokat, ezért a Mapbox SDK ideális választás volt. Ez a csomag támogatja a nagyítható, mozgatható térképeket, valamint lehetőséget biztosít egyedi jelölők és információs panelek hozzáadására, így könnyedén megvalósíthatóvá vált Balassagyarmat történelmi helyszíneinek megjelenítése az alkalmazásban.

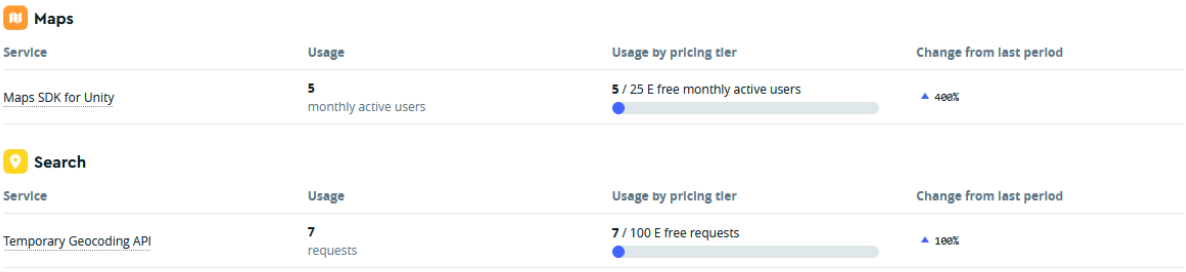
Ebben a csomagban található az Abstract Map Script, ami felelős a térkép betöltésére. A Scipt úgy van megírva, hogy a Unity-ben az Inspector-on keresztül be lehet állítani neki szinte mindent, ami lényeges.



137. ábra Abstract Map Script

A kamera pozíciója koordináták megadásával került beállításra, hogy Balassagyarmat középpontjára fókuszáljon. A térkép megjelenéséhez használt Tile style módosítása a Mapbox felületén történt, egy alapértelmezett stílus testreszabásával a projekt igényeinek megfelelően.

A térképet viszont csak akkor tudja betölteni, ha meg van adva neki egy érvényes Access Token, amit a Mapbox oldaláról lehet szerezni ingyenesen is. Viszont az ingyenes verzió által szerzett Access Tokennel limitált lekéréseket lehet lekérni minden hónapban.



138. ábra Access Tokenek

4.2.3. DB Manager

A DB_Manager Script egy Laravel backend API-val kommunikál, amely hitelesítést és felhasználói adatkezelést biztosít.

Az API elérési útvonalát egy statikus változóban tárolja.

```
private readonly static string API_URL = "https://bdk.teamorange.hu/api";
```

A RegisterUser metódus regisztrál egy új felhasználót a Laravel API-n keresztül.

Létrehozunk egy objektumot a regisztrációs adatokkal, amely tartalmazza a felhasználó nevét, e-mail címét és jelszavát.

```
var registerData = new RegisterRequest
{
        name = username,
        email = email,
        password = password,
};
```

Elküldjük a POST kérést az API-nak, amely a SendPostRequest függvényen keresztül hajtódik végre.

```
yield return SendPostRequest(url, jsonData, (success, response) =>
```

Ha a kérés sikeres, megpróbáljuk feldolgozni a válaszként kapott JSON adatokat és eltárolni a felhasználó tokent, és ha a JSON feldolgozás sikertelen (pl. rossz formátum vagy váratlan válasz), akkor hibaüzenetet írunk ki a konzolra.

```
if (success)
{
        try
        {
        var loginResponse = JsonHelper.ParseJson<LoginResponse>(response);
        authToken = loginResponse.token;
```

```
        callback(true, loginResponse.user.name);
        }
        catch (Exception e)
        {
        Debug.LogError($"Failed to parse registration response: {e.Message}\nResponse: {response}");
        callback(false, GetErrorMessage(response));
        }
}
```

A LoginUser metódus végrehajtja a felhasználó bejelentkezését a Laravel API-n keresztül. Létrehozunk egy objektumot a bejelentkezési adatokkal, amely tartalmazza a felhasználó e-mail címét és jelszavát.

```
var loginData = new LoginRequest
{
        email = email,
        password = password
};
```

Elküldjük a POST kérést az API-nak a SendPostRequest függvény segítségével.

```
yield return SendPostRequest(url, jsonData, (success, response) =>
```

Ha a kérés sikeres, a JSON válaszban kapott adatokat feldolgozza, és eltárolja a felhasználó tokent. A felhasználó fontosabb adatait különböző PlayerPref-ekben tárolja. Ha a JSON feldolgozás sikertelen (pl. rossz formátum vagy váratlan válasz), akkor hibaüzenetet ír ki a konzolra.

```
if (success)
{
   try
   {
      var loginResponse = JsonHelper.ParseJson<LoginResponse>(response);
      authToken = loginResponse.token;
      PlayerPrefs.SetInt("UserID", loginResponse.user.id);
      PlayerPrefs.SetString("UserToken", authToken);
      PlayerPrefs.SetString("UserCreatedAt", loginResponse.user.created_at);
      PlayerPrefs.Save();
      callback(true, loginResponse.user.name);
```

```
    }
    catch (Exception e)
    {
        Debug.LogError($"Failed to parse login response: {e.Message}\nResponse: {response}");
        callback(false, GetErrorMessage(response));
    }
}
```

A SendPostRequest metódus végrehajt egy HTTP POST kérést az API felé. Létrehozunk egy UnityWebRequest objektumot, amely a megadott URL-re küld POST kérést a JSON adatokkal.

```
using (UnityWebRequest request = new UnityWebRequest(url, "POST"))
```

A kérés törzséhez (body) hozzáadjuk a JSON adatokat és beállítjuk a tartalom típusát application/json-ra.

```
byte[] bodyRaw = Encoding.UTF8.GetBytes(jsonData);
request.uploadHandler = new UploadHandlerRaw(bodyRaw);
request.downloadHandler = new DownloadHandlerBuffer();
request.SetRequestHeader("Content-Type", "application/json");
```

4.2.4. RegisterManager

A RegisterManager script egy regisztrációs rendszert kezel, amely ellenőrzi a felhasználó által megadott adatokat, majd elküldi azokat egy adatbázisba aszinkron módon.

Az IsValidEmail metódus ellenőrzi, hogy az e-mail cím formailag helyes-e egy reguláris kifejezés (Regex) segítségével.

```
private bool IsValidEmail(string email)
{
        string pattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";
        return Regex.IsMatch(email, pattern);
}
```

Miután mindent helyesen adott meg, akkor regisztrálódik az adatbázisba.

```
private IEnumerator RegisterUserCoroutine(string email, string username, string password)
{
    yield return StartCoroutine(DB_Manager.RegisterUser(email, username, password,
(success, message) =>
    {
```

```
        if (success)
        {
            Popup.gameObject.SetActive(true);
            PopupText.text = $"Successfully registered!\nWelcome {username}!";
        }
        else
        {
            ShowError($"Registration failed: {message}");
        }
    }));
}
```

4.2.5. LoginManager

A LoginManager Script egy bejelentkezési rendszert kezel. Az e-mail és jelszó ellenőrzése után elküldi az adatokat az adatbázisba.

A bejelentkezett felhasználó nevét el menti a PlayerPrefs-ben. Addig amíg a felhasználó nem lép ki a Logout gombbal, addig az applikáció megjegyzi, hogy be van jelentkezve.

```
if (success)
{
        PlayerPrefs.SetString("LoggedInUser", username);
        PlayerPrefs.Save();
        Popup.gameObject.SetActive(true);
        PopupText.text = $"Successfully logged in as: {username}";
}
```

4.2.6. PinManager

A PinManager osztály felelős a térképre elhelyezett pontok (pinek) adatainak lekérdezéséért és kezeléséért a Unity-ben.

Az API elérési útvonalát egy privát példányváltozóban tároljuk, amely tartalmazza a térképi tűzők lekérésére szolgáló URL-t.

```
private string PIN_URL = "https://bdk.teamorange.hu/api/pins";
```

A Pin osztály egy térképen megjelenített pont (pin) adatait tárolja. Ez tartalmazza a pin azonosítóját, nevét, leírását, a hozzá tartozó kép URL-jét, a földrajzi koordinátáit (szélesség és hosszúság), valamint a pin kategóriáját, amelyet egy külön Category osztály reprezentál.

```
[System.Serializable]
public class Pin
{
        public int id;
        public string pin_name;
        public string pin_description;
        public string image_link;
        public float latitude;
        public float longitude;
        public Category pin_category;
}
```

A Category osztály külön kezeli a pin-ek kategóriájának adatait, így egy pin-hez nem csak egy egyszerű kategória-azonosító tartozik, hanem annak neve és színe is. Ez megkönnyíti a kategóriákhoz tartozó vizuális megjelenítést és szűrést a felhasználói felületen.

```
[System.Serializable]
public class Category
{
        public int id;
        public string category_name;
        public string category_color;
}
```

A PinList osztály egy olyan gyűjtőosztály, amely egy pins nevű listában tartalmazza az összes lekért pin-t. Ez az osztály hasznos az API válaszának feldolgozásakor, ahol több pin adat érkezik egyidejűleg.

```
[System.Serializable]
public class PinList
{
        public List<Pin> pins;
}
```

A GetPinData coroutine egy HTTP GET kérést küld a PIN_URL címre, ha a válasz sikeres, akkor a JSON szöveget PinList objektummá alakítja.

```
IEnumerator GetPinData()
{
```

```
        using (UnityWebRequest request = UnityWebRequest.Get(PIN_URL))
        {
        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.Success)
        {
        string jsonResponse = request.downloadHandler.text;
        PinListpinListResponse =
JsonUtility.FromJson<PinList>($"{{\"pins\":{jsonResponse}}}");
```

Az adatok sikeres lekérése után a JSON objektumból kiolvassuk a pinek adatait, és eltároljuk különböző listákban. A pinLocationList például a pinek földrajzi koordinátáit tárolja szövegként.

```
foreach (Pin pin in pinListResponse.pins)
{
        idList.Add(pin.id);
        pinNameList.Add(pin.pin_name);
        pinDescList.Add(pin.pin_description);
        imageLinksList.Add(pin.image_link);
        pinLocationList.Add($"{pin.latitude.ToString().Replace(",",
".")},{pin.longitude.ToString().Replace(",", ".")}");
        pinCategoryList.Add(pin.pin_category);
        pinList.Add(pin);
}
```

A térképre történő megjelenítéshez a különböző adatokat átadjuk a SpawnOnMap Script-nek.

```
_spawnOnMap.SetLocationStrings(pinLocationList.ToArray());
_spawnOnMap.SetPinName(pinNameList.ToArray());
_spawnOnMap.SetDescription(pinDescList.ToArray());
StartCoroutine(_spawnOnMap.SetImage(imageLinksList.ToArray()));
_spawnOnMap.SetColor(pinCategoryList.ToArray());
_sideMenu.CategorySetter(pinCategoryList.ToArray());

_spawnOnMap.SetOnDescCoords(pinLocationList.ToArray(), pinNameList.ToArray(),
pinCategoryList.ToArray());
```

A GetCoordinatesByPinName(string pinName) metódus segítségével egy adott pin nevéhez tartozó földrajzi koordinátákat kapunk vissza.

```
public (float latitude, float longitude) GetCoordinatesByPinName(string pinName)
{
        Pin pin = pinList.Find(p => p.pin_name.Equals(pinName, System.StringComparison.OrdinalIgnoreCase));
        if (pin != null)
        {
        return (pin.latitude, pin.longitude);
        }
        return (0f, 0f);
}
```

4.2.7. CheckedPinManager

A CheckedPinManager script kezeli a felhasználó által "kipipált" (meglátogatott) pinek státuszát a térképen. Lehetővé teszi egy adott pin „bejelölését” gombnyomásra, és megjeleníti, hogy az adott helyet már meglátogatta-e a felhasználó. A státusz a pin nevétől függően kerül meghatározásra, amelyet a főmenüben listázott pin-adatokból (PinManager) vesz át.

A CheckUserPins() egy HTTP GET kérést küld a szerver felé, hogy lekérje a felhasználó által már bejelölt pin-ek listáját. A lekért adatokban megkeresi, hogy az aktuális pinId szerepel-e a listában. Ha szerepel, akkor meghívja a SetCompletedState() metódust, amely vizuálisan zöldre színezi a pin nevét, és letiltja a pipálás gombot. Ha nem szerepel vagy hiba történik, a SetNotCompletedState() metódus fut le, ami visszaállítja az alap (fekete színű) állapotot és engedélyezi a gombot.

```
IEnumerator CheckUserPins()
{
  UnityWebRequest request = UnityWebRequest.Get(getURL);
  request.SetRequestHeader("Authorization", $"Bearer {token}");
  yield return request.SendWebRequest();
  if (request.result == UnityWebRequest.Result.Success)
  {
    string json = request.downloadHandler.text;
    PinIdList response = JsonUtility.FromJson<PinIdList>(json);
    if (response.pin_ids.Contains(pinId))
```

```
        {
            SetCompletedState();
        }
        else
        {
            SetNotCompletedState();
        }
    }
    else
    {
        Debug.LogError("GET request failed: " + request.error);
        SetNotCompletedState();
    }
}
```

A SendPinData() egy HTTP POST kérést küld a szerverre az aktuális pin ID-val, jelezve, hogy a felhasználó „meglátogatta” azt. A küldött adat JSON formátumban tartalmazza az id-t egy tömbben. Sikeres küldés esetén a PlayerPrefs-be menti, hogy a pint már bejelölte, és frissíti a vizuális állapotot zöldre. Sikertelen küldés esetén hibát ír ki a konzolra. A PinPostPayload egy osztály, amely a szervernek küldött JSON adatstruktúrát írja le (pin_id tömb).

```
IEnumerator SendPinData()
{
    PinPostPayload payload = new PinPostPayload
    {
        pin_id = new int[] { pinId }
    };

    string json = JsonUtility.ToJson(payload);
    UnityWebRequest request = new UnityWebRequest(postURL, "POST");
    byte[] body = System.Text.Encoding.UTF8.GetBytes(json);
    request.uploadHandler = new UploadHandlerRaw(body);
    request.downloadHandler = new DownloadHandlerBuffer();
    request.SetRequestHeader("Content-Type", "application/json");
    request.SetRequestHeader("Authorization", $"Bearer {token}");
```

```
    yield return request.SendWebRequest();

    if (request.result == UnityWebRequest.Result.Success)
    {
        PlayerPrefs.SetInt($"pin_done_{pinId}", 1);
        PlayerPrefs.Save();
        SetCompletedState();
    }
    else
    {
        Debug.LogError("Upload failed: " + request.error);
    }
}
```

A SetCompletedState() a pin nevét zöld színűre állítja (Color.green), a Check gombot inaktívvá teszi (interactable = false).

A SetNotCompletedState() a pin nevét fekete színűre állítja (Color.black), a Check gombot újra aktívvá teszi (interactable = true).

4.2.8. UserManager

A UserManager Script felelős a bejelentkezett felhasználó adatainak kezeléséért. Sikeres bejelentkezés esetén a főmenü jobb felső sarkában megjeleníti a felhasználó adatait.

Az alkalmazás indításakor a Start() metódus elindítja a GetPinId() coroutine-t, amely a térképen elérhető összes pin adatát lekéri, és azokat tárolja.

A GetPinId() metódus először meghívja a GetCheckedPins() coroutine-t, amely a felhasználó által „meglátogatott” pinek listáját kéri le a szerverről. Ehhez jogosultsági token szükséges, amit a PlayerPrefs-ből olvas ki és az Authorization fejlécben küld el.

```
private IEnumerator GetCheckedPins()
{
    using (UnityWebRequest request = UnityWebRequest.Get(CHECKEDPINSURL))
    {
        request.SetRequestHeader("Authorization", $"Bearer {PlayerPrefs.GetString("UserToken")}");
        yield return request.SendWebRequest();
```

```
        if (request.result == UnityWebRequest.Result.Success)
        {
            string jsonResponse = request.downloadHandler.text;
            CheckedPinList checkedPinListResponse = JsonUtility.FromJson<CheckedPinList>(jsonResponse);

            checkedPinList.Clear();

            foreach (var checkedPin in checkedPinListResponse.pin_ids)
            {
                checkedPinList.Add(checkedPin);
            }
        }
        else
        {
            Debug.LogError("Error fetching checked pins: " + request.error);
        }
    }
}
```

Ezután a GetPinId() metódus lekéri az összes pin adatait a https://bdk.teamorange.hu/api/pins címről. A kapott adatokat két listába menti:
-idList – csak a pin azonosítók listája
-pinList – teljes pin adatok listája
A felhasználó statisztikáit is frissíti: megjeleníti a fiók létrehozásának dátumát (UserCreatedAt kulcs alapján a PlayerPrefs-ből) és a meglátogatott helyek számát az összes helyhez viszonyítva.

```
private IEnumerator GetPinId()
{
    yield return StartCoroutine(GetCheckedPins());
    using (UnityWebRequest request = UnityWebRequest.Get(ALLPINSURL))
    {
        yield return request.SendWebRequest();
        if (request.result == UnityWebRequest.Result.Success)
        {
```

```
            string jsonResponse = request.downloadHandler.text;
            PinList pinListResponse = JsonUtility.FromJson<PinList>($"{{\"pinIds\":{jsonResponse}}}");
            idList.Clear();
            foreach (Pin pin in pinListResponse.pinIds)
            {
                idList.Add(pin.id);
                pinList.Add(pin);
            }
            int visitedCount = 0;
            foreach (var pin in checkedPinList)
            {
                visitedCount++;
            }
            string created = PlayerPrefs.GetString("UserCreatedAt");
            statsText.text = $"Statistics:\nAccount created at\n{DateTime.Parse(created):yyyy.MM.dd}\nPlaces 'visited'\n{visitedCount}/{pinList.Count}";
        }
        else
        {
            Debug.LogError("Error fetching pin data: " + request.error);
        }
    }
}
```

4.2.9. SideMenu

A SideMenu script felelős az oldalsó menü animált megjelenítéséért és elrejtéséért, valamint a kategóriák (pl. helyek csoportosítása színek és nevek alapján) megjelenítéséért az alkalmazásban.

A script egy egyszerű "slide" animációt valósít meg, amely az oldalsó menüt kinyitja vagy becsukja a felhasználó gombnyomására. A Start() metódusban az alábbi eseménykezelők vannak regisztrálva az OpenSideMenu() a menü kinyitására és a CloseSideMenu() a menü bezárására.

```
private void Start()
{
  openPosition = new Vector2(-sidebarSize, sideMenu.GetComponent<RectTransform>().anchoredPosition.y);
  closedPosition = new Vector2(0, sideMenu.GetComponent<RectTransform>().anchoredPosition.y);
  sideMenu.GetComponent<RectTransform>().anchoredPosition = closedPosition;
  sideMenuOpenBTN.onClick.AddListener(OpenSideMenu);
  sideMenuCloseBTN.onClick.AddListener(CloseSideMenu);
}
```

Az animáció a Update() metódusban történik Vector2.Lerp segítségével, hogy simán mozduljon a menü.

```
private void Update()
{
  if (isSideMenuOpen && sideMenu.GetComponent<RectTransform>().anchoredPosition != openPosition)
  {
    sideMenu.GetComponent<RectTransform>().anchoredPosition = Vector2.Lerp(sideMenu.GetComponent<RectTransform>().anchoredPosition, openPosition, slideSpeed * Time.deltaTime);
  }
  else if (!isSideMenuOpen && sideMenu.GetComponent<RectTransform>().anchoredPosition != closedPosition)
  {
    sideMenu.GetComponent<RectTransform>().anchoredPosition = Vector2.Lerp(sideMenu.GetComponent<RectTransform>().anchoredPosition, closedPosition, slideSpeed * Time.deltaTime);
  }
}
```

Amikor az oldalsó menü kinyílik, a térképen található pinek collider-jeit kikapcsolja, így a háttér nem lesz kattintható, amíg a menü nyitva van. Bezáráskor újra aktiválja őket.

```
public static void SetAllPinsColliders(bool state)
{
  DetailsClcikHandler[] allPins = FindObjectsOfType<DetailsClcikHandler>();
```

```
    foreach (var pin in allPins)
    {
        Collider col = pin.GetComponent<Collider>();
        col.enabled = state;
    }
}
```

A SideMenu a kategóriák színét és nevét is beállítja. A CategorySetter() metódus egy PinManager.Category[] tömböt kap paraméterként. A kategóriák id szerint növekvő sorrendbe rendeződnek. Ha egy kategória színe (category_color) helyesen adható meg (ColorUtility.TryParseHtmlString), akkor az adott kategória Image komponensének színét, és a hozzá tartozó Text komponens szövegét frissíti.

```
public void CategorySetter(PinManager.Category[] catergoryData)
{
    _categoryData = catergoryData.GroupBy(c => c.id)
                                 .Select(g => g.First())
                                 .OrderBy(c => c.id)
                                 .ToArray();

    for (int i = 0; i < Mathf.Min(6, _categoryData.Length); i++)
    {
        if (ColorUtility.TryParseHtmlString(_categoryData[i].category_color.Trim(), out Color color))
        {
            categoryImages[i].color = color;
            categoryNames[i].text = _categoryData[i].category_name;
        }
    }
}
```

4.2.10. LoadingPanelController

A LoadingPanelController Script egy betöltő panelt kezel. A célja, hogy a térkép inicializálásakor megjelenjen egy "LOADING" feliratú panel, amely eltűnik, amikor a térkép teljesen betöltődött. Úgyszintén, ha a felhasználó keresget a térképen, amikor eljut egy pontra, ahol még nincsen betöltve a Tile, akkor is megjelenik a “LOADING” feliratú panel.

```
void _map_OnInitialized()
```

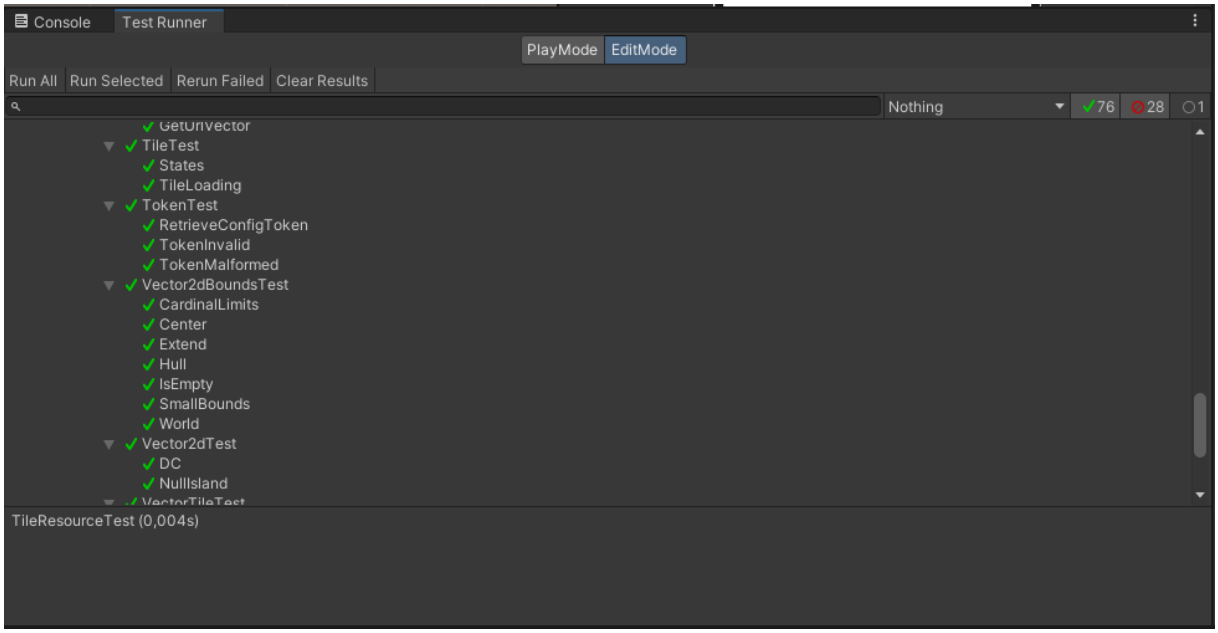
```
{
        var visualizer = _map.MapVisualizer;
        _text.text = "LOADING";
        visualizer.OnMapVisualizerStateChanged += (s) =>
        {
        if (this == null)
        return;

        if (s == ModuleState.Finished)
        {
        _content.SetActive(false);
        }
        else if (s == ModuleState.Working)
        {
        _content.SetActive(true);
        }
        };
}
if (Application.internetReachability == NetworkReachability.NotReachable)
{
        _text.text = "NO INTERNET CONNECTION\nPlease turn on the Wifi/Mobile Data
\nand restart the app!";
        _content.SetActive(true);
        Debug.LogError("No internet connection detected!");
        return;
}
```

4.2.11. A Kamera működése

A projektben a térképmozgatás úgy van megoldva, hogy a kamera helyzete állandó marad, míg a térkép maga mozog a Tile map rendszerén belül. Ez azt jelenti, hogy a felhasználó a térképet mozgathatja, miközben a kamera mindig egy fix pozícióban marad.

4.2.12. Unit Teszt

A Unity-ben a unit tesztek futtatásához a bal felső sarokban található Window menüre kell kattintani, majd a General menüpont alatt kell rákattintani a Test Runner opcióra. Ekkor megnyílik egy külön ablak, ahol az EditMode fül alatt elérhetők és lefuttathatók a unit tesztek a Run All gombbal.



139. ábra Unit tesztek Unityben

A legtöbb teszt, ami failed, azért jelenik meg, mert a Mapbox SDK-ban több teszt template található, és ezek scriptjeit is lefuttatja a rendszer. Nem szeretném ezeket eltávolítani, mivel fennáll a veszélye annak, hogy az egész rendszer összeomolhat.

5. Adatbázis

5.1. Az adatbázis célja

Az adatbázis fő feladata az alkalmazás és a weboldal által kezelt adatok hatékony és strukturált tárolása. Ez magában foglalja a felhasználói fiók adatait, különösen az adminisztrátorok által kezelhető információkat. Az adminisztrátorok jogot kapnak a látványosságok adatainak hozzáadására, módosítására és törlésére. Ezen felül az adatbázisban tárolt információk közé tartozik a látványosságok neve, elhelyezkedése, rövid leírása, kapcsolódó fénykép (ha van), valamint ezek kategorizálása.

Kiemelt figyelmet fordítottunk az adatok védelmére, mivel bizalmas információk, például elérhetőségi és személyes adatok kerülnek tárolásra. Emellett célunk egy hatékony és jól strukturált adatbázis létrehozása volt, amely minimalizálja az ismétlődéseket, lehetővé teszi a gyors adatlekérdezést, és biztosítja az adatok épségét. Különös hangsúlyt fektettünk a jelszavak biztonságos kezelésére.

5.2. Tervezés megkezdése

Az adatbázis tervezésének kezdetekor az alapvető feladat az volt, hogy átgondoljuk, pontosan milyen táblákra is lesz szükség az alkalmazás és a weboldal által kezelt adatok hatékony és strukturált tárolásához. Cél volt egy olyan adatbázis struktúra létrehozása, amely minimalizálja az ismétlődéseket, lehetővé teszi a gyors adatlekérdezést és biztosítja az adatok épségét. A Laravel keretrendszer alapértelmezetten tartalmaz bizonyos táblákat a belső működéshez, mint például a migrációk történetét (migrations), a sikertelenül lefutott feladatokat (failed_jobs), és a jelszó-visszaállítási kéréseket (password_resets), amelyek részét képezték az adatbázis szerkezetének.

5.3. Tervezési lépések

A tervezés kezdetben nehézséget okozott, különösen a táblák közötti kapcsolatok kialakítása. A nehézséget a különböző típusú összefüggések, különösen a sok-a-sokhoz (N:M) kapcsolatok kezelése jelentette, amelyekhez egy köztes, úgynevezett pivot tábla szükséges (például a felhasználók követésénél (follower_user) vagy az ötlet kedveléseknél (idea_like)). A táblák működőképessé alakítása többszöri próbálkozást igényelt. Mivel mindkét szoftverkomponens egyetlen adatbázison alapszik, a tervezéshez egy olyan online felületet kerestünk, ahol mindannyian hozzáférhetünk és közösen meg tudjuk tervezni mind a weboldalhoz, mind a mobilalkalmazáshoz szükséges táblákat az adatbázisban. Erre a célra a

DBDiagram.io online eszközt használtuk, amely egy letisztult és hatékony platform az adatbázis-struktúrák vizuális megjelenítésére. Segítségével egyszerű szintaxis segítségével lehet ER-diagramokat generálni, ami segítette a tervezést és a csapatmunkát. Az elkészült struktúrát végül a Laravel migrációs rendszerével hoztuk létre, a táblák adatainak kezelését pedig a Laravel modellek definiálásával valósítottuk meg. Minden tábla létrehozása a weboldal beépített migrációs funkcióján keresztül ment végbe a fejlesztés során, s ezen felül a táblák közti kapcsolatok is a weboldal modelljeiben meghatározott függvényeken keresztül kerültek meghatározásra. Ez a fajta megközelítés ugyan kezdeti nehézségeket okozott, azonban a keretrendszer széleskörű dokumentációjának és a benne lévő mintakódoknak köszönhetően könnyen megérthetővé és ezáltal gördülékennyé vált az adatbázis tervezésének és megvalósításának menete.

5.4.Közös felhasználói tábla

Felhasználó adatai (users) => felhasználónév, e-mail cím, email-cím hitelesítésének időpontja, jelszó, profilkép (opcionális), felhasználó által megadott leírás önmagáról (opcionális), admin jogosultság.

5.5.Mobilalkalmazás táblái

Látványosságok (pins) => A város látványosságainak adatai, beleértve a címüket, földrajzi koordinátáikat (szélesség, hosszúság), leírásukat, valamint a hozzájuk tartozó esetleges fénykép elérési útvonalát.

Kategóriák (pin_categories) => A látványosságok kategorizálása.

Felhasználók és látványosságok (pin_user) => A felhasználók és a látványosságok közötti kapcsoaltokat tárolja

5.6.Weboldal táblái

Ötletek (ideas) => A felhasználók által létrehozott ötletek tárolására szolgál.

Kommentek (comments) => A felhasználók által írt kommentek tárolására szolgál.

Követések (follower_user) => A felhasználók közötti követési kapcsolatokat tárolja.

Ötlet kedvelések (idea_like) => A felhasználók és ötletek közötti kedvelési kapcsolatokat tárolja.

5.7.A keretrendszer táblái

Migrációk (migrations) => A migrációk történetét tárolja, melyet a Laravel automatikusan kezel.

Jelszó visszaállítások (password_resets) => A Laravel által kezelt tábla, amely a jelszó-visszaállítási kérések ideiglenes tárolására szolgál.

Sikertelen feladatok (failed_jobs) => A végrehajtás során hibásan lefutott feladatok tárolására szolgál. A Laravel automatikusan rögzíti a sikertelen feladatokat ebben a táblában.

Hozzáférési tokenek (personal_access_tokens) => A Laravel által kezelt tábla, amely az API hozzáférési tokeneket tárolja az alkalmazás felhasználói számára.

5.8.Kapcsolatok meghatározása

pins => pin_categories: N:1-es kapcsolat, egy pin csak egy kategóriához tartozhat, egy kategóriához több pin is tartozhat.

users => ideas: 1:N-es kapcsolat, mivel egy felhasználó több ötletet (idea) is létrehozhat, de egy ötlet csak egy felhasználóhoz tartozhat.

users => comments: 1:N-es kapcsolat, mivel egy felhasználó több hozzászólást is írhat, de egy hozzászólás csak egy felhasználóhoz kapcsolódhat.

ideas => comments: 1:N-es kapcsolat, mivel egy ötlethez több hozzászólás is tartozhat, de egy hozzászólás csak egy adott ötlethez kapcsolódhat.

users => users: N:M-es kapcsolat, mivel egy felhasználó több másik felhasználót is követhet, és egy felhasználónak is lehet több követője. Ezért a kapcsolatot egy külön follower_user táblával valósítjuk meg, ahol minden sor egy követési kapcsolatot jelent.

users => ideas: N:M-es kapcsolat, mivel egy felhasználó több ötletet is kedvelhet, és egy posztot több felhasználó is kedvelhet. Ezért a kapcsolatot egy külön idea_like táblával valósítjuk meg, ahol minden sor egy ötlet-kedvelési kapcsolatot jelent.

users => pins: N:M-es kapcsolat, mivel egy felhasználó több pinnel is rendelkezhet, és egy pin több felhasználóhoz tartozhat. Ezért a kapcsolatot egy külön pin_user táblával valósítjuk meg, ahol minden sor egy kapcsolatot jelent egy felhasználó és egy pin között.

5.9.N:M kapcsolatok felbontása

Felhasználók követése (users - users)

- Kapcsolat típusa: N:M kapcsolat
- Egy felhasználó több másik felhasználót is követhet, és egy felhasználónak is lehet több követője.

Felbontás:

users ⇒ follower_user: 1:N-es kapcsolat, mivel egy felhasználó több követési kapcsolatot is létrehozhat.

follower_user ⇒ users: 1:N-es kapcsolat, mivel egy követési kapcsolat mindig egy adott követőre és egy követettre mutat.

A follower_user köztes tábla tárolja a követési kapcsolatokat a user_id és a follower_id oszlopokkal.

Ötletek kedvelése (ideas - users)

- Kapcsolat típusa: N:M kapcsolat
- Egy felhasználó több ötletet is kedvelhet, és egy posztot több felhasználó is kedvelhet.

Felbontás:

users ⇒ idea_like: 1:N-es kapcsolat, mivel egy felhasználó több ötletkedvelési kapcsolatot is létrehozhat.

follower_user ⇒ ideas: 1:N-es kapcsolat, mivel egy kedvelési kapcsolat mindig egy adott ötletre és egy felhasználóra mutat.

Az idea_like köztes tábla tárolja a követési kapcsolatokat a user_id és a idea_id oszlopokkal.

Userek és pinek közti kapcsolat (users - pins)

- Kapcsolat típusa: N:M kapcsolat
- Egy felhasználó több pinnel is rendelkezhet, és egy pin több felhasználóhoz tartozhat.

Felbontás:

users ⇒ pin_user: 1:N-es kapcsolat, mivel egy felhasználó több kapcsolatot is létrehozhat egy pinhez.

pin_user ⇒ users : 1:N-es kapcsolat, mivel egy kapcsolat mindig egy adott pinre és egy felhasználóra mutat.

Az idea_like köztes tábla tárolja a követési kapcsolatokat a user_id és a idea_id oszlopokkal.

5.10. Weboldal táblái

5.10.1. users

A felhasználói adatokat tárolja.

Mező	Típus	Leírás
id PK	bigint(20) UNSIGNED	A felhasználó egyedi azonosítója.
name	varchar(255)	A felhasználó neve.
email	varchar(255)	A felhasználó email címe.
email_verified_at	timestamp	Az email cím hitelesítésének időpontja.
password	varchar(255)	A felhasználó jelszava.
remember_token	varchar(100)	A "megjegyzés" token, amely lehetővé teszi a felhasználó automatikus bejelentkezését.
created_at	timestamp	A felhasználó regisztrációjának időpontja.
updated_at	timestamp	A felhasználó adatainak utolsó frissítése.
image	varchar(255)	A felhasználó profilképe.
bio	varchar(255)	A felhasználó rövid bemutatkozása.
is_admin	tinyint(1)	Admin jogosultság (0 vagy 1).

5.10.2. ideas

A felhasználók által létrehozott ötletek tárolására szolgál.

Mező	Típus	Leírás
id PK	bigint(20) UNSIGNED	Egyedi azonosító, automatikusan növekvő szám.
user_id FK	bigint(20) UNSIGNED	Az ötletet létrehozó felhasználó azonosítója. Ez az érték egy idegen kulcs, amely a **users** táblához kapcsolódik.
content	varchar(255)	Az ötlet szövege.
created_at	timestamp	Az ötlet létrehozásának időpontja.
updated_at	timestamp	Az ötlet utolsó frissítésének időpontja.

5.10.3. comments

A felhasználók által írt kommentek tárolására szolgál.

Mező	Típus	Leírás
id PK	bigint(20) UNSIGNED	Egyedi azonosító, automatikusan növekvő szám.
user_id FK	bigint(20) UNSIGNED	A felhasználó azonosítója, aki a kommentet írta. Ez az érték egy idegen kulcs, amely a **users** táblához

		kapcsolódik.
idea_id FK	bigint(20) UNSIGNED	Az ötlet azonosítója, amelyhez a komment tartozik. Ez az érték egy idegen kulcs, amely az **ideas** táblához kapcsolódik.
content	varchar(255)	A komment szövege.
created_at	timestamp	A rekord létrehozásának időpontja.
updated_at	timestamp	Az utolsó frissítés időpontja.

5.10.4. follower_user

A felhasználók közötti követési kapcsolatokat tárolja. Egy N:M-es kapcsolatot bont fel két user között, amelyet Laravel-ben egy pivot tábla valósít meg.

Mező	Típus	Leírás
user_id FK	bigint(20) UNSIGNED	A felhasználó azonosítója, akit követ egy másik felhasználó. Ez az érték egy idegen kulcs, amely a **users** táblához kapcsolódik.
follower_id FK	bigint(20) UNSIGNED	A követő felhasználó azonosítója. Ez az érték egy idegen kulcs, amely a **users** táblához kapcsolódik.
created_at	timestamp	A rekord létrehozásának időpontja.
updated_at	timestamp	Az utolsó frissítés időpontja.

5.10.5. idea_like

Eltárolja, hogy egy posztot melyik felhasználók kedvelték, s hogy egy felhasználó melyik posztokat kedveli. Egy N:M-es kapcsolatot bont fel egy idea és egy user között, amelyet Laravel-ben egy pivot tábla valósít meg.

Mező	Típus	Leírás
id PK	bigint(20) UNSIGNED	Egyedi azonosító, automatikusan növekvő szám.
user_id FK	bigint(20) UNSIGNED	A felhasználó azonosítója, aki kedveli az ötletet. Ez az érték egy idegen kulcs, amely a **users** táblához kapcsolódik.
idea_id FK	bigint(20) UNSIGNED	Az ötlet azonosítója, amit kedveltek. Ez az érték egy idegen kulcs, amely a **ideas** táblához kapcsolódik.
created_at	timestamp	A kedvelés létrehozásának időpontja.
updated_at	timestamp	A kedvelés utolsó frissítésének időpontja.

5.11. Mobilalkalmazás táblái

5.11.1. pins

A történelmi emlékek és annak tűzői (pin) adatai tárolására használt tábla.

Mező	Típus	Leírás

id PK	bigint(20) UNSIGNED	Egyedi azonosító minden pinhez.
pin_name	varchar(255)	Az adott tűzőn (pin) lévő történelmi emlék neve.
pin_description	longtext	Az adott tűzőn (pin) lévő történelmi emlék részletes leírása.
image_link	longtext	Az adott tűzőn (pin) lévő történelmi emlékhez tartozó kép linkje.
latitude	decimal(8,6)	A tűző (pin) GPS koordinátáinak szélességi foka.
longitude	decimal(8,6)	A tűző (pin) GPS koordinátáinak hosszúsági foka.
pin_category_id FK	bigint(20) UNSIGNED	Az adott tűzőn (pin) lévő történelmi emlék hozzárendelt kategóriájának azonosítója. Ez az érték egy idegen kulcs, amely a **pins** táblához kapcsolódik.
created_at	timestamp	A pin létrehozásának időpontja.
updated_at	timestamp	A pin utolsó frissítésének időpontja.

5.11.2. pin_categories

Az adott pin-en lévő történelmi emlék kategóriaját tároló tábla.

Mező	Típus	Leírás
id PK	bigint(20) UNSIGNED	Egyedi azonosító minden kategóriához.
category_name	varchar(255)	A kategória neve.
category_color	varchar(255)	Kategória színe.
created_at	timestamp	A pin kategóriájának létrehozásának időpontja.
updated_at	timestamp	A pin kategóriájának utolsó frissítésének időpontja.

5.11.3. users

Az applikáció ugyanazt a users táblát használja fel a bejelentkezéshez mint a weboldal.

5.11.4. pin_user

Ez a tábla összeköti a különböző tűzőket és a felhasználókat.

Mező	Típus	Leírás
id	bigint(20)	Elsődleges azonosító.
pin_id	bigint(20)	A hozzárendelt pin azonosítója.
user_id	bigint(20)	A hozzárendelt felhasználó azonosítója.
created_at	timestamp	Egy pin_user kapcsolat létrehozásának időpontja.
updated_at	timestamp	Egy pin_user kapcsolat utolsó frissítésének időpontja.

5.12. A Laravel keretrendszer további táblái

Ezeket a táblákat a Laravel automatikusan generálja és kezeli.

5.12.1. failed_jobs tábla

A végrehajtás során hibásan lefutott feladatok tárolására szolgál.

Mező	Típus	Leírás
id PK	bigint(20) UNSIGNED	Egyedi azonosító, automatikusan növekvő szám.
uuid	varchar(255)	Az egyedi azonosító UUID formátumban.
connection	text	A kapcsolat típusának leírása.

queue	text	A sor, ahol a feladat található.
payload	longtext	A hibásan végrehajtott feladat adatainak tárolása.
exception	longtext	A feladat végrehajtásának hibájának részletezése.
failed_at	timestamp	A feladat végrehajtásának hibás befejezésének időpontja.

5.12.2. migrations tábla

A migrációk történetét tárolja.

Mező	Típus	Leírás
id PK	int(10) UNSIGNED	A migráció egyedi azonosítója.
migration	varchar(255)	A migráció neve.
batch	int(11)	A migrációk batch száma, amely segít az alkalmazott migrációk nyomon követésében.

5.12.3. password_resets

A password_resets tábla a felhasználók jelszó-visszaállítási kérelmeit tárolja, ideiglenes tokenekkel és a kérelmek időbélyegével.

Mező	Típus	Leírás
email	varchar(255)	A felhasználó email címe,

		aki jelszó-visszaállítást kért.
token	varchar(255)	A jelszó-visszaállítási token.
created_at	timestamp	A jelszó-visszaállítási kérelem időpontja.

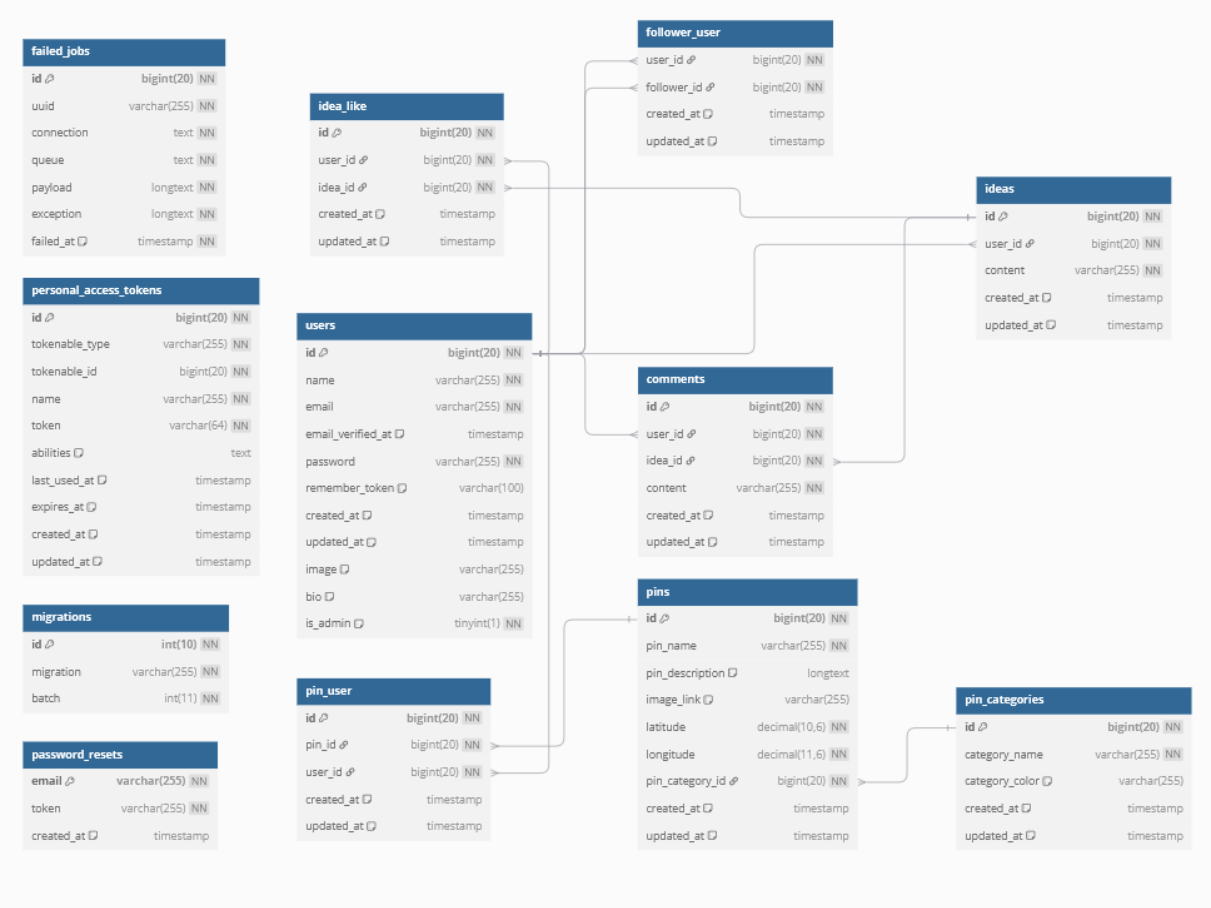
5.12.4. personal_access_tokens

A personal_access_tokens tábla a Laravel Sanctum által kezelt tokeneket tárolja, amelyeket felhasználók vagy más modellek kapnak autentikációhoz. A tábla egy polimorf kapcsolaton keresztül (tokenable_type és tokenable_id) kapcsolódik a jogosult modellhez (pl. User). Ez azt jelenti, hogy nem csak egy konkrét modellhez, hanem bármilyen típusú modellhez tud token tartozni, amely használja a HasApiTokens traitet.

Mező	Típus	Leírás
id PK	bigint(20) UNSIGNED	Egyedi azonosító, automatikusan növekvő szám.
tokenable_type	varchar(255)	Az entitás típusa, amelyhez a token tartozik.
tokenable_id	bigint(20) UNSIGNED	Az entitás azonosítója, amelyhez a token tartozik.
name	varchar(255)	A token neve.
token	varchar(64)	A titkosított token értéke.
abilities	text	A token jogosultságainak listája.
last_used_at	timestamp	A token utolsó használatának időpontja.
expires_at	timestamp	A token lejárati időpontja

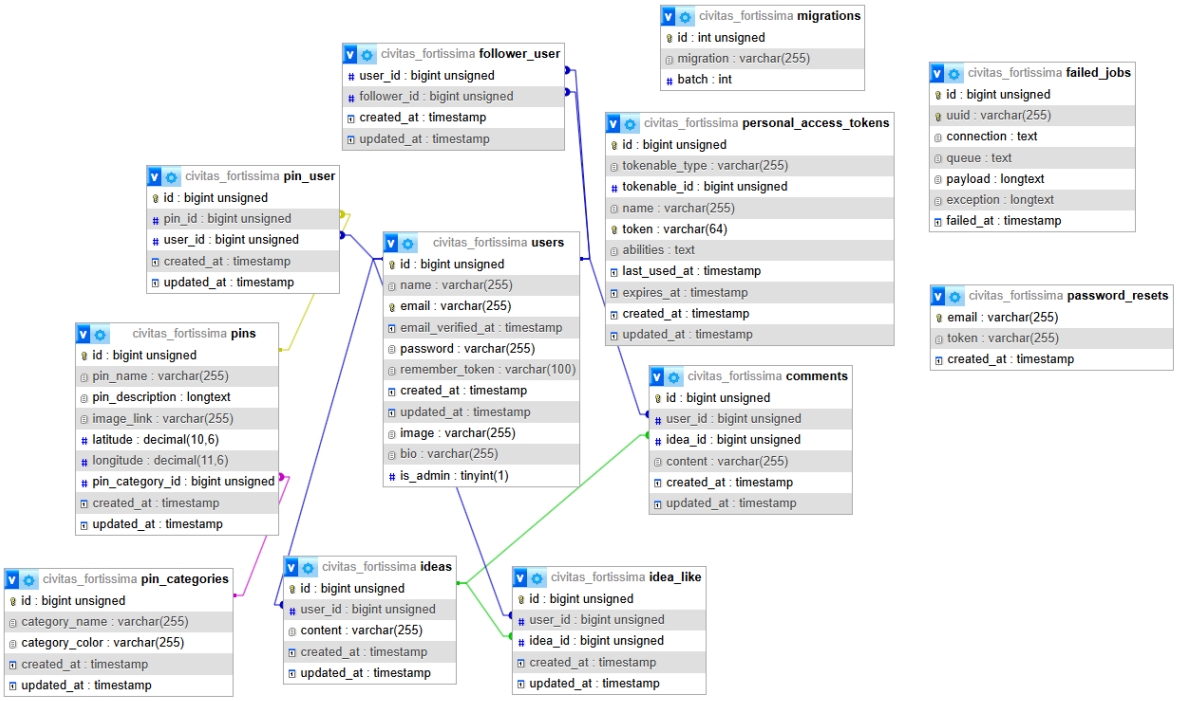
		(opcionális).
created_at	timestamp	A token létrehozásának időpontja.
updated_at	timestamp	A token utolsó frissítésének időpontja.

5.13. Adatbázismodell-diagram (dbdiagram.io)



140. ábra Adatbázismodell-diagram (dbdiagram.io)

5.14. Adatbázismodell-diagram (phpmyadmin)



141. ábra Adatbázismodell-diagram (phpmyadmin)

5.15. Adatbázis-továbbfejlesztési lehetőségek

Adminisztrátorok bejelentkezése felhasználóként: A rendszer jelenleg rendelkezik adminisztrátori jogosultság jelzésére szolgáló logikai mezővel a users táblában (is_admin), és az admin jogosultságokat a Laravel Gate-ek segítségével kezeli. Egy továbbfejlesztési lehetőség az lenne, hogy egy adminisztrátori jogosultságú felhasználó normál felhasználóként is be tudjon jelentkezni a weboldalra, azaz az adminok ugyanazon a bejelentkezési folyamaton mehessenek keresztül, s ugyanazon funkciókkal rendelkezhessenek, mint a többi felhasználó. Ez igényel egy kisebb adatbázis-struktúra módosítást is, például egy új mező (pl. logged_in_as_admin) bevezetését a users táblában, valamint az autentikációs és jogosultságkezelési logika kiegészítését.

Beágyazott kommentek (kommentek kommentelése): Jelenleg a comments tábla az ötletekhez kapcsolódó hozzászólásokat tárolja, egy 1:N kapcsolattal az ideas táblához. A kommentekre adott válaszok lehetőségének bevezetése beágyazott kommentek formájában adatbázis-struktúra módosítást igényelne a comments táblán. Ehhez egy önmagára hivatkozó idegen kulcs (pl. parent_comment_id) hozzáadása lenne szükséges a comments táblához, ami jelezné, hogy egy adott komment melyik másik kommentre válaszol.

Kommentek kedvelése: Az ötletek kedvelése már megvalósított funkció, mely egy N:M kapcsolatot kezel a users és ideas táblák között egy idea_like nevű pivot tábla segítségével. A

kommentek kedvelésének bevezetése hasonló adatbázis-struktúrát igényelne. Egy új, dedikált pivot tábla (pl. comment_like) létrehozására lenne szükség az adatbázisban, amely a users és a comments táblák közötti N:M kapcsolatot valósítaná meg, hasonló módon, mint az idea_like táblához. Ez a tábla tárolná, hogy melyik felhasználó melyik kommentet kedvelte.

Több kép tárolása látványosságokhoz: A pins tábla jelenleg egyetlen kép (image_link) tárolására alkalmas. A látványosságok bemutatásának gazdagítása érdekében lehetőség lenne több kép tárolására egy új tábla (pl. pin_images) létrehozásával, amely 1:N kapcsolatban állna a pins táblával. Ez a tábla tárolná a képek elérési útvonalait és esetleg további metaadatokat róluk.

Strukturáltabb helyadatok tárolása: A pins tábla a földrajzi koordinátákat (latitude, longitude) tárolja. A helyadatok részletesebb kezelése és megjelenítése érdekében további, strukturált címadatok (pl. utca, házszám, városrész, irányítószám) tárolása is megfontolható egy új, kapcsolódó táblában, vagy a pins tábla bővítésével.

5.16. Az adatbázis export fájlja

```
-- phpMyAdmin SQL Dump
-- version 5.2.2
-- https://www.phpmyadmin.net/
--
-- Gép: localhost
-- Létrehozás ideje: 2025. Ápr 26. 18:43
-- Kiszolgáló verziója: 10.11.11-MariaDB-0+deb12u1
-- PHP verzió: 8.2.28

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
*/;
```

```
/*!40101 SET NAMES utf8mb4 */;

--
-- Adatbázis: `civitas_fortissima`
--

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `comments`
--

CREATE TABLE `comments` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `user_id` bigint(20) UNSIGNED NOT NULL,
  `idea_id` bigint(20) UNSIGNED NOT NULL,
  `content` varchar(255) NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `failed_jobs`
--

CREATE TABLE `failed_jobs` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `uuid` varchar(255) NOT NULL,
  `connection` text NOT NULL,
  `queue` text NOT NULL,
  `payload` longtext NOT NULL,
  `exception` longtext NOT NULL,
```

```
  `failed_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `follower_user`
--

CREATE TABLE `follower_user` (
  `user_id` bigint(20) UNSIGNED NOT NULL,
  `follower_id` bigint(20) UNSIGNED NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `ideas`
--

CREATE TABLE `ideas` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `user_id` bigint(20) UNSIGNED NOT NULL,
  `content` varchar(255) NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `idea_like`
```

```
--

CREATE TABLE `idea_like` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `user_id` bigint(20) UNSIGNED NOT NULL,
  `idea_id` bigint(20) UNSIGNED NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `migrations`
--

CREATE TABLE `migrations` (
  `id` int(10) UNSIGNED NOT NULL,
  `migration` varchar(255) NOT NULL,
  `batch` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

--
-- A tábla adatainak kiíratása `migrations`
--

INSERT INTO `migrations` (`id`, `migration`, `batch`) VALUES
(1, '2014_10_12_000000_create_users_table', 1),
(2, '2014_10_12_100000_create_password_resets_table', 1),
(3, '2019_08_19_000000_create_failed_jobs_table', 1),
(4, '2019_12_14_000001_create_personal_access_tokens_table', 1),
(5, '2024_12_27_220714_create_ideas_table', 1),
(6, '2024_12_31_184808_create_comments_table', 1),
(7, '2025_01_07_130101_add_bio_and_image_to_users', 1),
```

```
(8, '2025_01_12_155435_create_follower_user_table', 1),
(9, '2025_01_13_213027_drop_likes_from_ideas', 1),
(10, '2025_01_13_213513_create_idea_like_table', 1),
(11, '2025_01_14_130228_add_is_admin_to_users', 1),
(12, '2025_03_11_091231_create_pin_categories_table', 1),
(13, '2025_03_11_092819_create_pins_table', 1),
(14, '2025_03_11_093616_create_pin_user_table', 1),
(15, '2025_04_07_135738_add_category_color_to_pin_categories', 1);

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `password_resets`
--

CREATE TABLE `password_resets` (
  `email` varchar(255) NOT NULL,
  `token` varchar(255) NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `personal_access_tokens`
--

CREATE TABLE `personal_access_tokens` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `tokenable_type` varchar(255) NOT NULL,
  `tokenable_id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) NOT NULL,
  `token` varchar(64) NOT NULL,
  `abilities` text DEFAULT NULL,
```

```
  `last_used_at` timestamp NULL DEFAULT NULL,
  `expires_at` timestamp NULL DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `pins`
--

CREATE TABLE `pins` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `pin_name` varchar(255) NOT NULL,
  `pin_description` longtext DEFAULT NULL,
  `image_link` longtext DEFAULT NULL,
  `latitude` decimal(10,6) NOT NULL,
  `longitude` decimal(11,6) NOT NULL,
  `pin_category_id` bigint(20) UNSIGNED NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

--
-- A tábla adatainak kiíratása `pins`
--

INSERT INTO `pins` (`id`, `pin_name`, `pin_description`, `image_link`, `latitude`, `longitude`, `pin_category_id`, `created_at`, `updated_at`) VALUES
(1, 'Monument to the heroines of 1648 and 1848', 'This is the city\'s first non-religious public art work, commemorating the women who played a role in the successful defense of Gyarmat Castle in 1648 and the 1848 revolution. Its cost was raised by donations, and an obelisk was erected. The ceremonial inauguration took place in 1908.',
```

'https://upload.wikimedia.org/wikipedia/commons/3/34/Memorial_of_female_heroes_Balassagyarmat.jpg', 48.075361, 19.289000, 1, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(2, 'World War I memorial', 'This memorial was erected in memory of the citizens who died in the First World War. Its cost was raised through donations. The inauguration took place on October 20, 1929, in the square next to the county hall, in the presence of Governor Miklós Horthy. Since the area was temporary, the statue was moved to Palóc liget in 1937. The memorial was later renovated, including in February 2016, when missing parts of the lion were replaced based on archival photos and the model.', 'https://upload.wikimedia.org/wikipedia/commons/thumb/1/13/World_War_I_memorial_Balassagyarmat_2.jpg/1280px-World_War_I_memorial_Balassagyarmat_2.jpg', 48.074611, 19.290111, 1, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(3, 'Statue of the 16th home guard', 'The erection of the World War I home guard memorial was initiated in 1935 by former members of the 16th Home Guard Infantry Regiment and was realized through donations. It was made by Jenő Körmendi-Frim. The memorial was inaugurated on October 27, 1937, in Hősök Square. The statue was moved to a secluded corner of Palóc liget in 1967, then returned to its original location in 1990. It was renovated in 2015.', 'https://upload.wikimedia.org/wikipedia/commons/thumb/0/0a/H%C5%91s%C3%B6k_tere_4.JPG/800px-H%C5%91s%C3%B6k_tere_4.JPG', 48.072389, 19.290472, 1, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(4, 'Madách statue', 'The statue of Imre Madách, an outstanding personality of Hungarian literature, was made by Ferenc Sidló. The county issued a closed competition in 1935. The location for the statue was the square between the Courthouse and the County Hall. The ceremonial handover took place on September 26, 1937.', 'https://upload.wikimedia.org/wikipedia/commons/5/5c/Bgyarmat_Madach2.jpg', 48.076611, 19.289417, 2, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(5, 'Mikszáth bust', 'Bust of Kálmán Mikszáth, made by Munkácsy Prize-winning sculptor Klára Herczeg. The city council decided on its erection in the late 1950s. The completed bust was inaugurated in 1961 in front of the county hall, in Köztársaság Square. The bust was renovated in 2012-2013.', 'https://upload.wikimedia.org/wikipedia/commons/d/de/Miksz%C3%A1th_K%C3%A1lm%C3%A1n_fel%C3%BAj%C3%ADtott_mellszobra_Balassagyarmaton.jpg', 48.077028, 19.290722, 2, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(6, 'Petőfi statue', 'Full-figure bronze statue of Sándor Petőfi, made by Tibor Borbás. It was handed over on March 15, 1977, in Palóc liget. The city\'s commemoration of the March 15th

national holiday is held at the statue. Due to the shape of the cape stretched out by the extended arms, locals often mockingly call it the \"bat\".', 'https://upload.wikimedia.org/wikipedia/commons/0/07/Pet%C5%91fi_statue_Balassagyarmat_2.jpg', 48.074917, 19.290889, 2, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(7, 'Jenő Komjáthy bust', 'Sándor Győrfi\'s bust of Jenő Komjáthy was handed over in 1981 in Palóc liget.', 'https://upload.wikimedia.org/wikipedia/commons/d/d7/Komj%C3%A1thy_statue_Balassagyarmat.jpg', 48.074583, 19.288861, 2, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(8, 'Ernő Kondor bust', 'Barna Búza\'s pyrogranite bust of Ernő Kondor has stood in the courtyard of the Local History Collection since 1980.', NULL, 48.079972, 19.304139, 2, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(9, 'Lőrinc Szabó bust (2017)', 'The new Lőrinc Szabó bust, created by Balassagyarmat sculptor György Petró, somewhat evokes the work by Miklós Borsos erected in 1977 and disappeared in 1997. The inauguration ceremony took place on October 3, 2017.', 'https://upload.wikimedia.org/wikipedia/commons/5/5e/Szab%C3%B3_L%C5%91rinc%2C_Pal%C3%B3c_liget_2.jpg', 48.074972, 19.289000, 2, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(10, 'Memorial plaque for driving out the Czechs', 'Balassagyarmat\'s first public artwork created with public funds, commemorating the uprising of January 29, 1919. The memorial plaque was made by Hugó Keviczky. It was inaugurated on November 22, 1922, on the wall of the town hall , Horthy Miklós also gave a speech. Later, the bronze turul bird was removed and the plaque was turned over, engraving a new text on it. After the change of regime, it was turned back, and the turul was replaced with a copy.', 'https://upload.wikimedia.org/wikipedia/commons/thumb/c/ca/Repulsion_of_Czech_troops_plaque_%28Balassagyarmat_R%C3%A1k%C3%B3czi_fejedelem_%C3%BAt_12%29.jpg/1024px-Repulsion_of_Czech_troops_plaque_%28Balassagyarmat_R%C3%A1k%C3%B3czi_fejedelem_%C3%BAt_12%29.jpg', 48.077389, 19.291083, 3, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(11, 'Imre Madách memorial plaque', 'Memorial plaque made by Hugó Keviczky. It was inaugurated on September 6, 1925, on the wall of the county hall during the opening of the economic exhibition. Festive speeches at the inauguration were given by House Speaker Béla Scitovszky, Governor Miklós Horthy, and MTA President Albert Berzeviczy. The inscription on the plaque was later partially erased , then made readable again or re-engraved based on civil

initiative.',
'https://upload.wikimedia.org/wikipedia/commons/thumb/2/25/Mad%C3%A1ch_Imre_plaque_%28Balassagyarmat_Civitas_Fortissima_t%C3%A9r_2%29.jpg/1024px-Mad%C3%A1ch_Imre_plaque_%28Balassagyarmat_Civitas_Fortissima_t%C3%A9r_2%29.jpg', 48.076917, 19.290528, 3, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(12, 'Kálmán Mikszáth memorial plaque', 'Memorial plaque made by Hugó Keviczky. It was inaugurated on September 6, 1925, on the wall of the county hall during the opening of the economic exhibition. Festive speeches at the inauguration were given by House Speaker Béla Scitovszky, Governor Miklós Horthy, and MTA President Albert Berzeviczy. The inscription on the plaque was later partially erased, then after 1990, the \"fragmented\" words were made readable again.',
'https://upload.wikimedia.org/wikipedia/commons/3/3f/Miksz%C3%A1th_K%C3%A1lm%C3%A1n_plaque_%28Balassagyarmat_Civitas_Fortissima_t%C3%A9r_2%29.jpg', 48.076833, 19.290361, 3, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(13, 'Memorial plaque for the heroes of Balassi Bálint Grammar School', 'Memorial plaque for the teachers and students who died in the First World War. The memorial plaque was made by János Branceisz. It was unveiled on June 10, 1926, at the celebration of the 25th anniversary of the establishment of the Hungarian Royal State Grammar School.',
'https://upload.wikimedia.org/wikipedia/commons/0/06/WWI_plaque_%28Balassagyarmat_De%C3%A1k_Ferenc_u_17%29.jpg', 48.075722, 19.294167, 3, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(14, 'Statues of the courthouse', 'Statues depicting a praetor symbolizing the guardian of the law and Iustitia, the Roman goddess of justice, on the main facade of the Balassagyarmat Courthouse. The building was handed over in 1912. Due to the deterioration of the statues, the originals were replaced with new statue copies in 2022-2023.',
'https://upload.wikimedia.org/wikipedia/commons/b/b6/Praetur_und_Justitia_Statue%2C_Landgericht%2C_2022_Balassagyarmat.jpg', 48.076389, 19.288889, 4, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(15, 'Boy Riding statue', 'Bronze statue created by Károly Vasas. It was placed in front of the Ifjúság Úti Elementary School in 1968 , then relocated to the front of the Szabó Lőrinc Elementary School after the closure of the former.', NULL, 48.067806, 19.304778, 4, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(16, 'Balassi Bálint statue', 'Waist-up sculpted statue created by Péter Párkányi Raab, depicting Balassi while writing. It was inaugurated in 2004 in the garden of the Balassi Bálint Grammar

School , at the former location of the Balassi stele. (Note: Not a public art work.)', NULL, 48.075833, 19.294167, 5, '2025-04-26 18:43:27', '2025-04-26 18:43:27'),
(17, 'Decorative plates', 'A series of six pieces made of pyrogranite by ceramist Veronika Szabady, depicting three grotesque faces. They were exhibited in 1974 , then displayed on the facade of the main square building. Following the building\'s renovation, they were not returned to their original location but were moved to the Jánossy Gallery after restoration. (Note: Demolished/Relocated work.)', 'https://upload.wikimedia.org/wikipedia/commons/thumb/c/cc/D%C3%ADszt%C3%A1lak%2C_Balassagyarmat.jpg/1280px-D%C3%ADszt%C3%A1lak%2C_Balassagyarmat.jpg', 48.076750, 19.291139, 6, '2025-04-26 18:43:27', '2025-04-26 18:43:27');

-- --

--
-- Tábla szerkezet ehhez a táblához `pin_categories`
--

```
CREATE TABLE `pin_categories` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `category_name` varchar(255) NOT NULL,
  `category_color` varchar(255) DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

--
-- A tábla adatainak kiíratása `pin_categories`
--

INSERT INTO `pin_categories` (`id`, `category_name`, `category_color`, `created_at`, `updated_at`) VALUES
(1, 'Historical events', '#8B0000', '2025-04-26 18:43:17', '2025-04-26 18:43:17'),
(2, 'Famous people', '#1E90FF', '2025-04-26 18:43:17', '2025-04-26 18:43:17'),
(3, 'Memorial plaques', '#FFD700', '2025-04-26 18:43:17', '2025-04-26 18:43:17'),

```
(4, 'Other works', '#32CD32', '2025-04-26 18:43:17', '2025-04-26 18:43:17'),
(5, 'Non-public works', '#8A2BE2', '2025-04-26 18:43:17', '2025-04-26 18:43:17'),
(6, 'Demolished works', '#A9A9A9', '2025-04-26 18:43:17', '2025-04-26 18:43:17');

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `pin_user`
--

CREATE TABLE `pin_user` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `pin_id` bigint(20) UNSIGNED NOT NULL,
  `user_id` bigint(20) UNSIGNED NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- --------------------------------------------------------

--
-- Tábla szerkezet ehhez a táblához `users`
--

CREATE TABLE `users` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `email_verified_at` timestamp NULL DEFAULT NULL,
  `password` varchar(255) NOT NULL,
  `remember_token` varchar(100) DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  `image` varchar(255) DEFAULT NULL,
```

```
  `bio` varchar(255) DEFAULT NULL,
  `is_admin` tinyint(1) NOT NULL DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

--
-- Indexek a kiírt táblákhoz
--

--
-- A tábla indexei `comments`
--
ALTER TABLE `comments`
  ADD PRIMARY KEY (`id`),
  ADD KEY `comments_user_id_foreign` (`user_id`),
  ADD KEY `comments_idea_id_foreign` (`idea_id`);

--
-- A tábla indexei `failed_jobs`
--
ALTER TABLE `failed_jobs`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `failed_jobs_uuid_unique` (`uuid`);

--
-- A tábla indexei `follower_user`
--
ALTER TABLE `follower_user`
  ADD KEY `follower_user_user_id_foreign` (`user_id`),
  ADD KEY `follower_user_follower_id_foreign` (`follower_id`);

--
-- A tábla indexei `ideas`
--
ALTER TABLE `ideas`
```

```
  ADD PRIMARY KEY (`id`),
  ADD KEY `ideas_user_id_foreign` (`user_id`);

--
-- A tábla indexei `idea_like`
--
ALTER TABLE `idea_like`
  ADD PRIMARY KEY (`id`),
  ADD KEY `idea_like_user_id_foreign` (`user_id`),
  ADD KEY `idea_like_idea_id_foreign` (`idea_id`);

--
-- A tábla indexei `migrations`
--
ALTER TABLE `migrations`
  ADD PRIMARY KEY (`id`);

--
-- A tábla indexei `password_resets`
--
ALTER TABLE `password_resets`
  ADD PRIMARY KEY (`email`);

--
-- A tábla indexei `personal_access_tokens`
--
ALTER TABLE `personal_access_tokens`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `personal_access_tokens_token_unique` (`token`),
  ADD KEY `personal_access_tokens_tokenable_type_tokenable_id_index`
(`tokenable_type`,`tokenable_id`);

--
-- A tábla indexei `pins`
```

```
--
ALTER TABLE `pins`
  ADD PRIMARY KEY (`id`),
  ADD KEY `pins_pin_category_id_foreign` (`pin_category_id`);

--
-- A tábla indexei `pin_categories`
--
ALTER TABLE `pin_categories`
  ADD PRIMARY KEY (`id`);

--
-- A tábla indexei `pin_user`
--
ALTER TABLE `pin_user`
  ADD PRIMARY KEY (`id`),
  ADD KEY `pin_user_pin_id_foreign` (`pin_id`),
  ADD KEY `pin_user_user_id_foreign` (`user_id`);

--
-- A tábla indexei `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `users_email_unique` (`email`);

--
-- A kiírt táblák AUTO_INCREMENT értéke
--

--
-- AUTO_INCREMENT a táblához `comments`
--
ALTER TABLE `comments`
```

```
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT a táblához `failed_jobs`
--
ALTER TABLE `failed_jobs`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT a táblához `ideas`
--
ALTER TABLE `ideas`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT a táblához `idea_like`
--
ALTER TABLE `idea_like`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT a táblához `migrations`
--
ALTER TABLE `migrations`
  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=16;

--
-- AUTO_INCREMENT a táblához `personal_access_tokens`
--
ALTER TABLE `personal_access_tokens`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

--
```

```
-- AUTO_INCREMENT a táblához `pins`
--
ALTER TABLE `pins`
 MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=18;

--
-- AUTO_INCREMENT a táblához `pin_categories`
--
ALTER TABLE `pin_categories`
 MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=7;

--
-- AUTO_INCREMENT a táblához `pin_user`
--
ALTER TABLE `pin_user`
 MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT a táblához `users`
--
ALTER TABLE `users`
 MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

--
-- Megkötések a kiírt táblákhoz
--

--
-- Megkötések a táblához `comments`
--
ALTER TABLE `comments`
```

```
 ADD CONSTRAINT `comments_idea_id_foreign` FOREIGN KEY (`idea_id`) REFERENCES `ideas` (`id`) ON DELETE CASCADE,
 ADD CONSTRAINT `comments_user_id_foreign` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Megkötések a táblához `follower_user`
--
ALTER TABLE `follower_user`
 ADD CONSTRAINT `follower_user_follower_id_foreign` FOREIGN KEY (`follower_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,
 ADD CONSTRAINT `follower_user_user_id_foreign` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Megkötések a táblához `ideas`
--
ALTER TABLE `ideas`
 ADD CONSTRAINT `ideas_user_id_foreign` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Megkötések a táblához `idea_like`
--
ALTER TABLE `idea_like`
 ADD CONSTRAINT `idea_like_idea_id_foreign` FOREIGN KEY (`idea_id`) REFERENCES `ideas` (`id`) ON DELETE CASCADE,
 ADD CONSTRAINT `idea_like_user_id_foreign` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Megkötések a táblához `pins`
--
ALTER TABLE `pins`
```

```
  ADD CONSTRAINT `pins_pin_category_id_foreign` FOREIGN KEY (`pin_category_id`) REFERENCES `pin_categories` (`id`) ON DELETE CASCADE;

--
-- Megkötések a táblához `pin_user`
--
ALTER TABLE `pin_user`
  ADD CONSTRAINT `pin_user_pin_id_foreign` FOREIGN KEY (`pin_id`) REFERENCES `pins` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `pin_user_user_id_foreign` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

6. Felmerült akadályok

6.1. Frontend

Az oldal reszponzivitása az első navigációs menü miatt gondokat okozott. Ezalatt értve, hogy a display: none nem tűntetett el bizonyos diveket a mobilos nézetben. Ezt megpróbáltam megoldani egy javascript fájllal, mely megnézte a weboldal szélességét és dinamikusan html fájlt váltott. Ezzel a felmerült probléma az volt, hogy sok idő lett volna módosítani a fájlokat, és emellé a kód mérete a tervezettnél sokkal nagyobbra sikeredett volna. Ezzel a megközelítéssel a végző problémát az jelentette, hogy a weboldal nagyon lassan töltött be, ezért media query-ket vezettem be, mely nagy mértékben megnövelte az oldal teljesítményét. A navigációs sávot áthelyeztem az oldal tetejére, mellyel könnyedén meg tudtam valósítani az oldal rezponzivitását media query-k segítségével, s ezzel a felhasználói élményen javítani tudtam.

Az ikonok kezdetben url-en keresztül töltöttek be. Ezzel az volt a probléma, hogy kijelöléskor a weboldal az ikont darabokra bontotta. Ha például 4 betűs volt egy ikon neve, akkor 4 részre bontotta fel. Ez úgy lett megoldva, hogy az ikonokat letöltöttem, s egy közös mappába helyeztem el őket.

6.2. Backend

A backend fejlesztése során felmerült egy jelentős akadály, amely a projekt Githubról történő friss letöltésekor okozott problémát.

A helyzet abból adódott, hogy a legtöbb oldalon megjelenő "top users" funkcióhoz (amely a legtöbb ötletet megosztó felhasználókat listázza) adatokat kellett elérhetővé tenni az összes nézet számára (global variable). Erre a View::share() metódust használtam az AppServiceProvider.php boot() metódusában.

A probléma az volt, hogy a AppServiceProvider boot() metódusába helyezett kód megpróbált egy lekérdezést futtatni az adatbázison, mégpedig a felhasználók ötleteinek számát, a withCount('ideas') segítségével. A projekt friss letöltésekor az adatbázis táblái – beleértve a felhasználókat (users) és az ötleteket (ideas) tároló táblákat, valamint a migrációk követéséhez szükséges migrations táblát – még nem léteztek, mivel a migrációk nem futottak le.

Ez a korai adatbázis-hozzáférés egy hibát okozott az alkalmazás bootoláskor. A Laravel keretrendszer úgy működik, hogy ha hiba történik az indítási folyamat során, akkor nem engedélyezi a php artisan parancsok futtatását. Ez patthelyzetbe vezetett: nem lehetett futtatni a php artisan migrate parancsot, mert az alkalmazás indításakor hiba lépett fel. A hiba pedig

azért lépett fel, mert a boot() metódusban a migrációk által létrehozandó táblákból próbált adatot lekérdezni. A migrációk szükségesek voltak a hiba elhárításához, de a hiba megakadályozta a migrációk futtatását.

A megoldást a nézetkomponensek (View Composers) alkalmazása jelentette. Ezzel a módszerrel az adatokat nem az alkalmazás indításakor kötik a nézetekhez, hanem közvetlenül azelőtt, hogy a nézet renderelésre kerülne.

Ezáltal a TopUserComposer osztály compose() metódusában történő adatbázis-lekérdezés csak akkor fut le, amikor a nézet ténylegesen megjelenik. Ekkorra már le lehetett futtatni a migrációkat (mivel a boot folyamat már nem omlott össze az előbb említett adatbázis-lekérés miatt), így a szükséges táblák rendelkezésre álltak, és a lekérdezés sikeresen lefutott. A nézetkomponens regisztrációja továbbra is a AppServiceProvider boot() metódusában történt, de maga az adatlekérés későbbre tolódott, feloldva ezzel a korábbi patthelyzetet.

6.3. Mobilalkalmazás

A projekt korai szakaszában, amikor még nem használtam a Unity Mapbox SDK-t, egyedi URL-ek segítségével próbáltam lekérni térképadatokat. Azonban a beolvasás valamiért nem működött megfelelően. Hosszabb hibakeresés során lépésről lépésre végigmentem az URL különböző részein, hogy meghatározzam, mi okozza a problémát. Végül rájöttem, hogy a gond a földrajzi koordinátákkal – azaz a hosszúsági (longitude) és szélességi (latitude) értékekkel – kapcsolatos. A böngészőben az URL helyesen tartalmazta a pontokat tizedeselválasztóként, azonban a beolvasás során ezek a pontok valamilyen oknál fogva vesszővé alakultak. Ez problémát okozott, mivel az URL-ben a különböző paramétereket vesszők választják el egymástól, így a koordináták is szétcsúsztak, és az URL értelmezhetetlenné vált. A megoldást végül az jelentette, hogy a beolvasott koordináta-értékeken futtattam egy Replace() metódust, amely a vesszőket (',') visszaalakította pontra ('.'). Ezzel biztosítani tudtam, hogy az URL megfelelő formátumban és szerkezetben maradjon, és a térképadatok sikeresen betöltődjenek.

A kategóriák (az információ részlet) kiírása során azt tapasztaltam, hogy nem jelent meg az összes elérhető kategória – konkrétan az első kategória kimaradt, és csupán az utolsó kettő került megjelenítésre. A debugolás közben rájöttem, hogy a kategóriák listájának összeállítása nem közvetlenül a kategórianevek alapján történt, hanem a pinekhez rendelt színek sorrendje szerint. Ez azt eredményezte, hogy az első kategória kimaradt a megjelenítésből. A probléma megoldásaként a LINQ Distinct() metódusát alkalmaztam, amely biztosítja, hogy az ismétlődő elemek kizárásával minden egyes kategória csak egyszer szerepeljen a listában – így a teljes, duplikációktól mentes kategórialista helyesen jelenik meg.

Összefoglalás

A hatékony csapatmunka menetét nagy mértékben az tette lehetővé, hogy alapvetően minden funkció megvalósítása előzetes megbeszélést igényelt a csapat többi részével, a projekt felépítéséből adódóan. Ez különösen jellemző volt az API megvalósításakor, ahol gyakorlatilag egymással párhuzamosan, egyidejűleg valósítottuk meg a szükséges logikát.

Elsődleges továbbfejlesztési célunk a projekt lokalizációja, mivel jelen pillanatban mind a weboldal, mind a mobilalkalmazás teljes egészében angol nyelvű. Ezen felül szeretnénk megvalósítani a sötét témát a weboldalunkon, kényelmesebb felhasználó élményt és környezetet biztosítva ezzel.

A weboldallal kapcsolatban továbbá azt szeretnénk a jövőben megvalósítani, hogy 2 felhasználó egymással privát üzenetek formájában kommunikálhasson, mivel jelenleg ezt csak publikusan tehetik meg a weboldalon belül. Szerencsére a Laravel rendelkezik egy Chatify nevű rendszercsomaggal, amely ez könnyedén lehetővé tenné.

Egy másik megvalósítani kívánt funkciónk még az, hogy egy admin felhasználónak lehetősége lehessen hétköznapi felhasználóként is bejelentkezni a weboldalra. Ez a változtatás azonban módosításokat követelne meg az adatbázisunk szerkezetében, például egy különálló admin tábla létrehozása.

Ezt követően szeretnénk létrehozni egy szebb és letisztultabb kinézetű admin panelt, melyen lehetőség adódna különböző felhasználói aktivitások nyomon követésére, valamint a weboldal általános forgalmának monitorálására. A Laravel Filament nevű keretrendszerének segítségével ez a funkció is könnyedén megvalósítható.

A mobilapplikáció, abból eredően, hogy a Unity játékmotorban hoztuk létre, könnyen és nagy léptekben továbbfejleszthető. Szeretnénk egy nevezetességhez több információs lapot is létrehozni, valamint az interaktív térképet 3D-ben is megtekinthetővé fejleszteni.

Források

Laravel Dokumentáció: https://laravel.com/docs/11.x

Laravel Debugbar: https://github.com/barryvdh/laravel-debugbar

Mapbox: https://docs.mapbox.com/android/maps/guides/install/

W3schools: https://www.w3schools.com/

Frontendhez segítőanyagok: https://css-tricks.com/

Weboldal SVG-k: https://fonts.google.com/ és https://uiverse.io/

Mailtrap köszöntő email tesztelésére: https://mailtrap.io/

Laravel http kényszerítés: https://stackoverflow.com/questions/70367815/makelaravel-use-https-by-default

Adatbázis források: https://hu.wikipedia.org/wiki/Balassagyarmat_szobrai_%C3%A9s_eml%C3%A9km%C5%B1vei

Route model binding: https://stackoverflow.com/questions/74277663/route-model-binding-in-laravel

Middleware-ek hozzáadása egy route-hoz: https://stackoverflow.com/questions/38039159/how-to-assign-middleware-to-routes-in-laravel-better-way

Nem létező kontroller hiba: https://stackoverflow.com/questions/63807930/error-target-class-controller-does-not-exist-when-using-laravel-8

Mass assignment: https://stackoverflow.com/questions/22279435/what-does-mass-assignment-mean-in-laravel

Profilkép cseréje Storage-ban: https://stackoverflow.com/questions/45065039/laravel-5-4-how-to-delete-a-file-stored-in-storage-app

Pivot táblák: https://www.youtube.com/watch?v=V5xINbA-z9o

Autorizáció (Gate-ek, Policy-k): https://www.youtube.com/watch?v=M1HMtm6hj5Q

Redirectelés paraméterekkel: https://stackoverflow.com/questions/21079280/laravel-redirect-from-controller-to-named-route-with-params-in-url

Validátor használta hibák megjelenítésére JSON-ben: https://stackoverflow.com/questions/67369333/validation-in-laravel-8

Keresősáv értékének megtartása lapozás után: https://stackoverflow.com/questions/58142297/how-to-append-query-string-to-pagination-links-using-laravel

Adatbázis seedelése sql fájlokkal: https://stackoverflow.com/questions/37369452/use-laravel-seed-and-sql-files-to-populate-database

CSRF tokenek kikapcsolása: https://stackoverflow.com/questions/

78279024/laravel-11-disable-csrf-for-a-route

Unity csatlakoztatása adatbázishoz: https://www.youtube.com/watch?v=khmqwgQVv2A&list=PLZuQHy0K6unWIOJ-lXAYWR9du-xTXJsd5

Unity UI Rounded Corners package: https://github.com/ReForge-Mode/Unity_UI_Rounded_Corners/releases/tag/v3.0.2-candidate

Unity side menu: https://github.com/Keyiter/One-Minute-Unity/blob/%234-Side-Menu/Assets/Scripts/SideMenu.cs és https://www.youtube.com/watch?v=BKeva_IlUeE

Unity dark/light mode: https://www.youtube.com/watch?v=vhfzpKWWA-A

Ábrajegyzék